The Cost of Retrying: Exploiting Retry-Mechanisms in Cloud Applications by DDoS Attacks

Jhonatan Tavori
School of Computer Science, Tel-Aviv University
Tel Aviv, Israel

Anat Bremler-Barr
School of Computer Science, Tel-Aviv University
Tel Aviv, Israel

Hanoch Levy
School of Computer Science, Tel-Aviv University
Tel Aviv, Israel

ABSTRACT

Retry mechanisms are commonly used in microservices architectures as a mechanism for recovering from transit errors, including network failures and service overloading. This research aims at studying the operation of cloud retry mechanisms under deliberated DDoS attacks, and their effect on the application performance and operational costs. In this poster we focus on the economic aspect, and demonstrate that enabling such mechanisms improperly might be counter-productive and expose the system to substantial and quadratic economical damage in the presence of attacks.

CCS CONCEPTS

• **Security and privacy → Network security**; • **Networks → Network performance analysis**.

KEYWORDS

Cloud Attacks; Microservices Architecture; Retry Mechanisms;

ACM Reference Format:
Jhonatan Tavori, Anat Bremler-Barr, and Hanoch Levy. 2023. The Cost of Retrying: Exploiting Retry-Mechanisms in Cloud Applications by DDoS Attacks. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies (CoNEXT Companion '23), December 5–8, 2023, Paris, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3624354.3630091

1 INTRODUCTION

Cloud applications in recent years are moving toward microservices-based architecture. As such, they are composed of many smaller loosely-coupled services. Each service is developed, deployed and managed independently, and is responsible for specific tasks [2].

In order to ensure their response time (latency) and availability, microservices-based applications rely on several mechanisms to mitigate the impact of partial outages and fluctuating traffic loads. These include capacity auto-scaling mechanisms[1] and resiliency patterns such as circuit-breakers[2] and retry mechanisms [3].

Retry mechanisms strive to establish a connection with a service in the event of an initial connection establishment failure, and continue their attempts until the call either succeeds or ceases in accordance with the retry designated policy. See Figure 1. Retries can be an effective way to handle and recover from transient failures, thereby preventing the need to discard entirely the request. Such failures can result from network errors (when a component loses connectivity for a short period of time), a service error (e.g., during automatic recovery from a crash), or when a service is overloaded and needs to limit incoming traffic. Retry patterns are commonly used. For example, most of AWS and Microsoft Azure services and client SDKs have implemented features for performing retries [2].

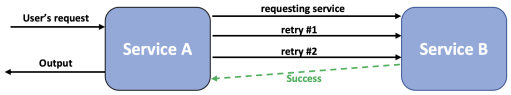

Figure 1: A retry mechanism. A user's request that is served in Service A makes repetitive retry requests for service from Service B.

Security is a major concern for businesses shifting their applications to the cloud. *YoYo* attack [1], which is a Denial-of-Service method formed by short and periodic high-rate traffic bursts, makes it hard to detect and mitigate the attack due to the reaction time of conventional DDoS mitigation solutions and by confusing them. It was shown that YoYo attacks can cause significant economical damage by exploiting auto-scaling mechanisms [1].

We study how retry mechanisms can be exploited by these attack patterns. During an attack, the retry mechanism has to manage traffic peaks of significantly greater magnitude than the usual traffic fluctuations which are typically effectively handled with a by a well-tailored retry policy. Due to the pricing structure of numerous cloud services, wherein the application administrator (cloud customer) is charged for each instance of the service running, and for the duration of each run, the extended processing time resulting from the retry mechanisms can lead to significantly high costs.

Our goal is to provide deeper insights into the operation of these mechanisms when exposed to deliberate DDoS attacks, as well as their impact on both the performance of the cloud application and the associated economical costs. We aim at gaining an understanding of how should retry policies be designed to withstand attacks, and identifying potential vulnerabilities in their operation.

Our results show that activating standard retry mechanisms may allow an attack to inflict a substantial economic damage, which can be *quadratic* in the attack size. As a result, attackers can achieve significant damage with reduced attack resources when retry mechanisms are improperly used, and extend their attack to multiple sites. Furthermore, we conduct an evaluation of our results and perform an experiment to demonstrate the damage inflicted by a YoYo attack on a basic AWS cloud application.

[1]Allowing dynamically increase and decrease of the number of resources used.
[2]Circuit breakers reject incoming requests, prioritizing latency over availability, while enabling faster and cheaper reactions to load spikes compared to auto-scaling.

CoNEXT Companion '23, December 5–8, 2023, Paris, France

ACM ISBN 979-8-4007-0407-9/23/12.
https://doi.org/10.1145/3624354.3630091

2 QUADRATIC DAMAGE: ATTACK ANALYSIS

We analyze the economical damage induced by YoYo attacks and establish that under certain retry policies the resulting damage can exhibit quadratic growth in proportion to the attack volume.

We consider a basic Amazon AWS cloud application consisting of two widely used services: AWS Lambda and Amazon DynamoDB [2]. Lambda is a compute service that executes code without the cloud's customer involvement in managing its infrastructure, and is suitable mainly for short duration processes due to high pricing relatively to user-controlled compute services such as EC2; DynamoDB is a key-value database managed service. While Lambda has no maximum concurrency limitation[3], DynamoDB is often configured to use provisioned mode in which an upper limit for read/write operations rates is defined that the database provisioning should not exceed. The application operates as illustrated in Fig. 1: users requests trigger the execution of a Lambda function (Service A), which, in turn, calls upon DynamoDB (Service B) to finalize its execution and provide a response to the user. Under high loads the number of concurrent Lambda instances will exceed the database provisioned capacity limit, causing retries of read/write requests.

The maximal number of retries per request, and the chosen time interval between retry attempts (known as the backoff time[4]), are frequently adjustable parameters within an implementation of a retry pattern. We consider a naïve retry pattern, in which the maximal number of retries is unbounded, and the backoff time is constant and negligible. I.e., once Service B is able to accommodate requests it will start serving pending requests from Service A[5].

We assess the overall duration spent, and paid for, within Service A for all incoming requests, benign and malicious, including time spent waiting for a response from Service B if retries were needed. This cumulative duration will be denoted as the *Attack Damage*.

Theorem 2.1. *Enabling retry mechanism allows DDoS burst attacks like the YoYo pattern to cause economical damage that can scale quadratically with the size of the attack.*

We omit its full proof due to lack of space, and demonstrate the results using Fig. 2. The figure depicts the number of parallel Lambda executions within the system during a YoYo attack. Once the database completes handling all the retries of the malicious traffic of the preceding attack burst, a new burst is launched. The

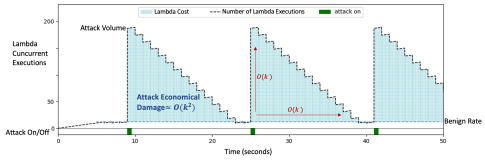

Figure 2: Economical damage is quadratic in the attack size.

economic damage caused by the attack is depicted in the figure by the blue area. As can be seen the blue area is composed of "triangles" following each attack burst. Each triangle is of size $O(k^2)$, due to it's height and width being proportional to the attack size.

[3]An initial limit for this number is set at the account level and can be increased by contacting AWS.

[4]The backoff time usually fits into three categories: constant, incremental, or random, each suited to different scenarios and objectives (e.g., response time or availability).

[5]This pattern provides a *lower bound* on the attack's damage, compared to incremental or random backoff policies, in which even when Service B becomes available, it's possible that requests will experience waiting periods until the next retry occurs.

3 EVALUATION

We perform an experiment to demonstrate the economical damage on the described AWS cloud application. We employed the default AWS Standard Retry mode, which utilizes exponential backoff as its retry mechanism. The attacker enforced a YoYo attack pattern, i.e., overloaded it with repeated bursts of size k, with each attack burst lasting 10 seconds. The benign traffic rate was consistently maintained at 320 requests per second, and each attack burst generated a load of $k \cdot 320$ requests per second. The experiment was repeated for several values of k. We collected the attack damage using Amazon CloudWatch. The results are presented in Table 1.

Attack Size	Attack Damage
k = 1	1455.24
k = 2	3025.52
k = 4	6146.79
k = 8	25237.11
k = 16	72550.56

Table 1: Experiments results – Total time of Lambda execution and cost.

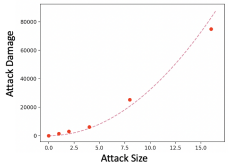

Figure 3: Attack damage as a function of the attack size.

When dealing with low attack volumes, raising the attack value resulted in a linear increase in the incurred damage since the database having provisioned capacity that can handle sudden traffic spikes, and thus retries are nearly nonexistent; In cases where the attack volume was comparatively high, the impact of doubling the attack size surpassed a simple doubling effect, and followed a quadratic nature. In Figure 3 we plotted the attack damage as a function of the attack size with a quadratic interpolation between the points.

4 CONCLUSIONS AND FURTHER WORK

Our results highlight that when enabling and configuring retry mechanisms within cloud applications, it is crucial to consider not only the tradeoff between service availability and response time but also the potential economic costs in the event of attacks, as attackers may exploit this vulnerability to devise attacks which inflict substantial damage while using minimal attack resources.

We plan to further study the operation of these mechanisms under diverse attack patterns, including situations where self-inflicted DDoS could potentially arise. We aim to develop well-considered and efficient strategies for the implementation of these mechanisms.

Acknowledgment. This research was supported in part by the Israel Science Foundation (grant No. 2482/21), by the Blavatnik Family Foundation and by RedHat Research.

REFERENCES

[1] Anat Bremler-Barr, Eli Brosh, and Mor Sides. 2017. DDoS attack on cloud autoscaling mechanisms. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[2] Sajee Mathew and J Varia. 2014. Overview of amazon web services. *Amazon Whitepapers* 105 (2014), 1–22.

[3] Mohammad Reza Saleh Sedghpour, Cristian Klein, and Johan Tordsson. 2022. An Empirical Study of Service Mesh Traffic Management Policies for Microservices. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*. 17–27.