Warmonger: Inflicting Denial-of-Service via Serverless Functions in the Cloud

Junjie Xiong
University of South Florida
junjiexiong@usf.edu

Mingkui Wei
George Mason University
mwei2@gmu.edu

Zhuo Lu
University of South Florida
zhuolu@usf.edu

Yao Liu
University of South Florida
yliu21@usf.edu

ABSTRACT

We debut the *Warmonger attack*, a novel attack vector that can cause denial-of-service between a *serverless computing platform* and an *external content server*. The Warmonger attack exploits the fact that a serverless computing platform shares the same set of egress IPs among all serverless functions, which belong to different users, to access an external content server. As a result, a malicious user on this platform can purposefully misbehave and cause these egress IPs to be blocked by the content server, resulting in a platform-wide denial of service. To validate the Warmonger attack, we ran months-long experiments, collected and analyzed the egress IP usage pattern of four major serverless service providers (SSPs). We also conducted an in-depth evaluation of an attacker's possible moves to inflict an external server and cause IP-blockage. We demonstrate that some SSPs use surprisingly small numbers of egress IPs (as little as only four) and share them among their users, and that the serverless platform provides sufficient leverage for a malicious user to conduct well-known misbehaviors and cause IP-blockage. Our study unveiled a potential security threat on the emerging *serverless computing* platform, and shed light on potential mitigation approaches.

CCS CONCEPTS

• **Security and privacy** → *Denial-of-service attacks*; • **Networks** → *Cloud computing*;

KEYWORDS

Cloud Computing, Edge Computing, Serverless Functions, Denial-of-Service

ACM Reference Format:
Junjie Xiong, Mingkui Wei, Zhuo Lu, and Yao Liu. 2021. Warmonger: Inflicting Denial-of-Service via Serverless Functions in the Cloud. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3485372

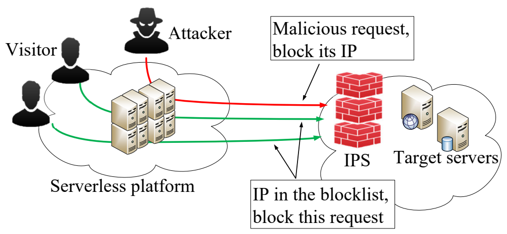

Visitor
Attacker
Malicious request, block its IP
Serverless platform
IPS
Target servers
IP in the blocklist, block this request

Figure 1: Warmonger attack threat model.

1 INTRODUCTION

In recent years, the Functions-as-a-Service [25, 29, 44], or FaaS, has emerged as a new cloud computing paradigm, replacing traditional cloud services like IaaS, PaaS, and SaaS. As a new service model, FaaS provides a platform on which developers, particularly Web App developers, can directly compose code on the platform using popular languages like JavaScript and Python; while tasks like code storage, maintenance, and execution are all handled by the FaaS platform that is backed by the cloud service provider's (CSP's) powerful infrastructure. Because the code is migrated from the origin server to the FaaS platform provided by the CSP, the application owner no longer needs to maintain a local server, and thus the FaaS is also commonly known as the *Serverless Functions*.

In a nutshell, a developer can compose a piece of code, usually one that is simple and single-purposed that is suitable for web applications, and deploy it to one of the CSP's dedicated data centers or its distributed edge servers. This piece of code, or the *serverless function*, remains dormant until it is triggered by an *event*, such as an HTTP(S) request sent to a dedicated URL. After being activated, the function will run once and subsequently return to be inactive until the next trigger arrives. A serverless function runs on the CSP's powerful infrastructure and can be deployed on edge servers close to the end-user, and thus provides the benefit of a more agile response. Furthermore, a serverless function only runs once after it is invoked, rather than occupying an always-run server instance, which can save significant computational resources and costs.

Like many other cloud-based services, serverless functions accommodate multiple Internet users to allow the share and reuse of CSP's limited hardware and software resources. On such a shared platform, a typical security threat is that improper or insufficient user isolation may cause a user's private data to be accessed by another unauthorized user on the same platform. To date, many research studies have been published in this regard. For example, [29] found that allowing code to move smoothly towards shared data storage may exacerbate the potential for rogue code to collect customer-wide signals. [61] studied the security implications of the ways the serverless platforms separate the roles of various accounts using virtual machines or containers, and evaluate their performance in terms of scalability, cold start latency, and resource efficiency.

In this paper, however, we unveil a novel security threat on such a shared platform that differs from previous studies. Specifically, in this attack vector, the attacker's target is not a particular user, but all the users on the shared platform; and the attacker's objective is to interrupt the normal operation of all the users on the platform including herself, by intentionally abusing the shared resources to invoke platform-wide punishment. An example is demonstrated in Figure 1. In this example, the attacker instructs a serverless function to send obviously-malicious requests from the serverless platform to an external server. Such requests may inflict the external server's protective behaviors such as rate limit and IP blockage, which, consequently, result in collateral damage to all the users on the serverless platform who share the same set of IPs.

To understand the practicality and impact of this attack, namely the Warmonger attack, we conducted experiments to collect and analyze the egress IP usage patterns of four major SSPs; *Amazon Web Service (AWS) Lambda*, *Google App Engine*, *Microsoft Azure Functions*, and *Cloudflare Workers*. Our study revealed worrisome results in some of the SSPs. For instance, we found Cloudflare Workers mainly relies on as little as 4 egress IPs on some of the edge servers that we have evaluated, these IPs are shared among other users on the serverless platform and are used for roughly one month before they are replaced with another 4 IPs. Our study also demonstrates interesting and non-intuitive IP usage patterns of these SSPs, which can further facilitate studies in this direction. We further evaluated the leverages a serverless platform provides to a user, i.e., what malicious behaviors a user can conduct on this platform. Our study shows that it is possible for a malicious user to conduct well-known offensive behaviors such as HTTP flooding, continuous SSH connection and port scanning. We summarize our contributions as follows:

- We identified the potential risk on the serverless functions platform, where a misbehaving user (i.e., the attacker) can cause collateral damage and affect other users. The proposed Warmonger attack, to the best of our knowledge, is the first such attack to reveal and address this new attack vector.
- We conducted experiments to collect and statistically analyze the egress IP usage patterns of different SSPs, such as the amount of IPs used and their usage frequencies, and how they are shared among different users.
- We also assessed the "leverage" a serverless function provides to a malicious but nonetheless unprivileged user, that is, what malicious behavior a regular user can conduct on a serverless platform to really trigger protective reactions from the external server.
- Our study reveals drastically different IP usage patterns between different SSPs and consequently identifies various potential risks. Our study elucidates potential remedies against such attacks.

The rest of this paper is structured as follows. In Section 2, we first introduce the background of the SSPs and how they are used in today's Internet. We describe the threat model in Section 3, where we provide details of the Warmonger attack, including the roles that are involved and our assumptions. In Section 4, we describe the details of the experiment setup, the results, and our analysis and insights. In Section 5, we experiment with the capabilities of a malicious user on the serverless platform and discuss potential malicious behaviors that an attacker can use to intrigue the Warmonger attack between the serverless platform and an external server. We discuss potential remediation and preventative measures that can take to address this attack in Section 6, and discuss related work in Section 7. We finally conclude our work in Section 8.

2 BACKGROUND

2.1 Serverless Computing

As we have described in Section 1, serverless computing is a cloud computing paradigm in which the cloud service provider allocates on-demand computing resources and operates the machines on their customers' behalf. It differs from conventional cloud computing services in the following aspects.

Functionality: Serverless computing is a dedicated code-execution platform. A serverless function service provider provides an IDE (usually web-based) where a developer can directly write code. Such functions are most commonly used for web applications, which are usually light-weight and input- and output-oriented.

Infrastructure: a serverless function does not persist in memory, nor does it have reserved resources for its exclusive use. A serverless function becomes alive only when it is activated and is "destroyed" once the execution is done.

Convenience: SSPs support many of the most popular languages, such as JavaScript, Node.js, Python, and offer user-friendly web-based or command-line interface (CLI) based IDE. A developer only needs to compose the code either directly in the web-IDE, or write code locally and then upload it to the platform using CLI commands, without any worry about other issues such as purchasing and maintaining hardware, and configuring code composing and executing environments.

Cost: since a serverless function does not reserve any resource, the user does not pay if the function does not run. In other words, the user only pays for the resource the serverless function actually consumes when it runs, which is very inexpensive. For instance, AWS Lambda charges only $0.0000166667 for every GB-Second (i.e., cumulatively used 1 GB of memory for 1 second).

Depending on where a serverless function is executed, a serverless service platform can also be classified as *cloud serverless*, where the serverless function is stored and executed on one of the SSP's data centers selected by the user before the function is developed; and *edge serverless*, where the serverless function will be pushed to the SSP's edge servers across the world, and will be executed on an edge server closest to the location where the trigger request is sent.

2.2 Denial of Service with Shared Resources

Interference between users is a necessary security concern on any platform with shared resources. In the era of cloud computing, more and more users are steering from private and dedicated resources, i.e., hardware and software, to public and shared ones due to convenience and cost considerations. Such transition, however, comes with associated potential risks. Some recent studies covered serverless computing in various aspects. For example, authors in [61] illustrated how SSPs use virtual machines or containers to isolate the functionality of different accounts. And authors [29] in pointed out that allowing code to flow to shared data storage is potentially tricky because it exacerbates the security management challenges associated with multi-tenancy and the potential for rogue code to collect signals across users. In short, if these shared resources are not handled securely, it could lead to a degradation of service across the cloud platform and affect other regular users who use that shared infrastructure or code.

The Warmonger attack proposed and studied in this paper shares similarities with the studies described above. In this paper, our contribution lies in the study of a problem that may have been overlooked on a new platform, and we used experiments to demonstrate the practicality and impact of such an attack.

3 THREAT MODEL

In this section, we provide details of the threat model of the Warmonger attack. Specifically, we discuss the scenario where this attack applies, the attack vector and attacker's objective, the victim, the impact, and the scope of this attack.

The attack model includes three participating entities, i.e., the attacker, the victim user, and the visitor; it also involves two platforms, i.e., the serverless service provider (SSP), and the target server. We introduce each component in the following.

3.1 Participants

In this subsection, we present the definitions and assumptions of all the entities involved in the Warmonger attack.

Serverless Service Provider (SSP): An SSP is a service provider that provides serverless computing service to the public. The SSP provides a platform and computational resources that allow a user to register, write and run serverless functions on this platform.

Target Server: A target server refers to a server hosting resources that are accessible by external applications. The target server is rather an abstract concept. For example, a target server could be a private server that runs on a standalone machine or on the cloud, such as an online SQL server containing private data. It could also be a public service, such as GitHub's API entry point (https://api.github.com) that a user uses to access account-specific service or information. Nonetheless, the target server's IP or domain name is public information and is known to the attacker. We assume that the target server is protected by an intrusion prevention system (IPS), which monitors the network traffic and reacts to anomalies by blocking the IP addresses of malicious and intrusive traffic.

User: A user is a client of an SSP who develops and deploys serverless functions to exchange messages with the *target server*. A *victim user* is an innocent user that uses an SSP's service and is affected by the Warmonger attack.

Attacker: The attacker is a user who has a valid account on the SSP's serverless computing platform. We assume that the attacker knows how to trigger the reaction of IPS systems.

Visitor: A visitor is an entity that utilizes the serverless function by sending a request to activate it. A visitor can be someone who visits a website. For instance, a website owner can include a triggering URL in the website's main page, such that once the page is visited, a request will be sent to activate a serverless function. In this case, the website owner is a *user* of the serverless function, and the one who visits the web page is a *visitor*.

3.2 Attack in Action

The attack vector of the Warmonger attack is straightforward. To launch this attack, the attacker composes a serverless function that sends malicious traffic (such as a large amount of HTTP requests, we will discuss more details of such possible "malicious behaviors" in Section 5) to the target server to inflict its IPS to block the attacker's IP. However, since the blocked IP is actually the serverless platform's egress IP that is shared among all users, such IP-blockage will cause the victim user unable to access the target server.

Essentially, the Warmonger Attack is a denial-of-service (DoS) attack, whose objective is to disrupt the victim users' operation rather than gaining any direct benefit. However, the Warmonger attack differs from a traditional DoS attack in the following aspects. First, in a traditional DoS attack, the attacker is usually an outsider who is not part of the victim. While in the Warmonger attacker, the DoS is caused as collateral damage made to the attacker herself. The attacker acts like a "warmonger" that inflicts the "war" between the serverless computing platform and the target server. Thus, we name this attack the "Warmonger attack". Second, in a traditional DoS attack, the attacker's goal is to at least partially paralyze the target server or exhaust its bandwidth, which becomes more and more difficult with all DoS countermeasures in place [11, 20, 38]. The Warmonger attack, on the other hand, aims to purposefully inflict the target server's IPS system to block its egress IP, which is much easier to achieve. In some sense, the Warmonger attack is the inverse of a traditional DoS attack: while a DoS attack attempts to find novel ways to circumvent the IPS and cause damage to the target server; the Warmonger attack uses all the well-known and existing DoS attack vectors to "inform" the IPS of the arrival of an attack to cause IPS-induced DoS for users with the same egress IP.

4 STATISTICAL ANALYSIS OF EGRESS IPS

Based on our description in Section 3, two hypotheses must be true for the Warmonger attack theory to be realistic: first, a serverless computing platform indeed lets its users share a reasonably small set of egress IPs, such that an IP, which is blocked due to the attacker's misbehavior, has high probability to be reused by the victim user; and second, the serverless function provides sufficient leverage to the attacker, such that she can really use a serverless function to conduct some obviously malicious behavior and cause her egress IP to be blocked. We explore these two key components in depth in the following two sections.

Specifically, in this section, we demonstrate our results of egress IP usage patterns of four major SSPs in the North American market, which are based on a number of months-long experiments we have conducted during July and November of 2020. In the next section, we investigate the possibilities and limitations of a serverless function's capability, and use experiments to demonstrate some easy-to-achieve behaviors that can potentially inflict protective reactions of IPS systems.

We will start this section with a brief description of the evaluated SSPs. After that, we will provide details of the experiment setup and then move on to demonstrate the analytical results and discuss our observations.

4.1 Evaluated Vendors

We have studied four major SSPs: *Cloudflare Workers*[13], *Amazon Web Service (AWS) Lambda*[5], *Microsoft Azure Functions*[41], and *Google App Engine*[26]. Cloudflare is a dedicated Content Delivery Network (CDN) provider, while the other three are comprehensive cloud service providers. Cloudflare Workers is edge serverless, i.e., a serverless function will be distributed to all Cloudflare's CDN edge servers across its CDN network, and executed on the edge server nearest to where the triggering request is sent. On the other hand, all the other three SSPs are cloud serverless, a user must specify which data center they would like a serverless function to be hosted and executed. Note that AWS also provides an edge serverless service, namely Lambda@Edge, which combines Lambda and Cloudfront, AWS's CDN service, such that a serverless function can be executed on Cloudfront's CDN edge servers. In this study, we did not cover Lambda@Edge and only evaluated Lambda.

4.2 Experiment Setup

The purpose of these experiments is to thoroughly analyze the egress IP usage of each SSP. Based on the threat model we have described before, we deem the following factors can affect the egress IPs used by a serverless platform to access an external server: the visitor's location, i.e., where the triggering request is sent; the location of the server where the serverless function is executed; and the location of the target server.

To simulate the visitor's and the target server's locations, we rented five AWS EC2 instances on AWS's different data centers and one Alibaba Elastic Compute Service (ECS) instance in China. The six locations include: Montreal (Canada) and Ohio (U.S.) from North America; Hangzhou (China) from Asia; Sydney (Australia) from Australia; Paris (France) from Europe; and Sao Paulo (Brazil) from South America.

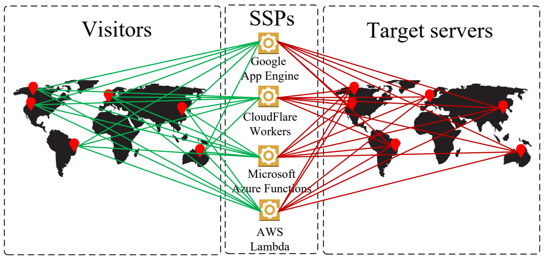


Figure 2: Egress IP Collection

4.3 Data Collection

The data collection scheme is demonstrated in Figure 2. Each instance servers both as a visitor and a target server simultaneously. As a visitor, each instance continuously sends requests to four serverless functions we have deployed on the four SSPs, and each serverless function will send six requests to the six instances once being activated. As a target server, each instance hosts a web server, which receives the requests sent by the four serverless functions and logs their source IP.

These experiments were run for roughly 3 months between mid-August 2020 and late October 2020. In order to achieve high granularity, we initially experimented with various intervals between two visitor-sent requests, ranging from 5 seconds to 40 seconds. We later found such small granularity is unnecessary and adjusted it to 600 seconds. Based on our analysis, although these 600 seconds intervals will cause some IPs not to be logged, it won't affect the overall trend. All the following demonstrations are based on 600 seconds intervals, and all previously collected finer-grained data was down-sampled to fit in the same scale.

In the following demonstration, we will use the format *visitor-SSP-target* to refer to a specific set of egress IPs. For example, *Montreal-Cloudflare-Paris* refers to egress IPs used by a serverless function on Cloudflare Workers' platform, when the function is triggered by a visitor in Montreal and the target server is in Paris.

4.4 Results

In the following, we analyze the egress IP usage patterns of the four SSPs based on our collected data.

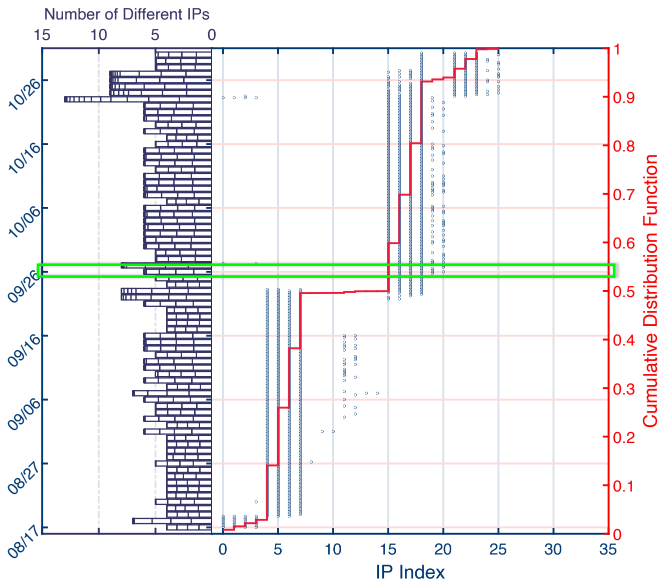


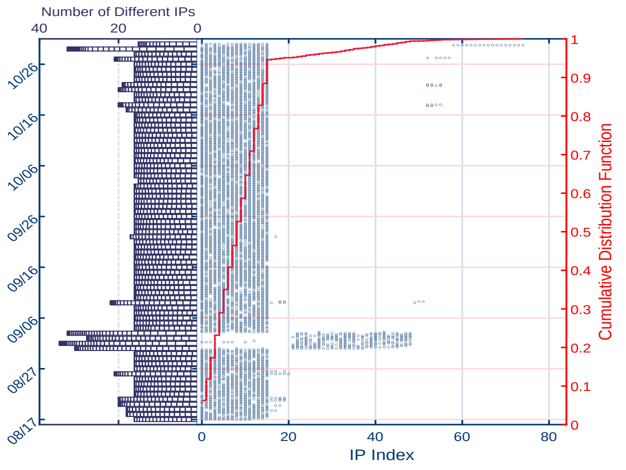
Figure 3: An illustration of Montreal-Cloudflare-Sydney

4.4.1 *Statistical Results Explained.*

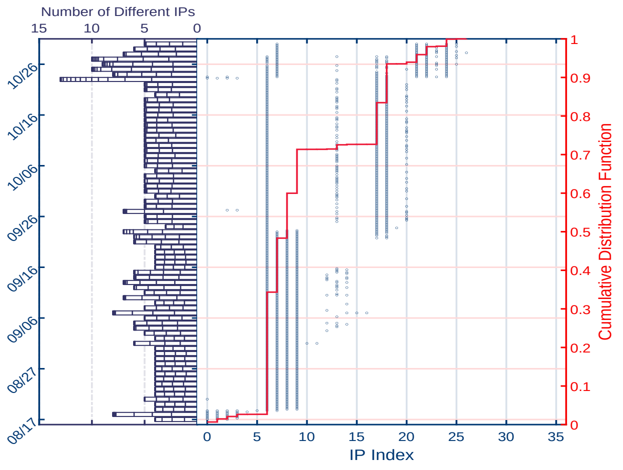
In the rest of this subsection, we demonstrate the results of our experiment using a highly-condensed figure as shown in Figure 3, which shows the egress IP usage pattern of *Montreal-Cloudflare-Sydney*. This figure integrates multiple dimensions and warrants a detailed explanation, which is provided below.

The figure contains three parts: a histogram plot (hist-plot) displayed on the left part of the figure; a Cumulative Distribution Function plot (CDF-plot) and a scatter plot (scatter-plot) that overlaps with each other and are displayed on the right.

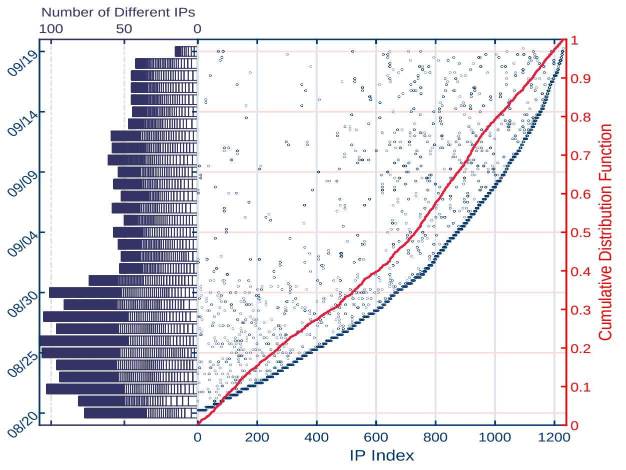
For the hist-plot, the y-axis (the left y-axis on the figure) represents the *date* of the experimented period, ranging from Aug 17,



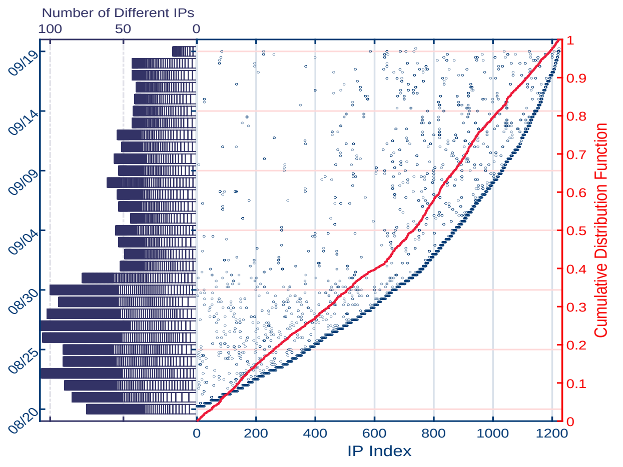
(a) Sao Paulo-Cloudflare-Ohio



(b) Montreal-Cloudflare-Ohio



(c) Ohio-Microsoft-Paris



(d) Sydney-Microsoft-Paris

Figure 4: Cloudflare and Microsoft Influenced by Visitor's Location

2020 to Oct 31, 2020 [1], with each tick represents one day. The x-axis (the top-left x-axis on the figure) represents the *number of different egress IPs* used on a specific day. For each bar of the hist-plot, we further divided it into multiple subsections, with each subsection represents the *frequency of appearance* of a specific IP on that day.

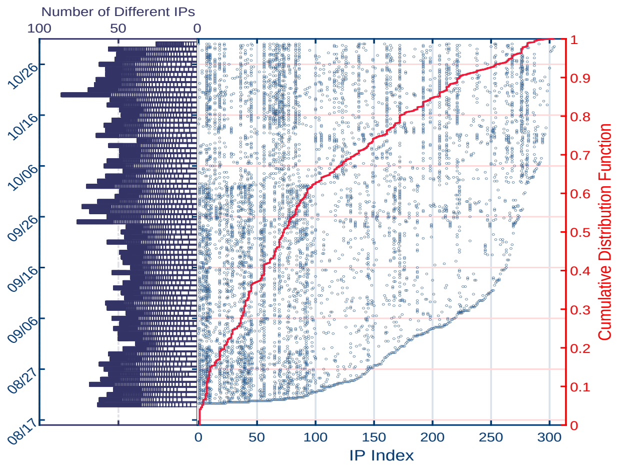
The hist-plot provides an intuitive view of *how many IPs were used, and how frequently each IP appears, on a specific day.* Take the date *26-Sep-2020* as an example, as highlighted in Figure 3. From the hist-plot, we can find 6 egress IPs were used on Sept 26 because the length of the bar is 6 according to its x-axis. This bar is divided into 6 subsections, but only 4 are visible and the other 2 are so short and packed at the left-end and cannot be clearly seen. The information conveyed by these subsections is that, out of the 6 IPs, 4 of them are used much more frequently than others. Furthermore, we can also tell these 4 IPs are evenly used because the 4 visible subsections have about the same length.

The scatter-plot is associated with the left y-axis and the bottom x-axis. The bottom x-axis shows the "indices" of different egress IPs. Specifically, for each different IP we observed during the experimented time period shown in the figure, we give it a unique integer index starting from 0 based on the first time it was seen. Also note that such indices across different figures are *not* comparable, i.e., IP-1 in two different figures may represent different IPs.

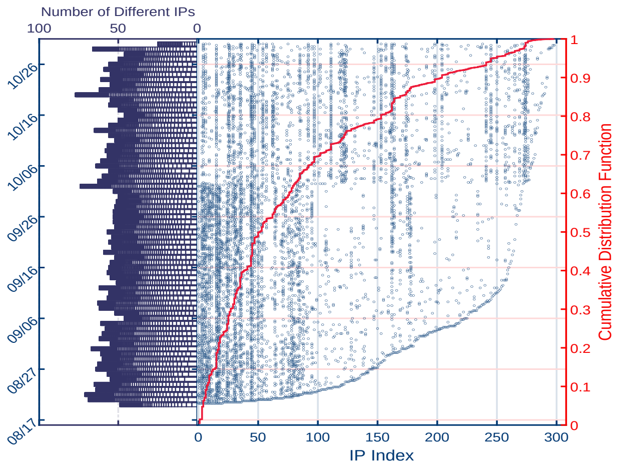
The scatter-plot gives details about the IP usage pattern that are not provided by the hist-plot. For example, using Sept 26 as an example again, we can find 6 IPs were used on that day based on the hist-plot, and from the scatter-plot, we can see these IPs are with the indices 15, 16, 17, 18, 19, and 20, as shown by the horizontal blue-circles. The scatter-plot also allows us to observe the day-to-day change of the IP usage pattern. For example, we can see the IPs 4, 5, 6, and 7 are the major four IPs used during Aug 19 to Sept 23, roughly a month; and in the next month, these four IPs were not used any longer and IPs 15, 16, 17, 18 became the new major-four. Based on the scatter-plot we may conclude Cloudflare Workers mainly uses only 4 IPs on this specific serverless platform and rotates the IPs on a monthly basis.

The CDF-plot is associated with the right y-axis and the bottom x-axis, which shows the overall use-percentage of a specific IP

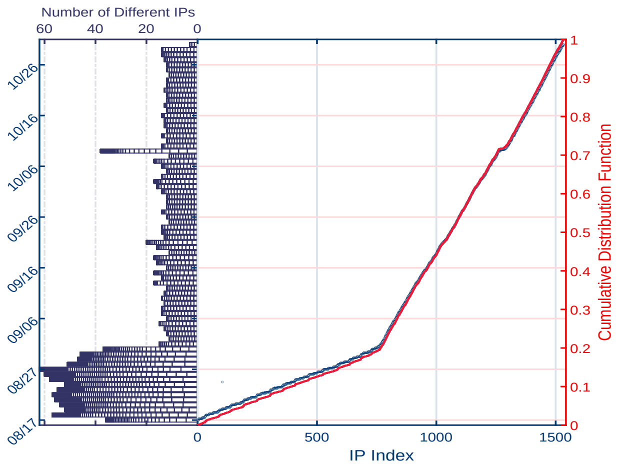
[1] A part of Microsoft Azure's data was corrupted and we only display a smaller range from Aug 19, 2020 to Sep 19, 2020.



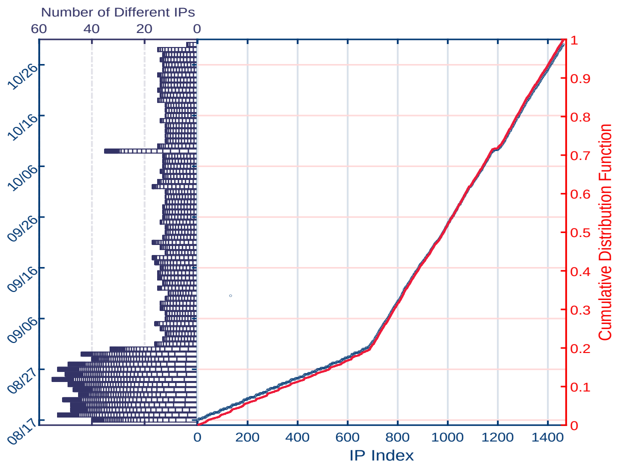
(a) Ohio-Google-Montreal



(b) Paris-Google-Montreal



(c) Sao Paulo-AWS-Montreal



(d) Paris-AWS-Montreal

Figure 5: Google and AWS Influenced by Visitor's Location

during the complete experimented period shown in the figure, such information cannot be obviously seen from either the hist-plot or the scatter-plot. For instance, from the scatter-plot, we can tell that a total of 25 IPs were observed during the experimented period, and IPs 4, 5, 6, 7, and 15, 16, 17, 18 are the most used 8 IPs, which, however, does not give accurate values of how frequently each IP were used. The CDF-plot provides this information in specific. For instance, we can observe the vertical length of the CDF-plot for IPs 4, 5, 6, and 7 are about 0.1 and are of roughly the same length, which indicates that each of these four IPs occupies 10% of the overall observed IPs during the period.

4.4.2 *Influence of Visitor's Location.*

As explained before, a serverless platform can be either *cloud serverless*, where the serverless function is stored and executed on an SSP's specific data center; or *edge serverless*, where the serverless function is executed on a distributed edge server. Intuitively, when sending requests to the same target server, edge serverless will use a different set of IPs if the visitor is at a different location, because the function will be executed on different edge servers; and cloud serverless will use the same set of IPs because the same data center executes the serverless function regardless from where the function is triggered. In this subsection, we validate our hypothesis by displaying the results by varying the visitor's locations but fixing the target server's location. Although we do not present all the results due to the space limit, we denote that the presented results representative and all other results follow the same trend.

Cloudflare is a CDN provider, and its Workers serverless platform is known to run on its distributed edge servers, making it an edge serverless platform. We compare two traces, i.e., *Sao Paulo-Cloudflare-Ohio* and *Montreal-Cloudflare-Ohio*, in Figures 4a and 4b. By observing these two figures, it is obvious that they do not share many similarities. Specifically, in the hist-plot of the *Sao Paulo-Cloudflare-Ohio* trace shown in Figure 4a, we see 18 IPs were used for most days, while there were only 5 for the *Montreal-Cloudflare-Ohio* trace in Figure 4b. Comparing these two figures, we can conclude that the edge servers located close to Montreal use much fewer egress IPs, which may be due to the fact that Sao Paulo has more than 10 times the population compared with Montreal and needs more egress IPs to accommodate more network traffic. On the other hand, however, the scatter-plot reveals the edge server in

Sao Paulo did not rotate its egress IPs, i.e., the same 18 IPs were used for almost the whole experimented period, with only 4 outlier-days from Aug 31 to Sep 3. For the Montreal trace, however, it is clear that only IP 6 was used for a longer time, other IPs, such as IPs 7, 8, 9, and 17, 18, were only used for one month. It is also important to denote that the IPs in Figures 4a and 4b are completely different without any overlap. Because we have seen thousands of different IP addresses during our experiment, it is not feasible to give each IP a unique index and plot them in one figure. Therefore, we decided to use the same index starting from zero on both figures, and verbally describe if those two sets of IPs ever overlap.

These observations help us to draw the following conclusions. It is harder for an attacker to cause complete IP blockage for the Sao Paulo edge server because of its larger number of egress IPs, however, once successful, the impact lasts much longer, which could be longer than the 3 months shown in our trace. On the other hand, it is quite easy for an attacker to cause complete IP-blockage for the edge server in Montreal since it only uses 4 to 5 IPs on most days, but whose impact can last only approximately one month.

Next, we present the results for the other SSPs, which exhibit non-intuitive and interesting differences.

First of all, all the other SSPs are cloud serverless, and a user must first select a data center to host the serverless function. Therefore, intuitively, one would assume the egress IPs should be the same regardless of where the request is from. However, we see different results in different SSPs. In the rest sub-figures in Figures 4 and 5, we present the results of the other three SSPs. In these figures, we intentionally selected traces different from the one used in Figures 4a and 4b to present more varieties of our results.

Although these figures look very similar to each other, it is important to point out that, the IPs between Figures 4c and 4d, and Figures 5a and 5b substantially overlaps (with more than 99% overlapping), while the IPs between Figures 5c and 5d merely have 0.7% overlap. In other words, Lambda will use different IPs to reach the same target server when the request is sent from different locations. One possible reason could be the serverless function on the Lambda platform may be executed on different hypervisors in the same data center, which was assigned different IPs, when the request is sent from different places.

These figures exhibit very interesting characteristics and we summarize our observations in the following.

We start our analysis by discussing the trend of the hist-plot. Looking at the hist-plot from the high-level, we can find Google does not present a clear trend of change on how it uses egress IPs. Although more IPs were observed in some days than others, the average IPs used through the experimented period is about 50. On the other hand, Azure and Lambda both show clear changes in how many IPs are used. For both Azure and Lambda, we see a significant drop in the number of IPs started from Sep 1, with Azure dropped from an average of 75 to about 50, and Lambda dropped from an average of 50 to about 15. However, it is unclear to us why there were such drastic changes in the IP usage pattern. Looking at the absolute numbers of IPs, Lambda uses fewer IPs, indicating it is easier to be attacked.

When we look closer to each bar of the hist-plot, we can find although each SSP uses many IPs in one day, some are used more frequently than others, which is indicated by the uneven distribution of the subsections in these bars (the dark-blue chunks on the left of each bar are subsections that are too short to be displayed individually). The exception lies in Lambda after Sept 1, where these (less than 20) IPs were used relatively evenly, as shown by most subsections can be clearly differentiated.

We then move on to examine the scatter-plot. Among the three, Lambda exhibits the most unique characteristic. On each day, Lambda would use a new set of IPs, and an old IP will be used at most for two to three consecutive days and then will not appear again. This IP usage pattern results in the line-like plot as presented in Figures 5c and 5d. For Google and Azure, both will use some new IPs each day that have never been seen before, which is indicated by the diagonal boundary line, while old IPs are still being used for a prolonged period of time, indicated by the scattered circles above the boundary line.

For Azure, new IPs are used each day, and such new IPs are used more frequently than old IPs, which is indicated by the clear dark-blue boundary on the scatter-plot. Old IPs will still be used for about 2 weeks and then gradually disappear, indicated by the trend that the circles above the boundary-line transit from dense to thin as time goes. For Google, such a trend is less clear. Each day, Google will also use some new IPs, but those IPs are not as heavily used as Azure, indicated by the fact that the boundary line is not as dense as that in Azure's case. And old IPs are still frequently used without any obvious trend to be abandoned. Observing Figures 5a and 5b, the scatter-plot for the first 100 IPs are slightly darker than the rest of IPs, indicating the first 100 IPs are used slightly more frequently than the rest.

At last, we look at the CDF plot. the CDF-plot of Azure trace presents a roughly straight line, indicating all IPs that appeared in the plot are used relatively evenly during the experimented period. The CDF-plot of the Google trace presents a slight concave at around IP index 100, which means the first 100 IPs are used slightly more frequently than the rest of IPs, which confirms our observation based on the scatter-plot discussed before. The CDF plot of Lambda presents a polyline with two phases. Both phases are straight lines, indicating all IPs are used more or less the same. The phase-change at around IP index 80 (or date Sep 1) is because of the sharp drop of the number of IPs since fewer IPs were used in each day after Sep 1.

These two cloud serverless platforms (excluding Lambda because it behaves similarly to edge serverless) exhibit completely different characteristics compared to Cloudflare Workers. First, since their egress IPs are independent of the visitor's location, an attacker can be at any location and still affect all the users across the world. In comparison, in Workers, an attacker is only able to attack the set of users that share the same edge server. Second, since all three SSPs use new IPs each day, the impact of the attack lasts much shorter compared to the case in Workers. For Lambda, in particular, the impact can only last for two or three days because each IP is used for two to three days at most (most IPs are used only for one day).

4.4.3 *Influence of Target Servers Location.*

A serverless platform may also use different IP addresses to access target servers at different locations. And whether or not the SSP uses different egress IPs for different locations may also affect

Table 1: Egress IP Overlap Ratio for different *visitor-SSP-target server* combinations.

	Montreal	Hangzhou	Ohio	Paris	Sao Paulo	Sydney
Montreal_Microsoft_6regions	99.93%	99.87%	99.87%	99.87%	99.87%	99.87%
Paris_Microsoft_6regions	99.93%	99.93%	99.93%	99.93%	99.93%	99.93%
Montreal_Google_6regions	99.68%	99.68%	99.68%	99.68%	99.68%	99.68%
Paris_Google_6regions	99.68%	99.68%	99.68%	99.68%	99.68%	99.68%
Montreal_AWS_6regions	66.72%	46.71%	66.88%	51.29%	55.31%	75.30%
Paris_AWS_6regions	73.49%	66.50%	73.56%	67.93%	69.02%	92.08%
Montreal_CF_6regions	21.40%	16.87%	17.70%	16.87%	19.34%	26.34%
Paris_CF_6regions	20.39%	15.13%	14.47%	14.64%	15.30%	18.59%

the effectiveness of the Warmonger attack. For instance, assume the target server is a cloud computing platform, says Google, which provides services in multiple locations that share the same IP blocklist. In this case, if the SSP uses the same IPs to access target servers in any location, the attacker only needs to attack one target server, and cause these IPs to be blocked completely by the entire cloud computing platform.

In this subsection, we analyze the egress IPs when the visitor's location keeps unchanged, but the visitor instructs the serverless function to send requests to target servers at different locations. We define the metric *Egress IP overlap ratio* as the percentage of egress IPs of a particular *visitor-SSP-target* among the entire IPs of *visitor-SSP-{all 8 targets}*. Our results show that different SSPs also exhibit different characteristics, as present in Table 1.

From Table 1, we can find that among these SSPs, Azure and Google do not differentiate egress IPs according to target server's locations, as shown the *overlap ratio* is very close to 1. On the other hand, we see clearly different behaviors for Lambda and Workers. Specifically, Lambda has overlap ratios ranging from 0.55 to 0.92, and Workers has an even smaller overlap ratio which is only around 0.2. This indicates that Lambda and Workers are more likely to use different IPs to access servers at different locations.

4.4.4 Feasibility Validation: Different Users/Serverless Functions.

In the previous subsections, we have thoroughly demonstrated different vendors' egress IP usage patterns. However, one may still question the feasibility of the proposed Warmonger attack: all the analyses done so far are from a single user's perspective, but can this user's misbehavior really affect others? Or in other words, do all the users really *share* the same set of egress IPs? This question leads to two scenarios, i.e., will the SSP use the same set of egress IPs when: 1). the same user-deployed serverless function is triggered by different visitors; and 2). different serverless functions on the same SSP platform are triggered.

Instinctively, the answer to these two questions should be both positive, because in today's Internet, IP addresses are scarce resources [50] and it does not seem reasonable for an SSP to assign one or a set of IPs to an individual user, especially considering the large user populations. To validate this hypothesis, we created two independent EC2 instances on AWS's Canada data center to emulate two different visitors who will send requests to activate serverless functions, and two independent but identical serverless functions on each SSP's platform. We conducted two experiments to simulate the scenarios described before, where in the first experiment we let the two visitors request the same serverless function on each platform; and in the second one we let one visitor trigger the two independent serverless functions on the same platform. Our results again show interesting differences among these SSPs that are described in the following.

Specifically, according to our experiment, a serverless function will always use the same set of IPs when being requested by different visitors. However, the two independent serverless functions share the same set of IPs on Azure and Workers platform but will use a completely different set of IPs on Google and Lambda platforms. We further investigated Azure and Workers platform by registering 5 different user accounts and deploy 2 serverless functions within each account, and still obtained the same conclusion, which suggests that Azure and Workers do share egress IPs among different users.

On the other hand, for Google and Lambda's case, at the first glance, they do not seem vulnerable to the Warmonger attack because egress IPs are not shared. However, we believe they still have the potential to be affected based on the reasons we listed below.

Recall that based on our IP usage analysis demonstrated in Figures 5a and 5d, we conclude that on Google and Lambda's platform, a serverless function will be assigned new IPs each day. Specifically, a serverless function will be given *some* new IPs each day while the old IPs will still be used for some days and then disappear (2 - 4 days on Lambda platform, and 10+ days on Google platform). However, because it is well-known that IPv4 addresses are scarce resources nowadays [50], it is reasonable to assume those IPs are not completely abandoned but are reused and reassigned to other serverless functions or other services on Google and AWS's cloud computing platform. In this case, if the attacker is able to cause a set of IPs being blocked by certain target servers for a long time, this would be like a time bomb and may cause unpredictable consequences depending on which user or what other cloud service uses these set of IPs and how they are used.

Furthermore, although such IP isolation among different serverless functions can prevent the platform-wide DoS, it is still vulnerable to a more targeted "point-kill". We leave a more detailed discussion regarding this attack in Section 6.3.

4.4.5 Launching Warmonger Attack.

The last question remains to be answered is how hard or easy it is for an attacker to really cause one specific IP to be blocked. Because the egress IP is automatically assigned to a serverless function by the platform and is out of the control of the attacker, it seems that the attacker may not be able to ensure the target server sees the same IP for each malicious request.

In fact, the attacker has two approaches to cause a specific IP to be blocked by conducting sufficiently *malicious behavior* against

Table 2: Numbers of IPv4 vs IPv6 for Cloudflare and Google . Cell format: {Workers-v4, Workers-v6}, {Google-v4, Google-v6}.

	Paris	Montreal	Ohio	SaoPaulo	Sydney	HongKong
Paris	{27, 28}, {14, 13}	{42, 44}, {10, 13}	{44, 39}, {15, 13}	{39, 28}, {13, 8}	{23, 41}, {16, 10}	{20, 42}, {15, 13}
Montreal	{11, 12}, {13, 11}	{18, 24}, {13, 14}	{24, 20}, {12, 13}	{23, 13}, {13, 7}	{11, 20}, {17, 10}	{12, 22}, {13, 13}
Ohio	{156, 144}, {14, 13}	{246, 222}, {13, 15}	{258, 240}, {13, 12}	{246, 157}, {12, 9}	{163, 248}, {19, 10}	{154, 244}, {14, 12}
SaoPaulo	{28, 29}, {12, 12}	{55, 54}, {12, 14}	{64, 59}, {14, 16}	{61, 31}, {11, 8}	{29, 60}, {15, 11}	{28, 52}, {15, 12}
Sydney	{98, 99}, {14, 11}	{163, 163}, {15, 15}	{178, 169}, {13, 14}	{173, 98}, {16, 9}	{102, 177}, {18, 11}	{102, 165}, {19, 12}
HongKong	{101, 102}, {12, 11}	{177, 183}, {13, 14}	{185, 178}, {13, 16}	{181, 105}, {11, 7}	{104, 179}, {16, 9}	{91, 174}, {14, 14}

the target server. We refer "malicious behavior" to any action that can trigger a protective reaction from a firewall or an intrusion detection/prevention system at the target server. An intuitive example can be sending a larger number of requests to the target server or constantly scanning its TCP/UDP ports.

The first approach is to behave maliciously in one serverless function activation. For instance, the attacker can instruct the serverless function to send 1,000 requests once it is activated. As we have tested on all platforms, as long as the serverless function is in a single execution, all the requests will have the same egress IP. However, this approach may not be applicable for Cloudflare Workers who limit the CPU time of one execution to be only 10 ms. The second approach is for the attacker to constantly activate serverless functions and conduct malicious behaviors on each activation. And the success rate of this approach highly depends on the number of IPs used in a specific time period, e.g., a day. If we simply assume all egress IPs are evenly used (which is not, the practical IP usage patterns are demonstrated by the hist-plots discussed before), in order to cause one IP to be blocked, the attacker will have to send malicious requests *n* times more than the target server's threshold where *n* is the number of used egress IPs. For instance, if the target server's threshold is 100 requests per hour and there are 10 IPs being used on the serverless platform, the attacker needs to send, statistically, 1,000 requests to cause any and all IPs to be blocked.

4.5 IPv6

We recognize that the deployment of IPv6 may be a potential mitigation to the Warmonger attack and conducted similar experiments to evaluate these SSP's IPv6 usage patterns. We first tested if a specific SSP supports IPv6 by using a serverless function to access an IPv6-enabled website, and found that only Google and Cloudflare support IPv6 at the time when the experiments were conducted. Microsoft and AWS gave "network is unreachable" and "address family not supported by protocol" errors, respectively, when being used to access an IPv6 website. The following experiments were only conducted on the two SSPs that support IPv6.

We began with assigning an IPv6 address to each of the six instances mentioned above, and then used each instance to activate a serverless function to access both IPv4 and IPv6 addresses of all other instances, similar to the experiments we have described in Sections 4.2 and 4.3. The experiments were conducted from July 24, 2021 to August 8, 2021, lasting 2 weeks. The request interval was set to be 600 seconds, similar to previous experiments. In Table 2, we present the comparison of the overall observed numbers of IPv4 and IPv6 addresses during the experimented period. To be concise, we put both Cloudflare's and Google's results together in one table cell. For instance, the top-left cell reads *{27, 28}, {14, 13}*, which means when we used the instance in Paris to access itself, we observed 27 IPv4 addresses and 28 IPv6 addresses used by Cloudflare Workers, and 14 IPv4 addresses and 13 IPv6 addresses used by Google App Engine. The results in Table 2 show that Cloudflare and Google use roughly the same number of (sometimes even less) IPv6 addresses albeit they have a much larger IPv6 address pool. We have also compared the detailed IPv4 and IPv6 usage patterns and found that they are very similar; therefore, we skip demonstrating the IPv6 usage patterns due to the page limit.

Our IPv6 experimental results reveal that despite the vast availability of IPv6 addresses in theory, the actual number of allocated/used addresses is quite limited, making the Warmonger attack a practical threat to current IPv6 designs by SSPs. We provide more detailed discussions on the IPv6 issue in Section 6.3.

5 WARMONGER ATTACK: EMPIRICAL EXPERIMENT

In the previous section, we have analyzed the egress IP usage patterns of different SSPs, which theoretically demonstrates the potential risk of the Warmonger attack. In this section, we use experiments to demonstrate the practicality of this attack.

5.1 Ethical Considerations

While the most straightforward approach to test the Warmonger attack's practicality is to deploy a serverless function and use it to attack a real target server, is obviously unethical to do so. To understand the influences of the Warmonger attack without causing real-world impacts, we design a *detached* experimental environment that can *indirectly* launch and evaluate this attack.

Specifically, we detach the serverless function and the target server by inserting a *surrogate server* of our own. The surrogate server is an EC2 instance running on AWS with a static IP, which acts as the target server to a serverless function and as a client to a real-world target server. All the offensive traffic, such as a large amount of HTTP(S) requests, will only be directed to the surrogate server to avoid any negative impact real users. And in the case we need to relay some traffic to a real target server, the worst case is only the surrogate server's IP be blocked. Furthermore, as demonstrated below, our analysis mainly stays on partial experiments, indirect analysis and comparison. We conducted very little amount of traffic forwarding via the surrogate server to a real-world target server without affecting any real user.

5.2 Approach

The basic idea of the Warmonger attack is for the attacker to purposefully misbehave and cause the target server to block the attacker's egress IP addresses. Although many simple brute-force attacks can very likely achieve this goal, such as simply instructing the serverless function to send an overwhelming number of HTTP(S) requests, it is not feasible to be tested due to ethical reasons even with our detached experiment setup, because this may violate the serverless service provider's user agreement. Therefore, the "experiments" in this section are combinations of real-world experiments, if we deem such experiment will not incur any real-world impact, and discussions and comparisons when a real-world experiment is not feasible.

5.3 Flooding

It is intuitive that flooding the target server is the easiest way to cause the sender's IP to be blocked. Flooding attacks can be launched on different layers. For instance, the Ping flood, or ICMP flood, exploits the ICMP protocol on the Network layer; the SYN flood takes advantage of the TCP protocol on the Transport layer; and the HTTP flood uses the HTTP protocol on the Application layer. The Application layer flooding is the easiest to apply, where the attacker only needs to use an application or a simple script to send HTTP(S) requests. The lower layer floodings, however, will require the attacker to have access to lower layers of the TCP/IP stack and modify the content of raw packets. During our experiment, we composed serverless functions on all the tested platforms attempting to access the *raw socket* in order to modify the raw packet, and found that none of the platforms allows us to access any resources below the Application layer. This may be because the SSP has purposefully disabled the access to lower layers for security considerations, or the serverless function platform is implemented as a virtualized sandbox that does not have a realized lower layer at all. Therefore, in the following flooding attack experiment, we only focused on evaluating the Application layer flooding attack.

Another potential barrier that may prevent the attacker from successfully launching the flooding-based Warmonger attack is the relative limitation imposed by the SSP and the target server. Intuitively, the SSP should also have a limitation on how a serverless function can be used. For instance, the SSP may disable a serverless function if it has been requested significantly frequently, or it sends too many requests per single execution. If such limitation is lower than the target server's IP-blockage limitation, the attacker will not inflict the target server's IPS reaction using the serverless function. To test such limitations of the serverless function platform, we designed the following two experiments.

In the first experiment, we composed a serverless function on each SSP platform, in which we used a `for` loop and the *concurrent* method (whenever possible, see later description for details) to send 500 HTTP requests to our surrogate server without any rate throttling. We then activated this serverless function and verified if it was executed and all the 500 requests were sent. In the second experiment, we changed the serverless function to send out one request to our surrogate server, but we ran a local script to activate this serverless function 500 times both using a `for` loop and *concurrent* method. The amount of 500 is arguably insufficient to be recognized as a real DoS attack and causes the IP to be blocked, but we were concerned that a larger amount of requests will make us be really recognized as a malicious user and cause our session or account to be suspended. Furthermore, as we show below, many real-world websites actually set the threshold much lower than 500.

We deployed these two experiments on the four SSPs. Our results show that only Cloudflare Worker cannot finish sending 500 requests in a single execution. This is due to Workers limits a single serverless execution to be no more than 10 ms CPU time [15], and we found only about 50 requests were sent before the serverless function is terminated by the system. For the remaining three, although we were unable to find any specification on such resource limit, we can make the empirical conclusions based on our experiments: Lambda supports *concurrent* execution and can send up to 100 requests per second, while Google and Azure do not support *concurrent* and can send only 5 to 8 requests per second.

Due to the reasons we have explained above, we think this experiment is infeasible to be tested in the real-world environment even with our detached configuration. However, we were able to find real-world examples that suggest reasonable and practical thresholds that are used for DoS detection. Specifically, in this Cloudflare technical blog [23], the author provides a number of Cloudflare customers' Firewall settings. From the blog, we can find that in reality, such thresholds are commonly set to be a very low value (only tens of times per minute). Therefore, it is very practical for an attacker to use a serverless function to inflict IP-blockage via HTTP flooding.

5.4 Malicious Requests

Besides flooding, another approach to cause IP blockage could be sending "malicious" requests to the target server. Because "malicious" is a vague term and it essentially depends on the target server's IPS configuration. In this subsection, we do not focus on any specific malicious behavior but discuss the capability that the attacker can achieve leveraged by the serverless platform.

5.4.1 *Malformed HTTP(S) Requests.*

A major functionality of the serverless function is to receive, process, and send HTTP(S) requests. Based on our evaluation, SSPs impose very few restrictions on how such requests can be constructed. As we have tested, we can instruct a serverless function to send HTTP(S) requests to any URL and include any special characters and keywords in almost any part of the request. For instance, we have tested to include the string `<script></script>`, one of the most straightforward patterns to test for potential web app vulnerabilities, as the query parameter of an URL, and as the value of the `Host`, `User-Agent` and `Cookies` headers, and find that only the change of the `Host` header are restricted by Google and Cloudflare. Since the strings we used are well-known offensive patterns that are commonly filtered by Web Application Firewall [34], it is reasonable to assume constantly sending requests with such strings is very likely to inflict IPS's protective reactions.

5.4.2 *Port Consuming and Scanning.*

Based on our investigation, none of the SSPs allows us to dive down and modify the transport layer and below, i.e., an attempt to access the raw socket will be rejected by the system with an "Operation not allowed" error message. However, we found that "SOCK_STREAM" is nonetheless allowed, indicating we can build a custom port scanner to constantly send port scanning packets to the target server. To demonstrate, we deployed Pfsense [21, 43], a popular open-source firewall on our surrogate server, and created a rule to block IPs who have more than 20 concurrent connections. Then, we attempted to create and maintain multiple TCP streams from the serverless function to our surrogate server on different ports and easily caused ourselves being blocked.

5.4.3 SSH.

SSPs also provide the convenience to allow a serverless function to establish SSH connections to a remote server [12, 42, 49], which provides another leverage for the attacker to conduct malicious behaviors. For instance, the attacker can use the serverless function to constantly SSH to the target server and sends apparently malicious commands, such as attempting to login with `root` and with various passwords, or including well-known malicious strings such as a series of `0x90` (the "nop slide" usually associated with buffer overflow attack [3]).

5.5 Censorship and State-Level Firewalls

The Great Firewall (GFW) of China is a well-known state-level censorship system that applies strict censorship rules. Although no explicit filtering rules have ever been published, many studies, such as [62] [58] [22], have revealed many behaviors based on empirical and extensive experiments.

It has been shown from existing studies that the Great Firewall applies filtering rules to both inbound and outbound traffic. For outbound traffic, i.e., traffic from China to other countries, the GFW deploys various rules ranging from simple IP and DNS filtering to sophisticated deep packet inspection that can inspect and filter sensitive keywords. For the inbound traffic, to the best of our knowledge, GFW applies at least keyword filtering.

The inbound keyword filtering rule makes it particularly easy for an attacker to launch the Warmonger attack against *all* target servers that are located within the GFW. To launch this attack, the attacker can find any web server that is located within the GFW and compose a serverless function to send an HTTP request with a filtered keyword being a parameter of the requested URL. Since an HTTP request is sent in clear text, GFW can identify the filtered keyword and henceforth block the sender's IP for a period of time.

To validate this hypothesis, we have conducted the following experiment. We first randomly selected a live website that is verified locates in China by checking its IP location, say `example.com`. Then, on the surrogate server, we use Python `requests` library to send an HTTP request with a parameter `?www.facebook.com` appended at the end, i.e., `http://example.com/?www.facebook.com`. Since `www.facebook.com` is a known sensitive keyword that GFW filters, such request will not get through but receive a "Connection reset by peer" error. Beyond this event, all subsequent requests, although without the `www.facebook.com` keyword, will receive the same error. Based on our experiment, such blockage is IP and port-specific, i.e., the above request will cause TCP connections to the website's IP with destination port number 80 being blocked. Using this technique, we can cause the GFW to block *any* port to this specific IP by simply specifying a different port of the HTTP request. For example, to cause port 443 to be blocked, we simply send `http://example.com/?www.facebook.com:443`. Note that it does not matter whether the web server recognizes or responds to requests sent to this port since the only purpose is to expose the filtered keyword and the port number to the GFW.

We ran further experiment to test out possible filtered keywords by referencing the "complete list of the blocked websites in China" [30] and used their domain names as the parameters, and find that 75 of the 141 domains can successfully trigger a GFW reaction. We also find that the average blocking period is around 120 seconds and such timer is reset on each new request. In other words, as long as a new "malicious" request being seen by the GFW every 2 minutes, the inbound connection will be perpetually blocked.

It is also noteworthy that other than a few hundred throttled requests, we did not attempt to aggressively send requests and comprehensively test the reaction of the GFW due to ethical considerations. However, it is reasonable to assume that more aggressive and intensive violations of the filtering rules may cause more serious punishments, which may eventually lead to prolonged time or even permanent IP blockage.

6 DISCUSSION

In the above two sections, based on our experimental results and surveys, we discussed and demonstrated the practicality and impact of the Warmonger attack. In this section, we discuss disclosure handling as well as potential mitigation approaches.

6.1 Responsible Disclosure

We disclosed our discoveries to the four SSPs through their corresponding security disclosure portals on April 6, 2021.

Microsoft contacted us on April 13 to request for a sample set of our identified egress IPs, and finally responded on June 29 that they use a large number of IPs and do not share them among users and closed this vulnerability report subsequently, which, however, contradicts our experimental results.

Google responded to our disclosure on April 14 stating that they have "complex protections in place to prevent such attacks" and decided to not track this "security bug".

AWS responded to our disclosure on May 23, mentioning that the same IP addresses can be shared by multiple clients, and they are aware of this risk and have an abuse detection and mitigation team to investigate fraudulent and malicious activities.

Cloudflare responded to our disclosures on June 8, commenting that it is a "weak point" in their architecture and they considered our disclosure informative but decided to close this report.

Based on our experiments presented in Section 5 and analysis in the following of this section, the Warmonger attack can be underestimated by an SSP to cause long-lasting practical impacts even when a detection system is deployed by the SSP. This is mainly because the attack focuses on exploiting a target server's detection system but not the SSP's, and the SSP's definition or recognition of malicious behavior may not be in perfect alignment with the target server's. We wish this paper can raise the broader awareness of the threat among users, service providers, and the research community.

6.2 Empirical Impact

The empirical impact of the Warmonger attack is directly determined by how popular serverless functions are used to access external servers. To this end, its impact depends on many factors and it is hard to derive its quantitative damage. On the other hand, however, there is a clear trend that the serverless computing market is experiencing unprecedented booming [16], and using a serverless function to access external resources is becoming a very common approach for various Internet applications. For instance, in [16], the author summarized nine "killer use cases" of AWS Lambda, which presented various use cases where a Lambda serverless function is used to access an external database, multimedia, and file servers. Besides, there are many other examples demonstrating practical use cases where serverless functions are used to access external servers for various purposes [9, 53, 54, 63]. Therefore, we believe significant damage can occur if an attacker is able to inflict the "war" between a popular serverless computing platform and an external server that hosts data frequently accessed by various users.

Interestingly, while we cannot launch the Warmonger attack in the real world to verify its validity due to ethical reasons, we were able to find indirect real-world evidence that the abuse of serverless functions can indeed cause their egress IPs to be flagged. Specifically, we randomly selected 1,000 IPv4 addresses from each SSP among our 2020 dataset, and 200 IPv6 addresses from Cloudflare Workers and Google App Engine among our 2021 dataset, and checked those IP addresses against the Composite Blocking List (CBL) [57] and the Scamalytics IP Address Fraud Checker [52], two online databases that detect and publish suspicious IP addresses in real-time. Respectively, CBL identified {2, 75, 259, 7}, and Scamalytics identified {193, 246, 63, 0}, IPv4 addresses from *{Cloudflare Workers, AWS Lambda, Google App Engine, Microsoft Azure Functions}* as suspicious, suggesting these serverless functions are indeed being used for actions that seemed suspicious to those two databases. We did not find any IPv6 identified as suspicious, which may be another indication that the usage of IPv6 is not as prevalent.

6.3 Existing and Potential Mitigation

In this subsection we discuss existing and potential mitigation strategies from both the SSPs' and the target servers' standpoint.

6.3.1 From SSPs' standpoint.

Based on the responses we have received from the four SSPs, it appears that at least Google and AWS were aware of this potential risk and have taken measures to mitigate it. The effectiveness, however, depends on how an SSP assigns and rotates IPs on their serverless computing platforms and among other cloud computing services. Taking Google App Engine as an example, although it does not share IP among users on any specific day, if it simply rotates the IPs among different users on a daily basis, an attacker can still cause the IPs assigned to her to be blocked on one day, and potentially harm other victim users in the following days. To this end, a better approach might be to let the IPs "cool down" for a while, i.e., do not immediately rotate IPs from one user to another, since in many cases IP-blockages should not be permanent.

It is also noteworthy that slowly rotating IPs, or even not sharing IPs among users, can only partially address this attack because they are still susceptible to a more specific "point-kill" attack as described in the following.

In today's Internet, many web applications use serverless functions to fetch resources from third-party repositories. For instance, [18] demonstrates how to use a Workers serverless function to fetch information from a user's GitHub repository and then post it to the user's Slack channel. The activation link is embedded in the user's Slack channel page, such that when the page is visited by a visitor, the visitor's browser will send a request to invoke the function, which then fetches, processes, and returns the user's Github information to the browser for display. Github, on the other hand, has an IP-based API request rate limit (60 requests per hour [19]). In this case, even if the serverless function is assigned with a unique IP that is not shared, an attacker can simply request the user's Slack channel 60 times, which will make Github to block the sesrverless function's particular IP and cause other visitors unable to view the user's Github information.

The SSP may also try to detect abnormal serverless function usage by itself. This approach, however, will not be effective as far as we are concerned. This is because anomaly detection by itself is not a very clearly defined approach, i.e., a behavior that seems normal to one server may be considered offensive by another. We have discussed in Section 5, some websites consider tens of requests within a minute as abnormal, which may not be agreed by a popular website that sees hundreds of visits per minute.

The SSP may also reduce the "leverage" by imposing more limits on the functionalities provided by a serverless function. For example, it can disallow a serverless function to SSH to a remote server. This, however, may cause the serverless computing service less attractive and eventually harm the SSP's profit.

All in all, we consider this issue is inherent in any service that shares resources among users, and one solution may be the "brute force" approach, i.e., providing sufficiently large IP pools such that the probability for an attacker to cause a real damage is small. To this end, it is similar to one of the ways to address DoS attacks: we make the bandwidth so broad that the attacker cannot bring enough traffic to overwhelm it. In practice, a holistic approach combining multi-facet protections (e.g., rotating IPs, cooling down IPs, enlarging IP pools and not sharing them, malicious behavior identification) should be adopted based on an SSP's hardware setups, software configurations, and service portfolios. By proposing this attack and conduct experiments to evaluate its impact in this paper, we aim to inform the community of this potential risk and taking necessary preventive steps before any real attack is launched.

6.3.2 From target servers' standpoint.

Some online resources, especially user-specific and private ones, can only be accessed with user credentials. For instance, in the previous Slack-and-GitHub example, if the user wants to make changes to her GitHub channel rather than to merely retrieve information, the request sent from the serverless function must include a user token, which is a string of random characters to authenticate a user, as a custom HTTP request header. In this case, a target server can apply a token-based rate limit; i.e., the server will block requests whose token has been seen more than the limit, but will not block the IP address from which the request is sent. For instance, GitHub allows 5,000 such "authenticated requests" per hour [19].

Such behavior of target servers may mitigate the Warmonger attack to some extend by making it harder to succeed. Specifically, in the case that the SSP does not share IPs among users, the SSP is still vulnerable to the "point-kill" attack we have explained earlier. However, the attacker now needs to request the user's Slack channel 5,000 times to saturate the new token-based limit. The attacker can use different approaches to make such a large number of requests without being flagged as malicious by Slack. For instance, the attacker can employ Internet bots to make such requests. Nonetheless, we denote that this ”point-kill” attack deviates from the main point of the Warmonger attack, and is less effective to inflict preventative reactions from a target server.

In the case that the SSP does share IPs among users, token-based rate limiting may not be effective because the target server may still block an IP address if malicious behavior, such as continuous port scanning, is observed from that IP.

6.3.3 *IPv6.*

Finally, it seems that the deployment of IPv6 can completely thwart the Warmonger attack because the SSP is able to afford a much larger number of IPv6 addresses and does not share them among users. This mitigation, however, stays in theory at least in the foreseeable future. Specifically, IPv6 can eliminate the Warmonger attack only if all three of the following requirements are met.

First, SSPs upgrade to IPv6, which is an ongoing process, especially for large SSPs. Based on the four SSPs in this paper, we found that Cloudflare Workers and Google App Engine now support IPv6, while AWS Lambda and Microsoft Azure Functions still do not.

Second, SSPs allocate a sufficiently large number of IPv6 addresses to their serverless function platforms. While theoretically possible, an SSP may face some technical and non-technical barriers, such as configuration complexity and cost considerations. In fact, all the four SSPs own a very large number of IPv4 addresses, but they only allocate a very small fraction to their serverless platforms [6, 14, 27, 40]. Likewise, we observed the same number of (sometimes even less) IPv6 addresses being used by these SSPs, as demonstrated in Section 4.5, suggesting that the Warmonger attack is still a real threat to their IPv6-based implementation.

Third, all external servers (i.e. target servers) also need to upgrade to IPv6. Otherwise, even if an SSP supports IPv6, the serverless function will still use IPv4 to access the target server. Considering the slow progress of IPv6 rollout [60], it is unlikely that this requirement will be met in the near future.

In conclusion, theoretically, IPv6 could eliminate the Warmonger attack with the proper setup. In practice, however, it is unlikely that IPv6 will completely replace IPv4 in the near future. Given the current IPv6 setup of SSPs, the threat of Warmonger attacks persists.

6.4 Limitation

Due to the limited resources and time, our study only covered four SSPs at six locations. Based on our results, we were able to see even for the same SSP, the IP usage patterns are drastically different at different locations. Therefore, the study we presented in this paper should be only viewed as a snapshot rather than the complete picture of the attack surface. We deem it very interesting to conduct a broader and more comprehensive evaluation to include more SSPs and locations, and we set these as our future direction.

7 RELATED WORK

Serverless computing is a new cloud computing business model and is under significant growth in recent years. On the other hand, very few studies focused on identifying or addressing security threats on this platform from either academia or the industry.

Serverless computing shares some similarities with CDN networking in the context of this study [2, 39]. In a typical operation, a visitor will send a request to a CDN edge server, and the CDN edge server will then fetch web documents from a web server and return them to the visitor. To this end, in [28, 31, 35, 59] , the researchers evaluated the egress IPs of CDN edge servers and observed similar results, i.e., a CDN edge also uses very few egress IPs to access external web servers. The Warmonger attack, however, is more worrisome because a user can instruct a serverless function to conduct many malicious behaviors that are not possible in a CDN.

Obtaining and analyzing the correct egress IP determines the likelihood of IP blocking attacks. Borgolte et al. explored how to obtain a specific IP on a cloud service [10]. After obtaining the correct egress IPs, threat intelligence datasets or IP blocklists may be used to label malicious, suspicious, or otherwise untrustworthy IPs [24, 37]. In addition, several existing studies related to co-location processing in the cloud raise the importance of security regarding shared resources in the cloud [7, 32, 51]. Sivaramakrishnan et al. have presented some studies on the side effects of blacklisting to detect and mitigate the impact of malicious Internet nodes [47, 48], whose results helped our study to identify suspicious egress IPs.

Several existing studies have explored the security of different serverless computing platforms [33, 46, 55]. Wang et al. evaluated metrics such as scalability, cold start latency, and instance runtime of SSP [61]. Furthermore, the source code of the function, some secrets stored in the container, or databases associated with the function may all be targets of event injection attacks [36, 45]. To address these issues, Alexandru Agache et al. proposed a new open-source framework that has been used in AWS Lambda to improve its security [1]. Kalev Alpernas et al. proposed a complex information flow control method for protecting serverless [4], and Pubali Datta et al. proposed a serverless network Valve for fine-grained monitoring of information flows [17]. To address the problem that serverless functionality is plagued by common vulnerabilities and failures in SDKs, third-party libraries, and platform code [8], Deepak Sirone Jegan et al. proposed a scalable security architecture of SecLambda for protecting and enhancing programs [56]. Our work complements this gap by exploring and dealing with the impact of shared egress IPs across different users of serverless vendors.

8 CONCLUSION

We presented the Warmonger attack in this paper. The Warmonger attack exploits the fact that all the users on a serverless computing platform share the same set of egress IPs. By purposefully misbehaving, an attacker is able to inflict external content servers to block these egress IPs and consequently cause all users on the same platform unable to access this external server. We conducted experiments to collect the egress IPs of four major serverless service providers, and analyzed their IP usage patterns. We further evaluated the potential moves an attack can take on a serverless platform. Our study shows that the Warmonger attack is a practical security threat that should be paid sufficient attention to by the security community and the serverless computing industry.

REFERENCES

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, pages 419–434, 2020.

[2] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. SAND: Towards high-performance serverless computing. In *2018 Usenix Annual Technical Conference (USENIX ATC 18)*, pages 923–935, 2018.

[3] Periklis Akritidis, Evangelos P Markatos, Michalis Polychronakis, and Kostas Anagnostakis. Stride: Polymorphic sled detection through instruction sequence analysis. In *IFIP International Information Security Conference*, pages 375–391. Springer, 2005.

[4] Kalev Alpernas, Cormac Flanagan, Sadjad Fouladi, Leonid Ryzhyk, Mooly Sagiv, Thomas Schmitz, and Keith Winstein. Secure serverless computing using dynamic information flow control. *arXiv preprint arXiv:1802.08984*, 2018.

[5] Amazon. AWS Lambda. https://aws.amazon.com/lambda/, 2021.

[6] Amazon. IP Ranges AWS. https://ip-ranges.amazonaws.com/ip-ranges.json, 2021.

[7] Yossi Azar, Seny Kamara, Ishai Menache, Mariana Raykova, and Bruce Shepard. Co-location-resistant clouds. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, pages 9–20, 2014.

[8] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017.

[9] Jeff Barr. Coca Cola things-go-better-with-step-functions. https://aws.amazon.com/blogs/aws/things-go-better-with-step-functions/, 2017.

[10] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. Cloud strife: mitigating the security risks of domain-validated certificates. 2018.

[11] Chen-Mou Cheng, HT Kung, and Koan-Sin Tan. Use of spectral analysis in defense against dos attacks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 3, pages 2143–2148. IEEE, 2002.

[12] Google Cloud. Securely connecting to VM instances. https://cloud.google.com/solutions/connecting-securely, 2021.

[13] Cloudflare. Cloudflare Workers. https://workers.cloudflare.com/, 2021.

[14] Cloudflare. IP Ranges Cloudflare. https://www.cloudflare.com/ips/, 2021.

[15] Cloudflare Docs. Cloudflare Worker Limits. https://developers.cloudflare.com/workers/platform/limits, 2021.

[16] David Correa. Serverless Architecture Market Is Expected to Reach $21.99 Billion by 2025, Says AMR. https://www.einnews.com/pr_news/526706260/serverless-architecture-market-to-reach-21-99-billion-by-2025-says-amr, 2020.

[17] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. Valve: Securing function workflows on serverless computing platforms. In *Proceedings of The Web Conference 2020*, pages 939–950, 2020.

[18] Cloudflare Docs. Build a slackbot. https://developers.cloudflare.com/workers/tutorials/build-a-slackbot, 2021.

[19] Github Docs. Rate limits for GitHub Apps. https://docs.github.com/en/developers/apps/rate-limits-for-github-apps, 2021.

[20] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, 2004.

[21] Tushar Subhra Dutta. Top 10 best open source firewall to protect your enterprise network 2021. https://cybersecuritynews.com/best-open-source-firewall/, 2021.

[22] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining how the great firewall discovers hidden circumvention servers. In *Proceedings of the 2015 Internet Measurement Conference*, pages 445–458, 2015.

[23] Alex Cruz Farmer. Rate limiting delivering more rules and greater control. https://blog.cloudflare.com/rate-limiting-delivering-more-rules-and-greater-control/, 2018.

[24] Álvaro Feal, Pelayo Vallina, Julien Gamba, Sergio Pastrana, Antonio Nappa, Oliver Hohlfeld, Narseo Vallina-Rodriguez, and Juan Tapiador. Blocklist babel: On the transparency and dynamics of open source blocklisting. *IEEE Transactions on Network and Service Management*, 2021.

[25] Geoffrey C Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv:1708.08028*, 2017.

[26] Google. Google App Engine. https://cloud.google.com/appengine, 2021.

[27] Google. IP Ranges Google. https://www.gstatic.com/ipranges/goog.json, 2021.

[28] Run Guo, Weizhong Li, Baojun Liu, Shuang Hao, Jia Zhang, Haixin Duan, Kaiwen Shen, Jianjun Chen, and Ying Liu. CDN judo: Breaking the CDN DoS protection with itself. NDSS, 2020.

[29] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651*, 2018.

[30] Ariel Hochstadt. List of blocked websites in China. https://www.vpnmentor.com/blog/the-complete-list-of-blocked-websites-in-china-how-to-access-them/, 2021.

[31] John Holowczak and Amir Houmansadr. Cachebrowser: Bypassing chinese censorship without proxies using cached content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 70–83, 2015.

[32] Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud. *IACR Cryptol. ePrint Arch.*, 2015:898, 2015.

[33] Abhinav Jangda, Donald Pinckney, Yuriy Brun, and Arjun Guha. Formal foundations of serverless computing. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–26, 2019.

[34] Robert RSnake Hansen Jim Manico. XSS filter evasion cheat sheet. https://owasp.org/www-community/xss-filter-evasion-cheatsheet, 2021.

[35] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. Unveil the hidden presence: Characterizing the backend interface of content delivery networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[36] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 427–444, 2018.

[37] Vector Guo Li, Matthew Dunn, Paul Pearce, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. Reading the tea leaves: A comparative analysis of threat intelligence. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 851–867, 2019.

[38] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 195–206, 2008.

[39] Garrett McGrath and Paul R Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410. IEEE, 2017.

[40] Microsoft. Azure IP Ranges Microsoft. https://www.microsoft.com/en-us/download/details.aspx?id=56519, 2021.

[41] Microsoft. Microsoft Azure. https://azure.microsoft.com/en-us/services/functions/, 2021.

[42] Vyom Nagrani. Scheduling ssh jobs using AWS lambda. https://aws.amazon.com/blogs/compute/scheduling-ssh-jobs-using-aws-lambda/, 2016.

[43] Netgate. Netgate pfSense Plus Firewall/VPN/Router. https://aws.amazon.com/marketplace/pp/B076TCMRWJ, 2021.

[44] Jussi Nupponen and Davide Taibi. Serverless: What it is, what to do and what not to do. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 49–50. IEEE, 2020.

[45] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid task provisioning with serverless-optimized containers. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 57–70, 2018.

[46] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 193–206, 2019.

[47] Sivaramakrishnan Ramanathan, Anushah Hossain, Jelena Mirkovic, Minlan Yu, and Sadia Afroz. Quantifying the impact of blocklisting in the age of address reuse. In *Proceedings of the ACM Internet Measurement Conference*, pages 360–369, 2020.

[48] Sivaramakrishnan Ramanathan, Jelena Mirkovic, and Minlan Yu. Blag: Improving the accuracy of blacklists. In *NDSS*, 2020.

[49] Sam Rhea. Announcing SSH Access through Cloudflare. https://blog.cloudflare.com/releasing-the-cloudflare-access-feature-that-let-us-smash-a-vpn-on-stage/, 2016.

[50] Philipp Richter, Mark Allman, Randy Bush, and Vern Paxson. A primer on IPv4 scarcity. *ACM SIGCOMM Computer Communication Review*, 45(2):21–31, 2015.

[51] Jungwoo Ryoo, Syed Rizvi, William Aiken, and John Kissell. Cloud security auditing: challenges and emerging approaches. *IEEE Security & Privacy*, 12(6):68–74, 2013.

[52] Scamalytics. IP Address Fraud Check. https://scamalytics.com/ip, 2021.

[53] Amazon Web Services. Irobot case study. https://aws.amazon.com/solutions/case-studies/irobot/, 2016.

[54] Amazon Web Services. Starling bank case study. https://aws.amazon.com/solutions/case-studies/starling-bank-case-study/?did=cr_card&trk=cr_card, 2020.

[55] Avraham Shulman, Ory Segal, and Shaked Yosef Zin. Methods for securing serverless functions, January 3 2019. US Patent App. 16/024,863.

[56] Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, Thomas Ristenpart, and Michael Swift. Guarding Serverless Applications with SecLambda. *arXiv e-prints*, pages arXiv–2011, 2020.

[57] Spamhaus. Composite Blocking List. https://www.abuseat.org/, 2021.

[58] Harsh Taneja and Angela Xiao Wu. Does the Great Firewall really isolate the Chinese? Integrating access blockage with cultural factors to explain Web user

behavior. *The Information Society*, 30(5):297–309, 2014.

[59] Thomas Vissers, Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. Maneuvering around clouds: Bypassing cloud-based security providers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1530–1541, 2015.

[60] W3Techs. Usage statistics of IPv6 for websites. https://w3techs.com/technologies/details/ce-ipv6, 2021.

[61] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 133–146, 2018.

[62] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. Your state is not mine: A closer look at evading stateful internet censorship. In *Proceedings of the 2017 Internet Measurement Conference*, pages 114–127, 2017.

[63] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. Building a chatbot with serverless computing. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, pages 1–4, 2016.