Tandem Attack: DDoS Attack on Microservices Auto-scaling Mechanisms

Anat Bremler-Barr*
Tel-Aviv University
anatbr@tauex.tau.ac.il

Michael Czeizler
Reichman University
michael.czeizler@gmail.com

I. Introduction

Today's software development landscape has witnessed a shift towards microservices architectures [6]. Using this approach, large software systems are composed of multiple separate microservices, each responsible for specific tasks. The breakdown to microservices is also reflected in the infrastructure, where individual microservices can be executed with different hardware configurations and scaling properties [8]. As systems grow larger, incoming traffic can trigger multiple calls between different microservices to handle each request [9].

Auto-scaling is a technique widely used to adapt systems to fluctuating traffic loads by automatically increasing (scale-up) and decreasing (scale-down) the number of resources used. We show that when microservices with separate auto-scaling mechanisms work in tandem to process ingress traffic, they can overload each other. This overload results in throttling (DoS) or the over-provisioning of resources (EDoS). This paper demonstrates how an attacker can exploit the tandem behavior of microservices with different auto-scaling mechanisms to create an attack we denote as the *Tandem Attack*. We demonstrate the attack on a typical and recommended *serverless* architecture [1], using AWS Lambda for code execution and DynamoDB as database.

The Tandem attack on our system architecture involves an attacker that overloads the system with requests, thereby exploiting the interaction between the Lambda's fast scale-up time and the database's relatively slow scale-up. The Lambda functions are triggered by the malicious traffic but the DB fails to keep up with the execution rate of the Lambda functions. This results in economic loss due to the increased cost of the Lambda functions, alongside a degradation in performance that impacts legitimate traffic.

By employing a Yo-Yo attack pattern [4] to generate periodic bursts of high traffic, we demonstrate that the database is forced to endure both denial-of-service and severe under-utilization of its resources, leading to a significant waste.

Furthermore, we analyze the implications of DynamoDB's provisioned mode policy, which restricts the number of scale-down operations per day. Our study shows that a malicious attack can leverage this limitation to inflict significant financial damage.

*The majority of this research was carried out while the author was at Reichman University.

We examine the attacks and their implications using established cybersecurity techniques, including DDoS and the Yo-Yo attack [4]. Nonetheless, self-inflicted Tandem attacks can also occur as a result of faulty configuration or design by system architects [7].

Unlike large-volume DDoS attacks that target network equipment, the Tandem attack is focused on the application layers and requires the attacker to invest far less resources. The differentiation between network and application attacks is also meaningful since the protection on the network layer is typically outsourced to scrubbing services and is offered by the cloud providers [2], while the application and infrastructure configuration are managed by system administrators.

II. Tandem Attack

We present the *Tandem attack*, which we conducted on a basic system consisting of two services:

AWS Lambda - a serverless computing service that executes code functions without requiring developers to manage or deploy the underlying infrastructure. Each Lambda execution is run separately, establishing a concurrency model that enables fast and adaptive scaling. The cost of a single Lambda is dominated by the duration of its code execution and the resources allocated to handle it.

DynamoDB - a popular AWS serverless key-value database service. The billing and capacity of DynamoDB are determined by the number of Write and Read Capacity Units (WCU and RCU, respectively); these are calculated based on the size of data written or read from the DB.

In our experiment, we configured DynamoDB to use *provisioned mode* with auto-scaling. In this mode, the user defines a range of the minimum and maximum capacity limits per second and specifies a target utilization percentage that DynamoDB aims to maintain. Essentially, the utilization percentage denotes the spare capacity the user is willing to pay for up-front in order to withstand an immediate spike in traffic; the total cost is constrained by the minimum and maximum capacity thresholds. Based on our findings, the scale-up process takes approximately 7 minutes, while scaling down takes approximately 20 minutes.

To achieve high throughput, the Lambda functions are executed simultaneously, resulting in parallel write requests to the DB. Our tests were crafted such that each HTTP request to the system was followed by 1 Lambda execution, performing a single write operation that consumes exactly

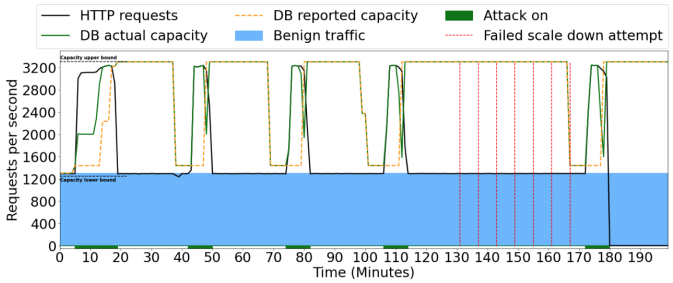

Figure 1: Experimental results

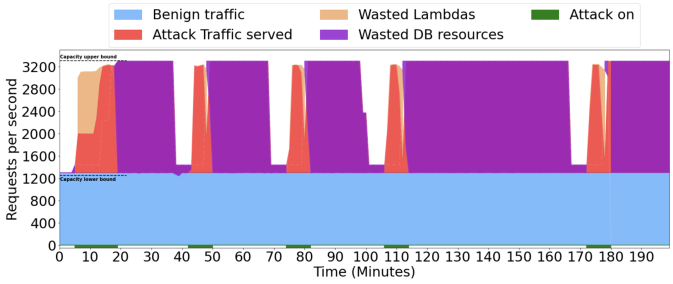

Figure 2: Economic damage analysis

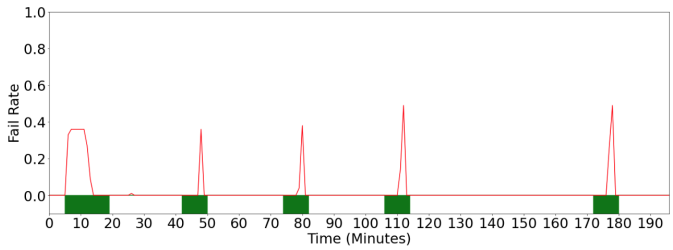

Figure 3: Performance damage

1 WCU. Although the ratio is not fixed in real production scenarios, we used the 1-to-1 mapping to simplify the analysis of the attacks.

In our experiment, the benign traffic rate to the application was configured to generate 1200 requests per second, while the database was set to a maximum capacity of 3300 WCU with a utilization level of 90%. The Lambda was limited to a rate of 3300 parallel executions. The attack overloaded the system by adding an additional 2000 requests per second.

The attacker in our experiment utilized the adaptive Yo-Yo pattern to execute a *Tandem attack*. In this pattern, the attacker created attack cycles by sending a burst of traffic until it detects that the system scaled-up to meet the extra traffic. Detection can be done by analyzing the traffic failure rate and Round Trip Time (RTT). Following the scale-up, the attacker ceases the traffic until scale-down is complete. The attacker can estimate the scale-down based on prior knowledge of the system or by calibrating the attack using trial and error. Once scale-down is complete, the attacker sends another burst of traffic.

Key Observations: The experimental results are presented in Figure 1. The data collected during the experiment included the actual WCUs consumed, which was calculated by the number of successful writes to the database, as well as the WCUs reported by AWS. It is worth noting that there was a delay of approximately 6 minutes in reporting the actual capacity. This can be explained by the fact that the DynamoDB scaling mechanism is based on sampling usage metrics every 6 minutes.

Figure 2 presents the economic damage incurred by the wasted resources, and Figure 3 shows the performance degradation calculated by analyzing the failure rate. Let us examine one cycle of the attack. During the attack, until the DB scales up, both economic loss and performance degradation take place. The Lambdas are scaled, but it takes time for the DB to scale up. Hence, there is the loss related to the cost of the Lambdas wasted because they could not be served by the DB. There is also a high failure rate. Although the failure rate decreases after the DB scales up, there is still economic damage incurred while the DB is serving the attack traffic. Although the attacker stops sending the traffic after the DB scales up, the economic damage continues to accumulate until the scale-down process is complete. This occurs because during this period, the system's owner is still paying for the high capacity of the DB, despite the absence of attack traffic. Then the attacker sends another burst of traffic, initiating a new cycle of the attack. Interestingly, in the fourth cycle of the attack, the scale-down took much longer, lasting between 105 and 177 minutes. The reason is that the provisioned mode contains a limitation on the number of scale-down actions that can be performed per day [3].

III. Conclusions

Our experiments, focused on serverless managed infrastructure, negate the concept of serverless services are fully handled by the cloud provider and do require strict planning by the system architect when it comes to scaling definitions. This paper presents part of a larger research work that defines and analyzes the *Tandem attack* in a broader context using multiple experiments and traffic patterns [5].

References

[1] AWS, "Aws lambda," 2023. [Online]. Available: https://aws.amazon.com/lambda/

[2] ——, "Aws shield," 2023. [Online]. Available: https://aws.amazon.com/shield/

[3] ——, "Decreasing provisioned throughput," 2023. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ServiceQuotas.html

[4] A. Bremler-Barr, E. Brosh, and M. Sides, "Ddos attack on cloud auto-scaling mechanisms," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[5] A. Bremler-Barr and M. Czeizler, "Tandem attack: Ddos attack on micro-service auto-scaling mechanisms - technical report," 2023. [Online]. Available: https://deepness-lab.org/publications/tandem-attack-ddos-attack-on-micro-service-auto-scaling-mechanisms/

[6] G. cloud, "What is microservices architecture?" 2023. [Online]. Available: https://cloud.google.com/learn/whatismicroservicesarchitecture?

[7] A. H. Dave Rensin, "How to avoid a self-inflicted ddos attack—cre life lessons," 2023. [Online]. Available: https://cloud.google.com/blog/products/gcp/how-to-avoid-a-self-inflicted-ddos-attack-cre-life-lessons

[8] Kubernetes, "Running multiple instances of your app," 2023. [Online]. Available: https://kubernetes.io/docs/tutorials/kubernetes-basics/scale/scale-intro/

[9] S. Luo and e. a. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 412–426.