A serverless architecture for frequency-based HTTP request filtering against distributed denial-of-service (DDoS) attacks

Soohyun Lee

Department of Biomedical Informatics, Harvard Medical School, Boston, MA. USA.

duplexa@gmail.com

Abstract

Hypertext Transfer Protocol (HTTP)-based distributed denial-of-service (DDoS) attacks are extremely common and damaging to both on-premise and cloud-based Web servers. In this project, I explored a possibility of using a serverless architecture to respond to HTTP-based DDoS attacks on cloud-based Web servers.

The main idea of the architecture is to monitor the frequency of HTTP requests in the form of invocation frequency of a serverless function that bridges the Web server and the Internet. If the invocation frequency goes above a threshold, it sets off an alarm, which triggers another serverless function to add high-frequency requesters to a denial list. The alarm also sends an email notification to an admin.

A proof-of-concept architecture is designed and implemented using Amazon Web Services (AWS) resources including Lambda serverless functions, API Gateway, CloudWatch, DynamoDB and Simple Notification Service (SNS). The architecture is deployed using AWS CloudFormation following the infrastructure-as-code approach. An example CloudFormation template and Lambda function code are available under the MIT license on <https://github.com/SooLee/serverless-ddd>.

Keywords: cloud, serverless, AWS Lambda, Amazon Web Services, AWS, cybersecurity, denial-of-service, DoS, DDoS, request filtering.

1. Background

Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks are cyberattacks on servers or networks aimed at overwhelming the target and disrupting the availability of the service. A DoS attack is usually initiated from a single source, whereas a DDoS attack involves multiple sources. A DDoS attack typically uses a large number of compromised hosts, called a botnet, that are hijacked and controlled by an attacker to send requests to a target (Hoque et al., 2014). The specific methods may vary. For example, some attackers recently started using memcached servers instead of botnets (Newman, 2018). There are known DDoS tools such as *Crazy Pinger*, *UDP Flood*, *HOIC*, *LOIC*, *Panther*, *Slowloris*, *BlackEnergy*,

etc. (Hoque et al., 2014). The *itsoknoproblembro* toolkit used in the 2012 DDoS attack on 6 American banks is another example (Kumar, 2012)). Some organizations lend botnets to attackers.

According to Mahjabin et al. (2017), most DoS events tend to be distributed since the first reported DDoS attack in 1999, and the volumes of these attacks have been rapidly increasing since 2012. The frequency of attacks increased more than 2.5 times between 2014 and 2017 (Galov, 2021). Nowadays, DoS and DDoS constitute the most common type of cyberattacks (Galov, 2021) and over 23,000 DDoS attacks happen each day (Osborne, 2020).

DDoS attacks can last for hours to days (Galov, 2021). For example, a DDoS attack in 2009 on Bitbucket resulted in 19 hours of down time (Metz, 2009). The two longest DDoS attacks in 2020 lasted for over ten days (Galov, 2021). According to a survey cited by Kobialka (2017), 51 percent of respondents said they needed at least three hours to detect a DDoS attack and 40% said they needed at least three hours to respond to an attack. Tariq et al. (2006) deemed DDoS attacks ‘the greatest security fear for IT managers.’ A DDoS attack may cause a significant financial loss (average between \$20,000 and \$40,000 per hour) (Galov, 2021). A DDoS attack could also act as a cover-up for another attack that may cause a more serious harm. Due to the distributed nature, the true origin of a DDoS attack is very difficult to identify (Hoque et al., 2014).

The most common type of DDoS attacks is User Datagram Protocol (UDP) flooding, and the second most common is HTTP-based application layer attacks (Galov, 2021). A DDoS attack can either be volumetric or slow (e.g. Slowloris), with volumetric accounting for 65% of the attacks (Chauhan, 2018). The focus of this project is on volumetric HTTP-based DDoS attacks.

Implementing anti-DDoS measures is challenging partly because it requires handling of a large volume of data but without using such capacity except when an attack happens (Dotson, 2019). A serverless architecture may be a good fit for this kind of problem, since it allows a fast scale-up by several orders of magnitude with little operational cost during normal time.

Serverless computing is a relatively new cloud computing paradigm that became available in 2014 when AWS introduced Lambda (Jangda et al., 2019). Other major cloud providers (e.g. Google Cloud, Azure, IBM) have since implemented serverless capabilities as well (Baldini et al. 2017). The AWS Lambda service allows users to define and invoke small pieces of custom code called serverless functions. These functions are executed in a dedicated server environment without the overhead of launching a virtual machine. Resource allocation, scaling and load balancing of the underlying servers are managed by the cloud provider and not visible to the cloud customer. The customer pays only for the duration of the execution of their serverless functions. These functions are given limited run time and compute resources, and therefore not fit for heavy-duty workloads individually, but collectively they can perform intensive distributed tasks due to their horizontal scalability. A serverless service based on serverless functions is also called function-as-a-service (FaaS) (Baldini et al. 2017). There are other types of serverless services as well, such as AWS DynamoDB which is a serverless database.

A cloud provider typically provides other integrations around serverless functions including logging, monitoring, and invocation by other services or by other serverless functions (Baldini et al., 2017). These integrations enable serverless functions to work as building blocks in a microservice architecture. Microservice architectures are widely used for cloud application development. The term was first used in 2011 and it refers to a loose coupling of independently deployable components that communicate through messages (Dragoni et al. 2017). The concept is often contrasted to a monolithic-style application that weaves different functionalities into a single unit. The architecture presented here is a microservice

architecture based on serverless functions and databases. The components of the architecture are integrated with one another using the cloud provider’s secure network and role-based permissions. By nature, the architecture is modular and the resources can be treated independently, though for this project they are deployed together as a single stack.

2. Results

2.1. Design and implementation

Figure 1. An architecture diagram describing the serverless DDoS prevention system on AWS Cloud.

The summary of the architecture is described in Figure 1. It is based on resources and services on AWS, but a similar design may be applicable to other public cloud services offering equivalent services. The basic idea is to use two Lambda functions between a Web server instance and the client, and to set up an alarm to be triggered by high-frequency invocations of one of those functions. The alarm invokes a third Lambda function to update the filter.

All HTTP requests from the Internet goes through an Application Programming Interface (API) endpoint created by the AWS API Gateway service. The API is of the Representational State Transfer (REST) (Fielding, 2000) style and supports the Hypertext Transfer Protocol Secure (HTTPS) protocol (Rescorla, 2000). A Lambda function integrated to the API called Filter Lambda is invoked for every request passing the endpoint, checks the request against a Denial List Dynamo DB table and passes it to a second Lambda function called Connection Lambda. Connection Lambda then sends the request to the Web server instance residing in a private subnet inside a Virtual Private Cloud (VPC), after recording the request information in a DynamoDB table called Raw table. The response from the server travels in the reverse direction, through Connection Lambda, Filter Lambda, API endpoint, then to the user.

An AWS Cloudwatch alarm goes off if Connection Lambda's invocation frequency goes above a threshold. The alarm triggers a third Lambda called Alarm Lambda to parse the Raw table to identify high-frequency requesters and to add them to the Denial List table. The alarm also sends an email notification to an admin using AWS Simple Notification Service (SNS).

As a result of an alarm response, requests associated with high-frequency requesters can no longer reach the Web server instance, while normal requests are not affected. To differentiate high-frequency requesters from normal ones, a combination of the source Internet Protocol (IP) address and user agent is used as an identifier. The selective blocking of high-frequency requests is described in Figure 2.

Figure 2. Architecture diagram showing the attacker being selectively blocked after the high-frequency invocation alarm. The Web server is no longer reached by the attacker, whereas normal users continue accessing the Web server's content.

If a DDoS attack is partially blocked but continues to bombard the endpoint with different source IP addresses and/or user agents not yet been listed in Denial List, those requests could be captured in the next alarm cycle. It usually takes a few minutes for Cloudwatch Alarm to go off after high-frequency

invocations start. The alarm typically stays in the ALARM state for a few minutes, which allows time for visual inspection of the alarm, at the expense of delaying the next alarm. This behavior can be changed, for example, by modifying the Alarm Lambda code to reset the alarm immediately, which could help accelerate the detection of remaining high-frequency requests. Effective blocking of an attack may take several rounds, depending on how many IP addresses and user agents are used, how frequently they change and how they are distributed over time.

2.2. Deployment using CloudFormation

The life cycle of the system can be conveniently and reproducibly managed through the infrastructure-as-code approach which allows defining the architecture as one or more text file(s). AWS CloudFormation (<https://aws.amazon.com/cloudformation/>) takes a template file that describes the desired state of the set of interconnected resources, and creates all the resources as a single unit called a stack. The Denial List DynamoDB table is deployed and managed as a separate CloudFormation stack, to avoid deleting existing records by redeploying the table along with the other components.

The deployment also requires three zipped Python code files uploaded separately to an AWS Simple Storage Service (S3) bucket in advance, to be used by CloudFormation to create the three Lambda functions. To enable the SNS email notification, an email confirmation needs to be done manually after deployment.

2.3. Limitations of the architecture

Web server performance

Lambda invocation is fast and adds minimal delay in response. The latency is within milliseconds according to AWS (<https://aws.amazon.com/lambda/faqs/>). Adding two Lambdas between the Server and the API endpoint does not seem to worsen user experience. However, it may depend on the latency requirement of a particular Website, and may not fit for a highly interactive Web application.

Lambda limits

Lambda functions come with run time limit (up to 15 minutes) and request / response size limit (up to 6 MB). This means certain types of requests or responses (e.g. downloading a large file) may not work well with Lambda. There are ways to overcome these limits (Yan Cui, 2020) such as giving a pre-signed S3 URL or redirecting the download to a different API Gateway endpoint that connects directly to S3.

Working with the Web server instance

Since the Web server instance is in a private subnet, it would require an additional set-up (e.g. a Virtual Private Network (VPN) such as OpenVPN (<https://openvpn.net/>)) to connect to the instance through Secure Shell (SSH), check or install something after the instance is deployed.

Alarm latency

The alarm's response speed is in the order of minutes. This means the Web server could become unavailable for a few minutes when a real attack happens.

API Gateway throttling

API Gateway cannot handle an infinite number of requests. For example, if more than 10,000 requests come to the endpoint in the same one-second period, some of them may be dropped (for more details, see <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html>). According to Mimoso (2015), a heavy DDoS attack could send 275,000 HTTP requests per second at its peak. The Web server itself will not be bombarded with this many requests with the serverless setup, since most of them are dropped at the API Gateway level. However, since throttling does not differentiate between requesters, normal requests would also be dropped. Rerouting to a temporary endpoint may help accommodate urgent legitimate requests. An AWS Web Application Firewall (WAF) could work as an additional filter and could reduce the number of malicious requests passing through the endpoint.

2.4. Potential extensions of the architecture

Approval list

In addition to a denial list, an approval list that overrides the denial list can be a useful addition, since it prevents known users from triggering an alarm. If a normal user happens to be caught in a denial list, an admin could manually add them to the approval list.

Information other than source IP address and user agents

A similar architecture can be used to support request information other than source IP addresses and user agents. For example, a specific path or parameters could be added to the combination.

Autoscaling and load balancing of the Web server

Adding an autoscaling group of Elastic Compute Cloud (EC2) instances (i.e. virtual machines) and a load balancer could help preventing the server from being overwhelmed by increased traffic.

Multi-tier architecture

The Web server instance could connect to a database server in the same VPC. The serverless system would enhance protection of the database server as well as the Web server in this case.

Multi-AZ architecture

The architecture could be extended to cover multiple availability zones (AZ) to ensure higher availability.

Domain name

The API Gateway's endpoint URL is randomly generated and is subject to change. A custom domain name can be mapped to the API, which may be a more realistic scenario.

2.5. Architecture Test

A proof-of-concept testing of the architecture has been done by using a low threshold for the alarm, without actually performing a DDoS attack on the system. The alarm threshold was set up to 10 invocations per minute. The threshold to determine high-frequency requesters was set to 12 requests for each IP address + user agent combination in the past ten minutes. A fake 'attacker' was simulated to make requests at about 45-per-minute rate. The attacker's requests were effectively blocked within a few minutes and the admin received an email notification a few minutes later, while other requests (a different

IP address or browser) continued to access the site. A piece of Python code used to simulate a fake attacker is shown in Figure 3. Some screenshots from this test are also shown in Figures 4-6.

```
import requests
import time

# replace below with your API endpoint URL!
URL = 'https://ijtiml3kd.execute-api.us-east-1.amazonaws.com/api'

# make a GET request at a 1-second interval (~45 requests per minute) for ~27 minutes
for i in range(1, 1200):
    res = requests.get(URL)
    print("status_code=" + str(res.status_code))
    print("content=" + res.content.decode('utf-8'))
    time.sleep(1)
```

Figure 3. Python code that simulates a fake attacker by sending a request to the API Gateway endpoint at a 1-second interval for more than 20 minutes.

	uuid	expdate (TTL)	requestTime	sourceIP	userAgent
☐	7b66bcc0-64e7-4544-88c5-b6f3bd1fe10c	1624902932	27/Jun/2021:17:55:32 +0000	173.76.14.70	python-requests/2.25.1
☐	28e878fc-3626-442f-a420-b0667e706c34	1624902937	27/Jun/2021:17:55:36 +0000	173.76.14.70	python-requests/2.25.1
☐	a513612a-74b5-47d7-8468-f5ae3d2ca5ff	1624902942	27/Jun/2021:17:55:41 +0000	173.76.14.70	python-requests/2.25.1
☐	a0acfa5-12e5-4920-a1d5-f63f2f25be9a	1624902947	27/Jun/2021:17:55:46 +0000	173.76.14.70	python-requests/2.25.1
☐	3487dc60-eb7d-4487-8c2c-e44c6663014a	1624902952	27/Jun/2021:17:55:51 +0000	173.76.14.70	python-requests/2.25.1
☐	e0cbbe63-4434-4196-bf6a-de4b9361dab0	1624902957	27/Jun/2021:17:55:56 +0000	173.76.14.70	python-requests/2.25.1
☐	3002753a-0cc9-4b5f-859a-5763fc254ab1	1624902961	27/Jun/2021:17:56:01 +0000	173.76.14.70	python-requests/2.25.1
☐	e4b7d037-4c46-420a-834f-4390798aae56	1624902966	27/Jun/2021:17:56:06 +0000	173.76.14.70	python-requests/2.25.1
☐	ae7771d5-3d68-4607-984b-3365ce6d35c6	1624902971	27/Jun/2021:17:56:11 +0000	173.76.14.70	python-requests/2.25.1
☐	88455c13-dadb-420b-8745-21485fa6ddc9	1624902976	27/Jun/2021:17:56:15 +0000	173.76.14.70	python-requests/2.25.1
☐	a717d4a9-ae59-4a5f-bc16-bbc00197796a	1624902981	27/Jun/2021:17:56:20 +0000	173.76.14.70	python-requests/2.25.1
☐	be5dd148-8704-48b9-86d7-4c2d987ce576	1624902985	27/Jun/2021:17:56:25 +0000	173.76.14.70	python-requests/2.25.1

Figure 4. A screenshot of the dynamoDB Raw Table getting filled with the source IP address and user agent information. Each record is set to expire in 24 hours through the Time to Live (TTL) specification of DynamoDB.

Figure 5. Cloudwatch alarm detecting above-threshold Connection Lambda invocations.

	key	timestamp
☐	173.76.14.70:::python-requests/2.25.1	27/Jun/2021:20:06:36 +0000

Figure 6. The fake attacker's source IP + user agent combination is listed in the Denial List dynamoDB table after the alarm.

Figure 7. Screenshot of request counts at the API Gateway level, showing the continued attempt of the 'fake attacker.'

Figure 8. Screenshot of EC2 Web server instance showing a temporary increased inbound network activity at the beginning of the 'attack.' The activity goes back in a few minutes due to the selective block by the serverless filter, while the attack is still going on at the endpoint.

2.6. Real-world incident detection

For the real-world setup, the parameters must be different from the ones used to test the architecture. Two parameters should be set: Connection Lambda invocation frequency threshold (for total requests per minute) and Alarm Lambda's Denial List threshold (for the request frequency of an individual source IP + user agent combination to be listed in Denial List). The threshold values should be higher than the maximum expected based on normal traffic and lower than a DDoS traffic.

The effective threshold could therefore depend on the normal traffic of the Website. For example, for a Website targeting a small group of intermittent users, on which an individual source IP + user agent is unlikely to make more than 100 requests per minute, the threshold can be set to 100 per minute to be added to the Denial list. Let's say the total requests per minute is expected to be less than 1,000 during normal time and less than 10,000 during a peak time. A single server may not be able to handle more than 1,000 requests per second, which translates to 60,000 per minute. Considering this, 20,000 – 30,000 total requests per minute seems a reasonable invocation threshold. Note that these numbers are examples and

would depend on how much traffic a specific server can handle and how many server instances are typically running to accommodate the normal traffic.

The email notification can bring in a quick human response in case the automated response is not sufficient. For example, more complex filtering rules could be manually added to a WAF to block the attack at the API Gateway level. It may also be worth notifying the legitimate users of the incident and creating a temporary private endpoint for urgent use cases such as a secondary API Gateway endpoint pointing to the same Lambda proxy.

2.7. Assessment and compliance

One of the ways to assess the impact of DDoS prevention is through an availability metric. For example, a metric of hours in a given year that the system was not available can be used. For example, if the down time is 4 hours 22 minutes 58 seconds, the availability is 99.95% (calculated by <https://uptime.is/99.95>). Failure to respond to DDoS attacks is likely to bring down the availability below this number. Arazi (2018) points out other metrics such as time-to-detect and time-to alert when a DDoS attack happens. With the Lambda-based architecture these metrics are likely to be several minutes.

According to AWS's compliance catalogue pages (<https://aws.amazon.com/compliance/services-in-scope/> and <https://aws.amazon.com/compliance/hipaa-eligible-services-reference/>), all the components of the architecture including Lambda, API Gateway, DynamoDB, SNS, Cloudwatch and CloudFormation are HIPAA-, FedRAMP- and PCI-compliant, suggesting that the architecture could be applied to a scenario subject to those compliances.

3. Conclusions

This article describes a serverless architecture based on AWS Lambda to trigger an alarm against a DDoS attack to update a denial list and send an email notification. The system is deployed using the infrastructure-as-code approach.

Existing DDoS solutions should definitely be considered for a fast and reliable strategy. For example, an AWS Web Application Firewall (WAF) could be attached to the API Gateway endpoint to block known attack sources and apply other filtering rules. AWS also provides a powerful DDoS prevention service called AWS Shield Advanced that protects against HTTP as well as UDP reflection flood, SYN flood and DNS query flood attacks (Priyam, 2018). AWS Shield Advanced is expensive (e.g. \$3,000 monthly fee plus data transfer), but its response time is less than seconds. For more information on the available AWS DDoS response services and the recommendations from AWS itself, see the white paper at https://d0.awsstatic.com/whitepapers/Security/DDoS_White_Paper.pdf). There are also third-party solutions such as Akamai Prolexic, which responded to a massive, tera-byte-scale DDoS attack on GitHub within 15-20 minutes (Newman, 2018).

Note out that a full DDoS load test was not performed on the Lambda-based architecture described here. The purpose of this project was to come up with a design that works on a small scale as a proof of concept, and not intended to replace existing or recommended DDoS response services. AWS Lambda is relatively inexpensive (\$0.216 for 1 million function calls if each takes a second and transfers 1MB data), and it may be worth exploring for those who prefer a more affordable solution or a combination of solutions. I

hope this article offers a useful insight to those interested in potentially using a serverless architecture for DDoS management.

4. Acknowledgement

This article is a revised version of the final project report submitted on May 10, 2021 after a live demo on May 5, 2021, for a CSCIE49 Cloud Security course at Harvard Extension School. I thank Ramesh Nagappan, the instructor of the course, and Tom Gould, a teaching assistant, for their supportive and helpful advice and feedback.

5. References

Hoque et al. (2014) "Network attacks: taxonomy, tools and systems." *Journal of Network and computer applications*. 40:307-324.

Specht and Lee (2004) "Distributed Denial of Service: taxonomies of attacks, tools and countermeasures" *ISCA PDCS*. 543–550

Lily Hay Newman (2018) "GitHub Survived the Biggest DDoS Attack Ever Recorded." *Wired*. Retrieved from <https://www.wired.com/story/github-ddos-memcached/>

Mohit Kumar (2012) "Cyber Attacks on Six Major American Banks." *The Hacker News*. Retrieved from <https://thehackernews.com/2012/10/cyber-attacks-on-six-major-american.html>.

Tasnuva Mahjabin et al. (2017) "A survey of distributed denial-of-service attack, prevention, and mitigation techniques" *International Journal of distributed sensor networks*. vol 13(12) doi: 10.1177/1550147717741463.

Charlie Osborne (2020, Feb 18) "16 DDoS attacks take place every 60 seconds, rates reach 622 Gbps" retrieved from <https://www.zdnet.com/article/16-ddos-attacks-take-place-every-60-seconds-rates-reach-622-gbps/>

Nick Galov (2021, Jan 16) "39 Jaw-Dropping DDoS Statistics to Keep in Mind for 2021" retrieved from <https://hostingtribunal.com/blog/ddos-statistics/#gref>

Usman Tariq et al. (2006) "A Comprehensive Categorization of DDoS Attack and DDoS Defense Techniques" *Advanced data mining and applications second international conference proceedings*. pp1025-1036.

Cade Metz (2009) "DDoS attack rains down on Amazon cloud" *The Register*. Retrieved from https://www.theregister.com/2009/10/05/amazon_bitbucket_outage/

Ajay Singh Chauhan (2018) "DoS and DDoS attacks." *Practical network scanning*. Packt publishing.

Chris Dotson (2019) "6. Network Security" in *Practical cloud security: a guide for secure design and deployment*. O'Reilly Media Inc.

Dan Kobialka (2017, May 2) "DDoS attack research: What's your response time?" retrieved from <https://www.msspalert.com/cybersecurity-research/ddos-attack-research-whats-response-time/>

Michael Mimoso (2015, Sep 28) "JavaScript DDoS Attack Peaks at 275,000 Requests-Per-Second" retrieved from <https://threatpost.com/javascript-ddos-attack-peaks-at-275000-requests-per-second/114828/>

Abhinav Jangda et al. (2019) "Formal foundations of serverless computing." *Proceedings of the ACM on Programming Languages*. vol. 3. Article no. 149. pp1-26. doi: 10.1145/3360575.

Ioana Baldini et al. (2017) "Serverless computing: current trends and open problems." *Research advances in cloud computing*. Springer. pp.1-20. doi:10.1007/978-981-10-5026-8_1.

Nicola Dragoni et al. (2017) "Microservices: Yesterday, Today, and Tomorrow." *Present and Ulterior Software Engineering*. pp195-216.

Roy Thomas Fielding (2000) "Architectural styles and the design of network-based software architectures." Ph.D. dissertation, University of California, Irvine.

Eric Rescorla (2000) "HTTP over TLS." Request for Comments. doi:10.17487/RFC2818

Yan Cui (2020, April 9) "Hit the 6MB Lambda payload limit? Here's what you can do." retrieved from <https://theburningmonk.com/2020/04/hit-the-6mb-lambda-payload-limit-heres-what-you-can-do/>

Eyal Arazi (2018, May 9) "6 must-have metrics in your SLA." retrieved from <https://blog.radware.com/security/2018/05/6-must-have-sla-metrics/>

Prashant Priyam (2018) "Identity and access management in the cloud." *Cloud Security Automation*, Packt Publishing.