The Interdisciplinary Center,
Herzliya

Efi Arazi School of Computer Science

M.Sc. program - Research Track

DDoS and Cloud Auto-Scaling Mechanism

Submitted by
Mor Cohen Gadol (Sides)

Under the supervision of
Prof. Anat Bremler-Barr

M.Sc. dissertation, submitted in partial fulfillment of the requirements
for the M.Sc. degree, research track, School of Computer Science
The Interdisciplinary Center, Herzliya

April 2016

This work was carried out under the supervision of Prof. Anat Bremler-Barr from the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya.

I would like to thank my supervision, Prof. Anat Bremler-Barr, for guiding and supporting me over the years throughout the thesis work. You introduced me to the world of academic research, I did not know before. You have opened me professional opportunities and exciting experiences, and were always ready to help and provided me insightful discussions about my work.

I would also like to thank my co-author, Dr. Eli Brush, your discussion, ideas, and feedback have been absolutely invaluable.

Contents

Abstract

One of the best practices for Distributed Denial of Service (DDoS) resiliency in the cloud is the auto-scaling mechanism, where machines can be added and removed in an online manner to respond to fluctuating load. It is commonly believed that the auto-scaling mechanism translates the DDoS attacks into Economic Denial of Sustainability attack (EDoS). Rather than suffering from performance degradation up to a total denial of service the victim suffers only from economic damage incurred by paying the extra resources required to process the bogus traffic of the attack.

In this paper we analyze the 'Yo-Yo attack', an efficient attack on the auto-scaling mechanism. 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 causes significant performance degradation in addition to economic damage, while the attack is harder to detect and requires less resources from the attacker compared to traditional DDoS. We demonstrate the attack on Amazon EC2 [2], and analyze protection measures the victim can take by reconfiguring the auto-scaling mechanism. Our work shows that DDoS mitigation is a crucial component in cloud environment, and that auto-scaling is not a bullet-proof remedy.

Chapter 1

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, which can often be configured to occur automatically, according to customer-set thresholds.

The main purpose of 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) [4].

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 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, it is assumed that the auto-scaling is a good enough solution, since it ensures that the service will continue to run with good performance. Moreover, the economic damage is not a real deterrent for the victim, since the alternative remedies, such as DDoS scrubbers middleboxes or DDoS scrubber cloud services [6, 24] also cost the victim money.

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 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 usually up to a few minutes, 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.

We demonstrate the Yo-Yo attack on Amazon's cloud service under different configurations and analyze the damages. The attack requires inferring 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. As we know, we are the first to analyze such attacks deeply and our work helps to explain the recently reported behavior of attacks which come in repeated waves.

Auto-scaling mechanisms employ one of two common policy 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 load of the traffic 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.

We then discuss various auto-scaling parameters and their influence on the damage from the Yo-Yo attack. We show that, auto-scaling is a not bullet proof remedy against variations of DDoS attacks such as the Yo-Yo attack, and that DDoS mitigation is a crucial component also in the cloud.

The remainder of this paper is organized as follows. Section 2 describes

cloud scaling characteristics and existing attacks in the cloud area. Section 3 presents the Yo-Yo attack in detail. Section 4 introduces the related work. In Section 5 we model the attack, analyze it mathematically and compare it to the DDoS attack. In Section 6 we evaluate the Yo-Yo attack and assess the impact of our attack on a real cloud service infrastructure. In Section 7 we discuss possible defense strategies. In Section 8 we present our conclusions.

Chapter 2

Cloud Auto-Scaling

Auto-scaling is a cloud computing service feature that automatically adds or removes compute resources depending upon actual usage. Its main benefit is helping to cope with load during rush hours; It is also purported to save costs. While the main aim of auto-scaling is to deal with the natural changes in load, it is also used as a remedy for DDoS attacks[4]. In this section we describe common auto-scaling mechanisms in the cloud, emphasizing the aspects relevant to DDoS attack and to our analysis of the Yo-Yo attack.

Each cloud solution comes with its own auto-scaling engine: Heat in Openstack [3], autoscaler in Google Cloud [11], Azure Autoscale in Microsoft Azure [10] and auto-scaling in Amazon Elastic Compute Cloud (Amazon EC2) [2]. Basically, 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 experiment, 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 the 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 *discrete policy* the number of machines increases or decreases with a fixed, predefined number of machines. If the overload was not resolved the process is continued iteratively. In *adaptive policy* the number of machines increases or decreases differentially and adaptive to the system load.

Google cloud scaling is always adaptive [1]. The user sets the target criterion value, e.g., CPU utilization above some threshold for scale-interval, and the autoscaler makes scaling decisions proportionally and maintains that level without the user having to set rules.

In Amazon EC2 the user should specify configuration for the auto-scaling algorithm. [2] 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 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. This time is called the *Warming time* and, we denote it by W_{up}. Mao et al. [18] show that this time is between 1 and 13 minutes, depending on the infrastructure provider, OS server and other factors, in addition to the service initialization time required. Scaling down takes W_{down} to release all the resources, but might take less time, if the service does not have a long backup operation. The extra machines are usually loaded before the rush hours, to avoid the latency of warming time[23]. While this solution is useful in the case of predictable, daily fluctuations of load, it is obviously not applicable to DDoS attacks.

In Distributed-Denial-of-Service (DDoS), the attack is launched from multiple connected devices that are distributed across the Internet (for example a botnet) and flood a target with fake requests. Nowadays, a cloud environment usually has extensive resources, and dynamic allocation capability of resources. As a result, theoretically it is not possible to deny the service of a cloud permanently. However, in this paper we show that the attacker can still cause damage using the Yo-Yo attack.

We demonstrate here how auto-scaling helps to mitigate DDoS, and the impact of the different auto-scaling parameters. We use here a very simplified and basic model using parameters of medium-size e-commerce web site [5]. We assume the site has in steady state, an average rate of 10,000 request, with 10 machines. The site is attacked with additional 20,000 requests (triple

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

[2]Discrete policy carried out using 'Simple scaling' or single step of 'Step scaling', Adaptive policy carried out using multiple steps of 'Step scaling'

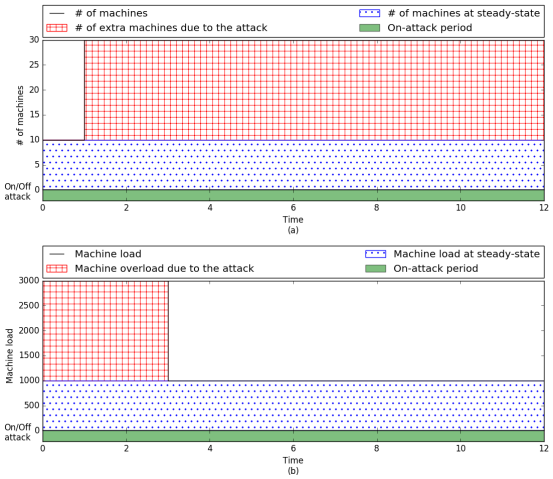

Figure 1: DDoS attack on system with adaptive scale policy

the average request rate). The user configured $I_{up} = I_{down} = 1minute$. The service is such that $W_{up} = W_{down} = 2minutes$. The impact of the DDoS attack is that the average load per machine increases by a factor of two (See Figure 1).

As Figure 1 shows, adaptive scaling action occurred after interval of I_{up}, and a few machines were added and ran for the duration of the attack. Therefore, system overload was experienced only for the duration of I_{up} plus W_{up}.

Figure 2 shows the impact of the same attack on an environment with discrete policy. As shown, same number of machines handle this overload in both environments, but the system overload is felt for longer duration under the discrete policy. Here the duration is estimated as a function of the number of machines to scale, multiple by a single machine scale time, I_{up} plus W_{up}.

As a result of the flood of fake requests in the DDoS attack, more instances of the service are launched. While the cloud services are provided as pay-per-use [15], the services under attack cost more, thus leading to an Economic Denial of Sustainability attack (EDoS)[14]. In our example the user would be requested to pay for 20 extra machines for all the duration of the attack.

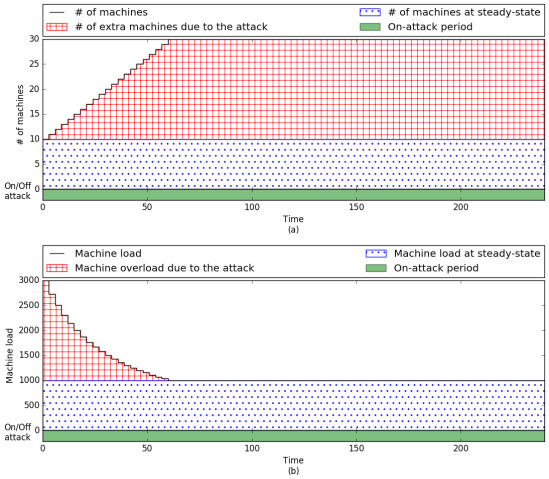

Figure 2: DDoS attack on system with discrete scale policy

Chapter 3

Yo-Yo Attack

In this section we present the Yo-Yo attack, which results in damages from both DDoS attack and EDoS attack.

The attack called *Yo-Yo attack* since the attacker oscillates from the *on-attack* phase to 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 has occurred. Once the attacker determines that scale down has occurred, the process is repeated.

Figure 3 demonstrates the Yo-Yo attack on an environment with discrete scaling policy on the same medium-size commercial site on which we demonstrate the DDoS attack. The attack configuration for both *on-attack* period and *off-attack* period set to 60 units time. The attack causes both economic and performance damage. During scale-up, the number of machines increases linearly and decreases linearly during scale-down, hence the economic damage. 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 number of linearly increasing number of machines. We note that it is harder to detect this attack, since its volume is smaller than that of a regular DDoS attack.

Clearly, the strength of the attack is partially determined by the ability of the attacker to determine when to switch between the two phases. In section 6.1 we show how an attacker can estimate the state of autoscaling by sending probe messages to the site, and observing when the site suffers from long response time and when the response time is short. This is in addition to 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

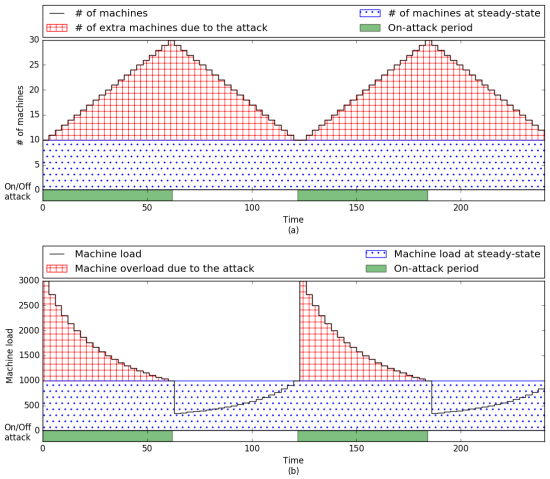

Figure 3: Yo-Yo attack on system with discrete scale policy

parameters and whether the auto-scaling up and down occur.

From discussions with administrator of cloud services, and also in Amazon's recommendation for handling DDoS [4], we have learned that some of the DDoS attacks come in repeated waves. We have no information as to whether these attacks are Yo-Yo attacks in the sense that they are tailored to exploit the auto-scaling mechanism. This paper sheds light on the impact of tailoring the waves of the attack to the auto-scaling mechanism.

Chapter 4

Related Work

Security, and DDoS prevention in particular, are curcial to every computing environment, especially in the cloud [31]. Several recent works and whitepapers [4, 21] recommend auto-scaling as a possible solution to mitigate DDoS attacks.

VivinSandar and Shenai [30] describe that a traditional DDoS attack can be transformed into an Economic Denial of Sustainability attack (EDoS) in the cloud environment. Somani et al. [26] describe that DDoS / EDoS attack in a cloud affecting everyone and all the environments.

Several work are trying to mitigate EDoS attacks [27, 9, 16]. Their proposed solutions are focused on classification of the clients and the ability to determine whether the user is legitimate or malicious bot. Mary et al.[20] propose a mitigation technique called DDoS&EDoS-Shield to protect against DDoS&EDoS attacks in cloud computing. All of these works ignore the effect of the autoscaling mechanism and attacks of the Yo-Yo type.

The basic form of the Yo-Yo attack and the general concept were presented as a brief announcement [25], but the authors ignored the impact of Yo-Yo on performance and the impact of different auto-scaling policies. This paper also provides not only a theoretical model but also discussion about possible remedies.

The Yo-Yo attack can also be seen as a type of Reduction of Quality (RoQ) [12] 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, load-balancing [13], and even TCP retransmission [17], are vulnerable to such attacks. The Yo-Yo attack is the first to show that the auto-scaling mechanism is vulnerable also.

There are several papers [8, 7, 28, 29] that discuss how to perform efficient auto-scaling, but they ignore the security aspect.

Chapter 5

Yo-Yo Analysis

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.

We follow our analysis with an evaluation of the impact of Yo-Yo attack on our use case example, representative of medium-size commercial site.

5.1 Auto-scale model environment

Consider a hosting environment that includes autoscaling with identical service machines behind a load balancer. Requests arrive with 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 a scale interval of I_{up} and I_{down} and warming time of W_{up} and W_{down}.

The auto-scaling policy might influence the autoscaling parameters and can also influence the parameters chosen by the attacker in the Yo-Yo attack. We denote by superscript letter 'd', parameters for the discrete policy, and by superscript letter 'a' the parameters for the adaptive policy. 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 application running on the machine.

5.2 Yo-Yo attack formalization

The Yo-Yo attack is composed of n cycles, and each cycle duration is T, 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}. Where k is 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 1 for notation summary.

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
n	Number of attack cycles	Attacker
T	Cycle duration	
$t_{on} \backslash t_{off}$	Time of *on-attack* phase and *off-attack* phase. $T = t_{on} + t_{off}$	
k	The power of the attack	

Table 1: Notation used in the model description

We assume that the attacker aims to optimize the damage, with the main goal of being active long enough to scale up to extra $k \cdot m$ machines, and a secondary goal of causing performance damage. In subsection 5.6, we discuss other possible attacks that aims to optimize the damage differently.

For the autoscaling to occur and exceed the policy threshold, two conditions must be met. First, the extra load of $k \cdot r$ should burden on the system such that the threshold for scaling is fulfilled, regardless the criterion (e.g., CPU utilization, traffic). Second, t_{on} should be greater than or equal to the scale-interval I_{up}.

We assume that in order to cope with additional $k \cdot r$ traffic, the autoscaling will scale up to all additional $k \cdot m$ machines. This assumption simplifies the analysis.

Under the discrete scaling policy, the on-attack time which is equal to the scale-up time directly depends on the number of machines to scale. In this policy, every $I^d_{up} + W_{up}$, only one machine is added. Thus [1]:

$$t^d_{on} = \quad k \cdot m \cdot (I^d_{up} + W_{up}) \tag{5.1}$$

The total duration of a cycle, including scaling up and scaling down $k \cdot m$ machines is:

$$T^d = \quad k \cdot m \cdot (I^d_{up} + W_{up} + I^d_{down} + W_{down}) \tag{5.2}$$

[1]This is an upper bound, that we use in order to simplify the analysis. To be more exact it is enough for the attacker to trigger the scale up of $k \cdot m$ machines be activate for $(k \cdot m) \cdot (I^d_{up} + W_{up}) - W_{up}$.

Under the adaptive scaling policy, after the attack has been perpetrated for I^a_{up} time, the scaled up is trigger of the additional $k \cdot m$ machines. Thus, in the adaptive policy:

$$t^a_{on} = \quad I^a_{up} \tag{5.3}$$

and

$$T^a = \quad I^a_{up} + W_{up} + I^a_{down} + W_{down} \tag{5.4}$$

Note that here the duration of the on-attack phase and the duration of a cycle do not depend on the number of machines to scale.

Figure 4 shows our use case under the adaptive scaling policy with $n = 2$. The parameter values of the use case are summarized in Table 2.

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

Table 2: Parameters values of use case example that is a representative of medium-size commercial site

5.3 Damage analysis

In this section we derive closed formulas to quantify the damage inflicted by an attacker and the cost of the attack for the attacker. We define D^{YoYo}_p, the performance damage caused by the attack and assess it as the average absolute extra load on the system during the total attack time. We define D^{YoYo}_e, the economic damage caused by the attack and assess it as the average absolute extra machines running in the system during the total attack time.

For each of the absolute damage above, we derive the formula of the relative damage as a function of k, $RD^{YoYo}_p(k), RD^{YoYo}_e(k)$. Relative damage is defined as the ratio between the damage following the attack and the corresponding value at steady state.

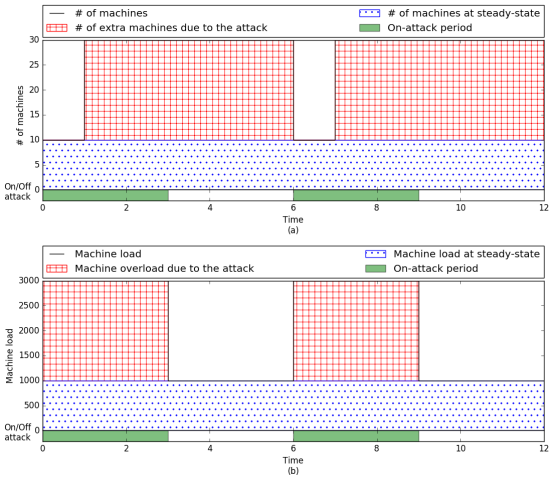

Figure 4: Yo-Yo attack on system with adaptive scale policy

5.3.1 Yo-Yo attack with discrete scaling policy

Figure 3 shows the damage under the Yo-Yo attack with the discrete policy. The economic damage is estimated as the additional number of machines as a result of the attack and their running duration. Note that in the discrete policy the number of machines increases and decreases linearly. Note that in the scaling up of a machine, during the warming time, W_{up}, the user pays for the machine but the machine is not fully activated, and thus does not help to reduce the load. Thus:

$$\begin{aligned} D_e^{YoYo^d} &= \\ &= \sum_{i=m}^{m\cdot(k+1)}(i-m)\cdot\frac{(I_{up}^d+W_{up}+I_{down}^d+W_{down})\cdot n}{T^d\cdot n} = \\ &= \frac{k\cdot m\cdot(k\cdot m+1)}{2}\cdot\frac{I_{up}^d+W_{up}+I_{down}^d+W_{down}}{T^d} \end{aligned} \tag{5.5}$$

By assigning the value of T^d (see Equation 5.2), and normalize it by the number of machine in the steady state, we receive:

$$RD_e^{YoYo^d}(k) = \frac{\frac{k\cdot m\cdot(k\cdot m+1)}{2}\cdot\frac{1}{k\cdot m}}{m} = \frac{k\cdot m+1}{2\cdot m} \approx \frac{k}{2} \tag{5.6}$$

The performance damage is estimated as the overload on a machine as a result of the attack [2]. In the steady state the load on a machine is $\frac{r}{m}$. During

[2]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.

the attack the general load $r \cdot (k+1)$ is divided on a linearly increasing number of machines. This results in a harmonic series. [3]

$$\begin{aligned} D_p^{YoYo^d} &= \\ &= \sum_{i=m}^{m\cdot(k+1)} \left(\frac{r\cdot(k+1)}{i} - \frac{r}{m}\right) \cdot \frac{(I_{up}^d + W_{up})\cdot n}{T^d \cdot n} = \\ &\approx r \cdot ((k+1)\cdot ln(k+1) - k - \frac{1}{m}) \cdot \frac{(I_{up}^d + W_{up})}{T^d} \end{aligned} \tag{5.7}$$

In order to simplify the equation of relative damage, here we made the simplified assumption that $I_{up}^d = I_{down}^d$ and $W_{up} = W_{down}$. We then assigning the value of T^d (see Equation 5.2), and normalized it by the load on a machine in the steady state, obtaining,

$$\begin{aligned} RD_p^{YoYo^d}(k) &\approx \frac{r\cdot((k+1)\cdot ln(k+1) - k - \frac{1}{m})}{\frac{r}{m}} \cdot \frac{1}{2\cdot k\cdot m} \approx \\ &\approx \frac{(k+1)\cdot ln(k+1) - k}{2\cdot k} \approx \frac{ln(k+1)-1}{2} \end{aligned} \tag{5.8}$$

5.3.2 Yo-Yo attack with adaptive scaling policy

Figure 4 shows the damage under the Yo-Yo attack with the adaptive policy. In adaptive scale policy, all the machines are scaled at once. Thus the economic damage is:

$$\begin{aligned} D_e^{YoYo^a} &= (m \cdot (k+1) - m) \cdot \frac{(T^a - I_{up}^a)\cdot n}{T^a \cdot n} = \\ &= k \cdot m \cdot \frac{T^a - I_{up}^a}{T^a} \end{aligned} \tag{5.9}$$

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

$$RD_e^{YoYo^a}(k) = \frac{k\cdot m \cdot \frac{T^a - I_{up}^a}{T^a}}{m} = k \cdot \frac{T^a - I_{up}^a}{T^a} \tag{5.10}$$

The performance damage is experienced until the machines are running and functioning, i.e, until I_{up}^a plus W_{up}.

$$\begin{aligned} D_p^{YoYo^a} &= \left(\frac{r\cdot(k+1)}{m} - \frac{r}{m}\right) \cdot \frac{(I_{up}^a + W_{up})\cdot n}{T^a \cdot n} = \\ &= \frac{k\cdot r}{m} \cdot \frac{I_{up}^a + W_{up}}{T^a} \end{aligned} \tag{5.11}$$

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

$$RD_p^{YoYo^a}(k) = \frac{\frac{k\cdot r}{m} \cdot \frac{I_{up}^a + W_{up}}{T^a}}{\frac{r}{m}} = k \cdot \frac{I_{up}^a + W_{up}}{T^a} \tag{5.12}$$

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

5.4 Cost and potency analysis

For the above Yo-Yo attack, we define $P_e^{YoYo}(k)$, the attack economic potency, to be the ratio between the relative economic damage $RD_e^{YoYo}(k)$ for customers caused by attack with power k and $C^{YoYo}(k)$, the cost of mounting such an attack.

$$P_e^{YoYo}(k) = \frac{D_e^{YoYo}(k)}{C^{YoYo}(k)} \tag{5.13}$$

We define similar notations for performance damage. Clearly, an attacker would be interested in maximizing the damage per unit cost – i.e., maximizing the attack potency.

In Yo-Yo attack, the attack cost is directly affected by the power of attack k and t_{on} period relative to attack cycle length.

$$C^{YoYo}(k) = k \cdot \frac{t_{on}}{T} \tag{5.14}$$

In our use case with discrete policy the attack economic potency is:

$$P_e^{YoYo^d}(k) \approx \frac{\frac{k}{2}}{k \cdot \frac{1}{2}} = 1 \tag{5.15}$$

The attack performance potency is:

$$P_p^{YoYo^d}(k) \approx \frac{\frac{ln(k+1)-1}{2}}{k \cdot \frac{1}{2}} = \frac{ln(k+1)-1}{k} \tag{5.16}$$

In our use case with adaptive policy the attack economic potency is:

$$P_e^{YoYo^a}(k) \approx \frac{k \cdot \frac{T^a - I_{up}^a}{T^a}}{k \cdot \frac{I_{up}^a + W_{up}}{T^a}} = \frac{T^a - I_{up}^a}{I_{up}^a + W_{up}} \tag{5.17}$$

The attack performance potency is:

$$P_p^{YoYo^a}(k) = \frac{k \cdot \frac{I_{up}^a + W_{up}}{T^a}}{k \cdot \frac{I_{up}^a + W_{up}}{T^a}} = 1 \tag{5.18}$$

5.5 Discussion

Table 3 summarizes and compares our model with different attacks and environments. The table also shows the DDoS attack with and without autoscaling. The bottom row indicates whether the system stabilizes to the steady state. In DDoS the system reaches a steady state, without auto-scaling the

	DDoS	DDoS with autoscaling	Yo-Yo Discrete policy	Yo-Yo Adaptive Policy
T			$k \cdot m \cdot (I^d_{up} + W_{up} + I^d_{down} + W_{down})$	$I^a_{up} + W_{up} + I^a_{down} + W_{down}$
t_{on}			$k \cdot m \cdot (I^d_{up} + W_{up})$	$I^a_{up} + W_{up}$
Attack Cost	k	k	$k \cdot \frac{t_{on}}{T}$	$k \cdot \frac{t_{on}}{T}$
Relative Economic Damage $RD_e(k)$	0	k	$\approx \frac{k}{2}$	$k \cdot \frac{T - I^a_{up}}{T^a}$
Relative Performance Damage $RD_p(k)$	k	0	$\approx \frac{ln(k+1)-1}{2}$	$k \cdot \frac{I^a_{up}+W_{up}}{T^a}$
Attack Economic Potency $P_e(k)$	0	1	1	$\frac{T^a - I^a_{up}}{I^a_{up}+W_{up}}$
Attack Performance Potency $P_p(k)$	1	0	$\approx \frac{ln(k+1)-1}{k}$	1
Stabilization	✓	✓	✗	✗

Table 3: Model Summary

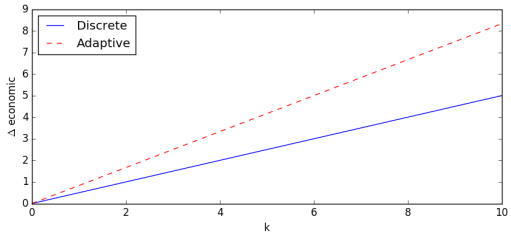

Figure 5: Relative economic damage of YoYo attack with different scaling policy as function of the attack's power.

relative performance damage is equal to k, while with autoscaling the relative economic damage is equal to k. In the Yo-Yo attack, the attacked system suffers from combined economic and performance damages.

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 shows the relative economic damage case as a function of k for the two auto-scaling policies. Figure 6 shows the relative performance damage as a function of k for the two auto-scaling policies. Although the adaptive policy responds more quickly to changes, we can see, in both graphs, that the relative damage is larger for the adaptive policy than for the discrete policy. Therefore, it can be concluded that the adaptive policy type is more vulnerable to such attacks.

In order to receive some sense of the actual damage inflicted by Yo-Yo attack, we analyze the results of the representative use case of a medium-size commercial site (see Table 2 for the parameters of the site). Table 4 summarizes our findings. With attack cost of 50% compared to the DDoS

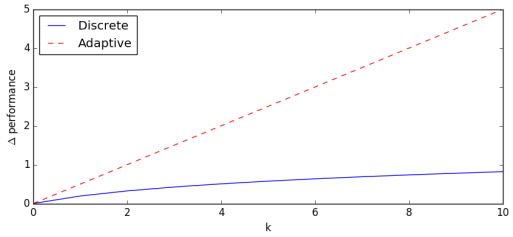

Figure 6: Relative performance damage of YoYo attack with different scaling policy as function of the attack's power.

attack, the damage with the adaptive policy is an extra of 160% machines as opposed to 200% in DDoS, and an extra load on each machine of 100% to 0% in DDoS. We note that one of the main reasons the Yo-Yo attack is so harmful under adaptive scaling is the extensive performance damage that results from the relative long W_{up} time (see the discussion of the warning time in Section 7).

Attack Scale policy	DDoS Adaptive	Yo-Yo Adaptive	Yo-Yo Discrete
$[\frac{t_{on}}{T}, \frac{t_{off}}{T}]$	$[1, 0]$	$[\frac{1}{2}, \frac{1}{2}]$	$[\frac{1}{2}, \frac{1}{2}]$
$C(2)$	2	1	1
$RD_e(2)$	$\approx 200\%$	$\approx 166\%$	$\approx 100\%$
$RD_p(2)$	$\approx 0\%$	$\approx 100\%$	$\approx 30\%$
$P_e(2)$	$\approx 100\%$	$\approx 166\%$	$\approx 100\%$
$P_p(2)$	$\approx 0\%$	$\approx 100\%$	$\approx 30\%$

Table 4: Use-case analysis

Note that any attack, of any kind, requires botnet attack force proportional to the attacked service; thus large-scale services are harder to attack. Nowadays, the cost of attack often translates to a real cost of renting the army of botnets. The fact that the Yo-Yo attack reduces the total cost of the attack, opens the door to attackers to be harmful to larger services.

5.6 Additional variants of the Yo-Yo attack

Up to now we analyzed the Yo-Yo attack under the assumption that the attacker aims to inflict both types of damages: performance and economic.

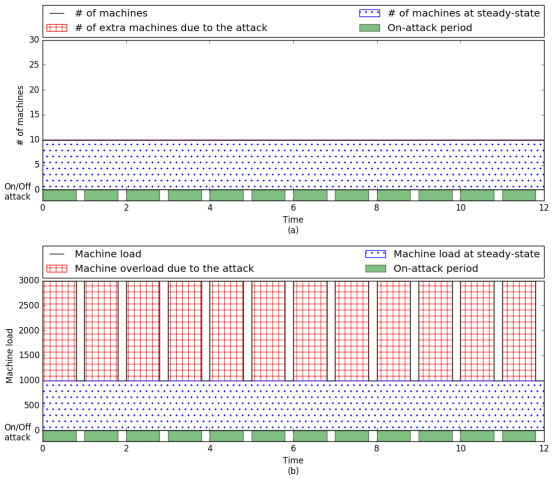

Figure 7: Maximum performance damage

In this section, we show that attacker can compose different attacks, with different damages. We present a variation of the Yo-Yo attack that maximizes the performance damage, and another variation that maximizes the economic damage.

It is important to emphasize that in the Yo-Yo attack that was presented so far, the precision of the on and off attack periods was not essential. The attack will take place even if the malicious bots will start sending around the same time, and not exactly at the same second. However, in the variants we present in this section, the precision of the on and off attack periods is more important and therefore these attacks are more difficult to carry out.

5.6.1 Variant that maximize performance damage

In this case the attacker sends bursts with duration close to the scale-up interval but smaller, i.e, t_{on} is chosen to be the maximum t_{on} such that $t_{on} < I_{up}$. Thus the scale-up process does not occur, but throughout t_{on}, a drop in performance is experienced. Note that in this case, in order to optimize the performance damage, the attacker needs to know the exact value of I_{up}, which is more difficult to learn under the discrete policy (see discussion in Subsection 6.1).

Figure 7 simulates this attack on our use case example environment with the adaptive policy but the graph for the discrete policy will be the same.

Damage analysis:

In order to realize the attack requires to set $t_{on} < I_{up}$.

Note that in this variant of Yo-Yo attack, scale-up process does not occur, thus there is no economic damage:

$$D_e^{YoYo_p^a} = 0 \tag{5.19}$$

$$RD_e^{YoYo_p^a}(k) = 0 \tag{5.20}$$

However, the performance damage is significant and close to the maximum:

$$\begin{aligned} D_p^{YoYo_p^a} &= (\tfrac{r\cdot(k+1)}{m} - \tfrac{r}{m}) \cdot \tfrac{t_{on}}{I_{up}^a} = \\ &\approx \tfrac{k\cdot r}{m} \cdot 1 = \tfrac{k\cdot r}{m} \end{aligned} \tag{5.21}$$

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

$$RD_p^{YoYo_p^a}(k) \approx \tfrac{\frac{k\cdot r}{m}}{\frac{r}{m}} = k \tag{5.22}$$

In order to receive some sense of the actual damage inflicted by this variant of Yo-Yo attack, we analyze the results of our use case example (see Table 2 for the parameters of the site use case). For example, we chose $t_{on} = 0.8 \cdot I_{up}^a$ in order t_{on} to be smaller than I_{up}^a, thus the attack cost is 80% compared to the DDoS attack. The performance damage in this case, with the adaptive policy is an extra load of 160% on each machines as opposed to 200% in DDoS. Of course that these numbers are changing depending on the chosen t_{on}.

5.6.2 Variant that maximize economic damage

In order to maximize the economic damage, the attacker does not let the machines scale back down after the initial scale up. Thus after the initial t_{on}, which will be long enough to finish scaling all the required machines, the attacker sends small bursts at a frequency of less than I_{down}, such that $t_{off} < I_{down}$ and the scale-down conditions will not occur.

Figure 8 simulates this attack on our use case environment with adaptive scaling. The discrete case is similar but with a longer first burst.

This variant of the Yo-Yo attack inflicts the same economic damage as the DDoS attack, but the cost of the attack is lower (see Figure 1 for reference).

Damage analysis:

In order to realize the attack requires to set the initial t_{on} to be greater than the scale-up interval I_{up}, after that the rest of t_{on} are not significant for the

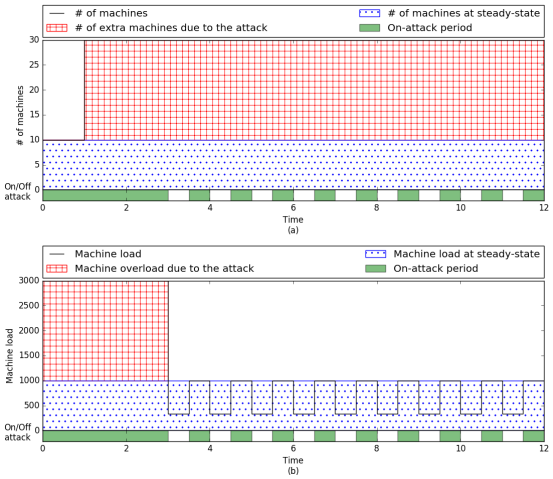

Figure 8: Maximum economic damage

economic damage. Additionally requires to set $t_{off} < I_{down}$ during the entire attack, in order to prevent scale-down.

In this analysis we relates $T^{YoYo_e^a}$ as the entire time of attack that maximize economic damage on system with adaptive policy. (Note that in this case $T^{YoYo_e^a}$ is not equal to $t_{on} + t_{off}$ as the other analyses).

$$\begin{aligned} D_e^{YoYo_e^a} &= (m \cdot (k+1) - m) \cdot \frac{T^{YoYo_e^a} - I_{up}^a}{T^{YoYo_e^a}} = \\ &= k \cdot m \cdot \frac{T^{YoYo_e^a} - I_{up}^a}{T^{YoYo_e^a}} \end{aligned} \tag{5.23}$$

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

$$\begin{aligned} RD_e^{YoYo_e^a}(k) &= \frac{k \cdot m \cdot \frac{T^{YoYo_e^a} - I_{up}^a}{T^{YoYo_e^a}}}{m} = \\ &= k \cdot \frac{T^{YoYo_e^a} - I_{up}^a}{T^{YoYo_e^a}} \approx k \end{aligned} \tag{5.24}$$

Note that in this case, in order to optimize the cost against performance damage, the attacker needs to know the exact value of I_{down}, which is more difficult to learn under the discrete policy (see discussion in Subsection 6.1).

The performance damage is felt only before scale-up process end, then, the performance damage is chaged from t_{on} periods to t_{off} periods, but anyway

not greater than the steady state.

$$\begin{aligned} D_p^{YoYo_e^a} = & \ (\tfrac{r\cdot(k+1)}{m} - \tfrac{r}{m}) \cdot \tfrac{I_{up}^a + W_{up}}{T^{YoYo_e^a}} + (\tfrac{r(k+1)}{m(k+1)} - \tfrac{r}{m}) \cdot \tfrac{t_{on}}{T^{YoYo_e^a}} + (\tfrac{r}{m(k+1)} - \tfrac{r}{m}) \cdot \tfrac{t_{off}}{T^{YoYo_e^a}} = \\ = & \ \tfrac{k\cdot r}{m} \cdot \tfrac{I_{up}^a + W_{up}}{T^{YoYo_e^a}} + 0 \cdot \tfrac{t_{on}}{T^{YoYo_e^a}} - \tfrac{r\cdot k}{m(k+1)} \cdot \tfrac{t_{off}}{T^{YoYo_e^a}} \approx \\ \approx & \ \tfrac{k\cdot r}{m} \cdot \tfrac{I_{up}^a + W_{up}}{T^{YoYo_e^a}} \end{aligned} \tag{5.25}$$

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

$$RD_p^{YoYo_e^a}(k) \approx \ \frac{\frac{k\cdot r}{m} \cdot \frac{I_{up}^a + W_{up}}{T^{YoYo_e^a}}}{\frac{r}{m}} = k \cdot \frac{I_{up}^a + W_{up}}{T^{YoYo_e^a}} \tag{5.26}$$

Because we relates $T^{YoYo_e^a}$ as the entire time of attack and scale-up occurs only once at the beginning, we can see that as $T^{YoYo_e^a}$ is larger, so $RD_p^{YoYo_e^a}(k)$ close to 0.

While analyzing the results of our use case example in this variant of Yo-Yo attack, we chose $t_{off} = 0.5 \cdot I_{down}^a$ in order t_{off} to be smaller than I_{down}^a, thus the attack cost is 50% compared to the DDoS attack. The damage with the adaptive policy is the same as DDoS, an extra of 200% machines, and an extra load on each machine close to 0%. In this variant, the attack cost is changing depending on the chosen t_{off}, but the damage remains more or less the same.

Chapter 6

Experimental Evaluation

In this section, we show proof of concept of the Yo-Yo attack on Amazon cloud service.

Our environment in Amazon consists of a simple HTTP server, front-end side stateless without back-end.[1] In our experiment we choose the CPU utilization criterion, and correspondingly implement the http server where each connection requires high CPU consumption. We use Amazon CloudWatch [1] to monitor our instances and manage the auto-scale group for scale triggers on high or low CPU utilization. It is important to note that the http-based implementation sometimes causes http errors (errors from type NoHttpResponseException are usually thrown when the server is under heavy load and may be able to receive requests but not be able to process them). These situations are not included in the average response calculation, but are represented in a separate plot of error percent. In addition, according to the http protocol, the client tries to resend the request in case of an error. The large number of retries further increases the error percent.

In our experiments, we begin with a single machine, the minimum number allowed, and set the maximum number of machines high enough to not restrict us in practice. In the steady state our http server handles 10 requests per second, and in the on-attack phase, the 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. See results in Figure 9.

Under the adaptive policy type we configure scaling with steps, which scales up to 4 machines, on a scale from 50% to 80% and scales down, on a

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

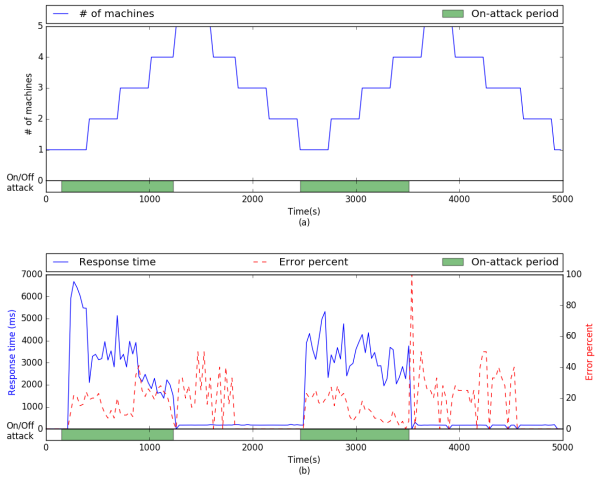

Figure 9: Yo-Yo attack on system with discrete scale policy

scale from 10% to 30%. Figure 10 shows the Yo-Yo attack on a system with the adaptive scaling policy.

The figures in adaptive and discrete policy clearly shows that in both cases there is economic damage, and the number of machines increases as the model predict. The figures of response time are similar in shape to the graphs of load per machine that were produced using our model. Clearly there is a correlation between the load on the machine and the repose 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 [13], and second, due to the http request there is still overload due to the http retries after the attacker has stopped sending traffic. We want to model these phenomena in the future.

In addition, the performance degregation in our model got even worse while using HTTPS [22], and other recommendation of Amazon for DDoS mitigation like CloudFront and Route 53 are not relevant to the autoscaling vulnerability discussed here or cannot help, since the requests require dynamic pages and cannot use CDN.

6.1 Detecting scale policy

In this subsection we show how an attacker can approximate the autoscaling state and configuration in order to optimize the attack damage. Note that for the basic Yo-Yo attack, which optimizes the economic and performance damage, the attacker only needs to know when the scale-up and scale-down ended.

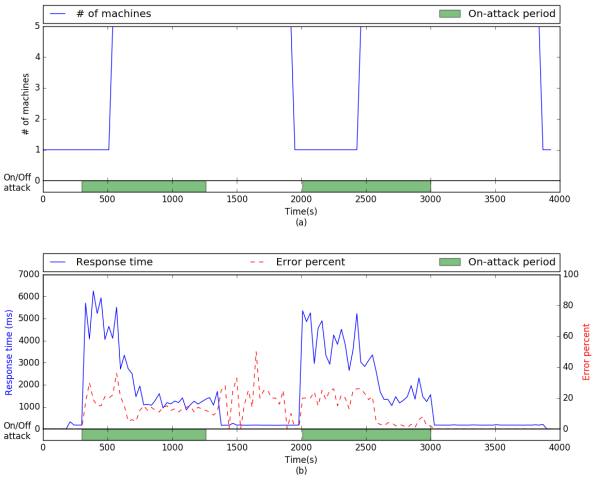

Figure 10: Yo-Yo attack on system with adaptive scale policy

In order to detect when the scale-up has ended, the attacker can send probe requests, and check their response time. From Figure 9 it is clear that if the scale-up process has not yet ended the probe requests will suffer from high response time and high packet loss. 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. We use the fact that the impact of the extra load will be immediately noticed in the response time (in our experiments it was noticeable in less than 1 second) .

In a pure adaptive policy, all the extra machines can be up or down, and there it never happens that only part of the extra machines remain up or down. If the response is smaller than $1000ms$, then all the extra machines are still up; Otherwise, the scale-down process has ended.

Under the discrete policy is much more delicate, since the process of scale-down is a continuous process where the extra machines are gradually shut down. However, as our experiments show, there is a clear correlation between the number of extra machines and the response time. Thus the attacker would need to first learn the response time when there is a burst in the load but no extra machine. The attacker can check this before lunching the attack.

The other variations of Yo-Yo attacks (described in Subsection 5.6), require more intricate knowledge of the autoscaling configuration. Specifically,

the Yo-Yo attack with maximum performance damage requires learning the I_{up}, and the Yo-Yo attack with maximum economic damage requires learning the I_{down}. An attacker cannot distinguish the exact scaling interval I_{up} and I_{down} from outside. However, in the adaptive policy, the attacker can learn the scale-up time, which is equal to $I_{up} + W_{up}$, and thus have some estimate as to I_{up}. The attacker can try to find the value of I_{up} by trying to deploy the attack in bursts with different intervals between them and observe whether the scale-up has occurred, with the goal of finding the maximum frequency of bursts that would not triggering scale-up. Similarly, the attacker can try to estimate the I_{down} from the scale-down interval. The discrete policy is much more complicated, since the scale-up interval is proportional to the $I_{up} + W_{up}$ multiplied by the number of machines loaded during the scale-up process. This scenario is part of our future work.

Note that for some of the techniques, the attacker would like to distinguish between the adaptive and discrete scaling policies. We think that the response time of probe messages can be used by the attacker to reveal the policy. This because the response time varies more under the discrete policy (see Figure 9), due to the fact that the scaling up is done gradually. This is also part of our future work.

Chapter 7

Discussions About Defense Strategy For The Yo-Yo Attack

In this section we discuss possible mitigation techniques against 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. Thus, by doing so, we are not relaying exclusivly on the auto-scaling mechanism as a remedy, or at least not as the only remedy. Note that 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.

Here, we focus on auto-scaling configuration changes which can be used to further decrease the problem.

- **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 scale up the machines, especially in discrete model. During the scale-up warming time, the user pays for the machine but the machine does not help to cope with the load. Thus one effort should be to minmize the warming time. Intensive efforts have been made to reduce that warming time [19] but this is a challenging task, since it takes time for the application to set up. As reported in [18], this duration can reach more than 13 minutes for some services, and usually it is at least a few minutes. Additional recommendation offer keep unused capacity available for quick response, in this approach the user pays for the unused machine all the time.

  Another recommendation is not to rush to scale down the machines immediately after the attack stops. In other words, the idea is to

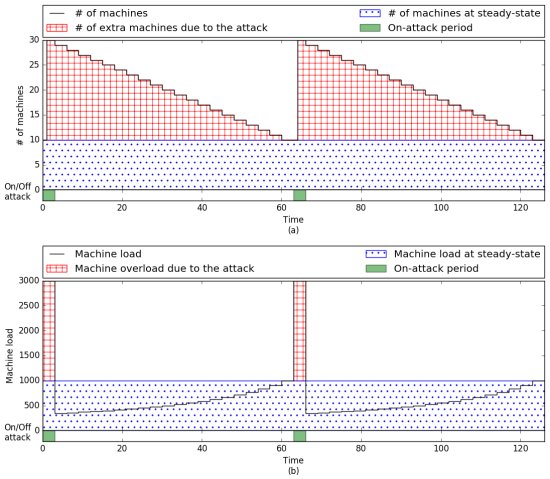

Figure 11: Yo-Yo attack on system with adaptive scale up policy and discrete scale down policy

configure a longer I_{down}. This is also part of the recommendation of Amazon [4], 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.

Hence, we recommend a solution with adaptive scale up and discrete scale down rules. Figure 11 shows a simulation of the Yo-Yo attack cause on such an environment. The on-periods of the Yo-Yo attack cause scale-up action and the system responds quickly to overload, but it takes a long time until the machines scale down and the attacker can return to attack again.

The phrase "scale up early, scale down slowly" appeared in Netflix's recommendation [23] for auto-scaling in Amazon EC2 environment not in the context of attacks. This solution does reduce the performance damage, but increases the economic damage.

- **Restrictions:** Resource limitation.

  In order to avoid unexpected expenses, it is possible to set the maximum number of machines allowed to scale and limit the system resources. An attacker could not cause economic damage beyond this limit. In our use case example, the maximum number of machines that were in use is 30. While limiting this number to be less than 30,

the amount of resources used during the attack decreases and thus, reducing the service cost. However, resource limitation also restricts the autoscaler operating range and may cause denial-of-service, so attention is required when employing this strategy.

There is a trade-off between paying for high service cost (while scaling down slowly) or suffering low performance (while limiting resources). Each system administrator can configure his system differently depending on what he compromise. An interesting and challenging problem is to find the suitable defense strategy for each service, and analyze its behavior, which is a subject of a forthcoming work.

Chapter 8

Conclusions And Future Work

In this work we shed light on the potential of exploiting the auto-scaling mechanism to perform an efficient attack that impacts the cost and the quality of a service. 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. However, we believe that autoscaling mechanism alone is not enough, alternative remedies, such as DDoS scrubbers middleboxes or DDoS scrubber cloud service [6, 24], are still necessary.

The trend of virtualization, and cloud services, also impact the area of network services, which are part of the Network Function Virtualization (NFV) revolution. While part of the promise of NFV is that network services, among them middleboxes, will enjoy the auto-scaling property, our work shows that special care should be taken to prevent attackers from using these mechanisms to reduce the value of NFV and attack crucial services in the network.

Finally, we quote from Netflix's blog[23]: "Auto scaling is a very powerful tool, but it can also be a double-edged sword. Without the proper configuration and testing it can do more harm than good". Our work sheds new light on problematic aspects of auto-scaling.

Bibliography

[1] Amazon cloudwatch. https://aws.amazon.com/cloudwatch/.

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

[3] Heat: Openstack orchestration. https://wiki.openstack.org/wiki/Heat.

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

[5] Private discussion with site administrators, 2016.

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

[7] Ali-Eldin, A., Kihl, M., Tordsson, J., and Elmroth, E. 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), ACM, pp. 31–40.

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

[9] Baig, Z. A., and Binbeshr, F. 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* (2013), IEEE, pp. 346–353.

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

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

[12] Guirguis, M., Bestavros, A., Matta, I., and Zhang, Y. Reduction of quality (RoQ) attacks on internet end-systems. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (2005), vol. 2, IEEE, pp. 1362–1372.

[13] Guirguis, M., Bestavros, A., Matta, I., and Zhang, Y. Reduction of quality (RoQ) attacks on dynamic load balancers: Vulnerability assessment and design tradeoffs. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE* (2007), IEEE, pp. 857–865.

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

[15] Jellinek, R., Zhai, Y., Ristenpart, T., and Swift, M. A day late and a dollar short: The case for research on cloud billing systems. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)* (2014).

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

[17] Kuzmanovic, A., and Knightly, E. W. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (2003), ACM, pp. 75–86.

[18] Mao, M., and Humphrey, M. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (2012), IEEE, pp. 423–430.

[19] Martins, J., Ahmed, M., Raiciu, C., and Huici, F. Enabling fast, dynamic network processing with clickos. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 67–72.

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

[21] Miao, R., Yu, M., and Jain, N. NIMBUS: cloud-scale attack detection and mitigation. In *ACM SIGCOMM Computer Communication Review* (2014), vol. 44, ACM, pp. 121–122.

[22] Naylor, D., Finamore, A., Leontiadis, I., Grunenberger, Y., Mellia, M., Munafò, M., Papagiannaki, K., and Steenkiste, P. The cost of the s in https. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies* (2014), ACM, pp. 133–140.

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

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

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

[26] Somani, G., Gaur, M. S., and Sanghi, D. 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.

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

[28] Stillwell, M., Schanzenbach, D., Vivien, F., and Casanova, H. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and distributed Computing 70*, 9 (2010), 962–974.

[29] Vaquero, L. M., Rodero-Merino, L., and Buyya, R. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review 41*, 1 (2011), 45–52.

[30] VivinSandar, S., and Shenai, S. Economic denial of sustainability (EDoS) in cloud services using http and xml based DDoS attacks. *International Journal of Computer Applications 41*, 20 (2012).

[31] Yu, S., Tian, Y., Guo, S., and Wu, D. O. Can we beat DDoS attacks in clouds? *Parallel and Distributed Systems, IEEE Transactions on 25*, 9 (2014), 2245–2254.

תקציר

אחת משיטות העבודה המומלצת להגברת חסינות מהתקפות מניעת שירות מבוזרות (DDoS) בענן היא שימוש במנגנון auto-scaling, אשר מאפשר הוספה והורדה של מכונות באופן אוטומטי על מנת להגיב לשינויים בעומס המערכת. ההערכה הרווחת היא כי מנגנון שינוי קנה מידה אוטומטי מתרגם התקפות מסוג DDoS להתקפות מסוג מניעת קיימות כלכלית (EDoS). במקום פגיעה ברמת הביצועים ושלילה מוחלטת של השירות, הקורבן סובל מנזק כלכלי בלבד הנוצר מתשלום על משאבים נוספים הנדרשים כדי לעבד את התעבורה המזויפת של ההתקפה.

בעבודה זאת ננתח את התקפת ה-״יו-יו״, התקפה יעילה על מנגנון ה-auto-scaling. בהתקפת יו-יו, התוקף שולח התפרצויות מחזוריות של עומס יתר אשר גורמות למנגנון ה-auto-scaling להיטלטל בין מצב של הגדלה כמות המכונות להקטנת כמות המכונות. התקפת יו-יו גורמת להקטנה משמעותית ברמת הביצועים במערכת בנוסף לנזק כלכלי, בעוד שההתקפה קשה יותר לזיהוי ודורשת פחות משאבים מהתוקף בהשוואה להתקפות מניעת שירות (DDoS) מסורתיות. נדגים את ההתקפה על סביבת Amazon EC2 [2], וננתח מדידות שונות לצורכי הגנה אשר הקורבן יכול לבצע על ידי שינויי קונפיגורציה במנגנון ה-auto-scaling. העבודה מראה כי הפחתה והגנה מהתקפות מניעת שירות (DDoS) היא מרכיב חיוני בסביבת ענן, וכי מנגנון auto-scaling לא מספק פתרון מלא לבעיה.

עבודה זו בוצעה בהדרכתה של פרופ׳ ענת ברמלר-בר מבי״ס אפי ארזי למדעי המחשב, המרכז הבינתחומי, הרצליה.

המרכז הבינתחומי בהרצליה
בית-ספר אפי ארזי למדעי המחשב
התכנית לתואר שני (M.Sc.) - מסלול מחקרי

התקפות מניעת שירות ומנגנון שינוי קנה מידה אוטומטי

מאת
מור כהן גדול (סידס)

בהדרכת פרופ' ענת ברמלר-בר

עבודת תיזה המוגשת כחלק מהדרישות לשם קבלת תואר מוסמך M.Sc.
במסלול המחקרי בבית ספר אפי ארזי למדעי המחשב, המרכז הבינתחומי הרצליה

אפריל 2016