Journal of Information Security and Applications 60 (2021) 102843

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Denial of wallet—Defining a looming threat to serverless computing

Daniel Kelly *, Frank G. Glavin, Enda Barrett

School of Computer Science, National University of Ireland, Galway (NUIG), Galway, Ireland

ARTICLE INFO

Keywords:
Serverless computing
Cloud computing
Cloud security
Denial-of-wallet
Function-as-a-service

ABSTRACT

Serverless computing is the latest paradigm in cloud computing, offering a framework for the development of event driven, pay-as-you-go functions in a highly scalable environment. While these traits offer a powerful new development paradigm, they have also given rise to a new form of cyber-attack known as Denial of Wallet (forced financial exhaustion). In this work, we define and identify the threat of Denial of Wallet and its potential attack patterns. Also, we demonstrate how this new form of attack can potentially circumvent existing mitigation systems developed for a similar style of attack, Denial of Service. Our goal is twofold. Firstly, we will provide a concise and informative overview of this emerging attack paradigm. Secondly, we propose this paper as a starting point to enable researchers and service providers to create effective mitigation strategies. We include some simulated experiments to highlight the potential financial damage that such attacks can cause and the creation of an isolated test bed for continued safe research on these attacks.

1. Introduction

Serverless computing is an application deployment architecture that aims to provide pay-as-you go event driven functionality. Applications are developed as per each desired process and the event that invokes it. Serverless function platforms provide the infrastructure to deploy code for execution across their cloud and define the event processing logic that prompts the functions to run using the model: *event, trigger, and action*. Serverless computing abstracts back-end management from users, allowing only minimal access to some basic parameters such as function memory allocation and function run time timeout. Functions execute on the platform's traditional Infrastructure as a Service (IaaS) virtual machine (VM) offerings, however the provision of such VMs is managed by the platform in response to function invocation and not by the developer. Upon these execution VMs, a function container is spun up where the serverless function will execute. A container will be made for each function invocation and containers may be reused for repeat invocations. Unlike IaaS, you do not pay for the uptime and resources consumed by the "execution" VM or function container but rather for the run time of each function, hence the name Function as a Service (FaaS). This system allows for serverless applications to scale massively as the application is hosted on the serverless platform's cloud and as such has access to its resources. However, this has led to a threat unique to serverless computing, a new form of cyber-attack called Denial of Wallet (DoW). It may operate in a similar fashion to Denial of Service (DoS) attacks, where a bad actor initiates a barrage of requests to a service. In DoS, this would use up available resources rendering the service unusable. Although, as serverless applications are incredibly scalable, more and more functions will be invoked and quickly the application owner will incur a large bill exhausting their finances, hence the term Denial of Wallet attack.

We perform an evaluation of the damage that could be incurred by a DoW attack via cost analysis of the four major commercial serverless platforms; AWS Lambda,[1] Google Cloud Functions,[2] Microsoft Azure Function[3] and IBM Cloud Functions,[4] in various scenarios. We also devise a means of testing DoW attacks in a safe manner that would not result in real financial exhaustion by creating a serverless platform emulation.

2. Related work

To date, there has been no academic analysis of DoW attacks. A number of blogs have drawn attention to the issue [1–3] although they only go as far as to describe what DoW is. Pursec [4] released a list of ten security risks that face serverless computing with DoW featuring on that list. They do not go into detail on DoW specifically, instead focusing on DoS attacks and suggesting mitigation techniques such as:

* Corresponding author.
E-mail addresses: d.kelly69@nuigalway.ie (D. Kelly), frank.glavin@nuigalway.ie (F.G. Glavin), enda.barrett@nuigalway.ie (E. Barrett).

[1] https://aws.amazon.com/lambda/
[2] https://cloud.google.com/functions
[3] https://azure.microsoft.com/en-us/services/functions/
[4] https://cloud.ibm.com/functions/

https://doi.org/10.1016/j.jisa.2021.102843

Available online 1 June 2021

1. Writing efficient serverless functions, which perform discrete targeted tasks.
2. Setting appropriate timeout limits for serverless function execution.
3. Setting appropriate disk usage limits for serverless functions.
4. Applying request throttling on API calls.
5. Enforcing proper access controls to serverless functions.
6. Using APIs, modules and libraries which are not vulnerable to application layer DoS attacks such as ReDoS and Billion-Laughs-Attack

While these points serve as a starting point for protection against DoW attacks, they are not a solution. Writing efficient serverless functions will of course minimise functions with excessive run times. However, this does not mitigate against potential attacks spanning long timescales and high volumes of attacking nodes, that will inevitably drive the function run time up. Setting limits such as timeouts, disk usage and API throttling, shifts the attack from DoW (exhausting finances), to DoS (exhausting resources). Access control is only as effective as its difficulty in acquiring credentials. While administrator specific functions may be safe, API endpoints accessible to account holders of an application are susceptible to exploitation by fake users i.e. bots or "Sybils" (discussed later in Section 4.7).

Since no DoW attacks have been publicly recorded, at the time of writing, we are in a rare position of being ahead of the attackers. We can preemptively develop mitigation strategies rather than reacting to an attack when it occurs. As such, we must first theorise the mechanisms that malicious entities may use. We can achieve this be drawing inspiration from other well studied attacks. Application layer DoS attacks are the strongest contender for the starting point of effective DoW attacks. As mentioned in point 6 above, such attacks could be re-purposed for DoW use, potentially leaving APIs, modules and libraries vulnerable to such modified attacks.

Further related work that is valuable to the analysis of DoW attacks are on serverless security and DoS mitigation. Since we are ahead of any known large scale attacks, it is important to utilise all knowledge from related fields. Datta et al. [5] proposes a system for mitigating indirect function flow by monitoring the activity of every function API trigger. This research was directed at preventing information theft. However, analysis of all exposed API Gateways is necessary for DoW mitigation as any unprotected function triggers will serve as a point of attack. Inspiration for DoW mitigation may be taken from research carried out by DoS mitigation. Barna et al. [6] proposed a system for DoS mitigation that dynamically changes the rules on a firewall to minimise false positive detections and then pass suspicious activity to a CAPTCHA challenge. The application layer was the area of the web application being monitored. This makes this mitigation system relevant to DoW research, as DoW attacks are conducted on the application layer via API triggers. Their experimental setup of a mock application that they then attacked with real DoS tools served as inspiration for the experimental setup devised in this paper (see Section 6). As DoW attacks only aim to exhaust finances, slow rate attacks may be preferable as they are more difficult to detect as observed by Mukhopadhyay et al. [7].

3. Defining denial of wallet attacks

We define DoW attacks as the intentional mass, and continual, invocation of serverless functions, resulting in financial exhaustion of the victim in the form of inflated usage bills. Execution costs are increased via excessive function run time, invocation count and additional resource consumption. This is achieved by exploiting the massive capability for scaling, as the platform will handle the increased load but will incur costs in doing so.

A DoW attack is a form of attack unique to serverless computing. Given the scaling capabilities of serverless platforms, as a result of being hosted on a vast cloud of virtual machines, the effects of a regular DoS attack can be dealt with by massively scaling to handle the volume of requests. However, this scaling comes at a price. Serverless, or Function-as-a-Service, bills the application owner per run time of each function invocation. The resulting volume of function invocations following a DoS attack would therefore incur financial exhaustion of the application owner. As such, the term DoW came to describe this attack since unlike DoS, where the service is targeted and disrupted, the "wallets" (that is the finances) are the target. The result is an application owner who must now pay for function executions that are not in fact *legitimate* usage.

One of the earliest uses of the phrase in a publication is in the Open Web Application Security Project's (OWASP) report on the security risks of serverless computing [8]. It is mentioned as a risk to consider along with traditional DoS attacks. Prior to this, DoW was often referred to as a "financial exhaustion attack", "denial of capital" or simply "serverless denial of service". However, we believe that a "serverless DoS" attack and a DoW attack can be seen as unique threats, with the former focusing on serverless specific resource exhaustion and the latter being as we have described.

3.1. Preemptive examination of a threat

To date, there is little to no targeted research on DoW. We believe this is due to an absence of publicly known occurrences of attacks. This poses a unique opportunity to be ahead of attackers by mitigating against attack methods that are early in development. The vulnerability exists in serverless technology and it would be naïve to believe it will not be exploited. In Section 5.1 we discuss the potential motivations to conduct such an attack and potential scenarios they may be executed in.

Research on DoW will encompass strategically approaching the issue from the perspective of a bad actor and is an example of preemptive cybersecurity investigation. This domain shares many key attributes with bot detection, DoS mitigation and cloud security, meaning it also inherits their pitfalls and difficulties.

This research aims to bring a dormant threat in the conscience of the community and potentially address it before it may even begin and develop effective mitigation strategies before it can become a prominent threat.

4. Mechanisms of a potential attack and mitigation strategies

Since there have been no reported major instances of DoW, it is necessary to theorise attack strategies that may be developed. We take precedent from the techniques employed by DoS attacks and analyze their potential for modification as a DoW attack. Much of the strategies used in DoS attacks are viable in a DoW setting. However, we propose a number of unique cases that invariably distinguish DoW as its own potential threat.

4.1. Traditional attack methods

As serverless computing operates on the "No Ops" principle (that is the developer is not involved with the setup or maintenance of the application's back-end), popular traditional DoS attack patterns such as ICMP Flood, SYN Flood and UDP Flood [9] may not necessarily work as functions themselves and do not possess the infrastructure such attacks target. If the goal is to target serverless functions specifically rather than taking down an application as a whole, the most appropriate attack method would be HTTP flooding given that the majority of serverless functions work from API triggers. Two potential DoS attacks that could be re-purposed for DoW are:

ReDoS—Regular expression Denial of Service (ReDoS) [10] is an algorithmic complexity attack that produces a DoS by providing a regular expression (regex) that takes a very long time to evaluate. The attack exploits the fact that most regular expression implementations have exponential time worst case complexity, so for larger input strings (the 'evil regex'), the time taken by a regex engine to find a match increases exponentially. The aim of the attacker is to provide such regex(s) so that it takes an indefinite amount of computation time which in turn will either slow down the application or completely bring it down.

In a DoW context, this would drive the run time of the function as high as possible. Serverless functions have a timeout parameter that is an upper limit most commonly of five minutes. However, with functions being billed per 100 ms, this time limit is capable of inflicting notable financial damage.

Billion-Laughs-Attack—This is a DoS attack that targets parsers of XML documents [11]. The attack is also known as an XML bomb or Exponential Entity Expansion Attack (XEE Attack). An example attack consists of defining 10 entities, each defined as consisting of 10 of the previous entity, with the document consisting of a single instance of the largest entity, which expands to one billion copies of the first entity.

Again, this is another potential method in which the run time of a function can be driven up to cause a DoW.

These traditional attack methods serve as inspiration for one aspect of potential DoW attacks, that is, driving up the function run time. In order to maximise the effectiveness of such methods, they should be used in conjunction with some mechanism of increasing the number of function invocations.

4.2. *Short time span attack*

Short time span attacks are any form of "flood" attack. As many requests as possible are sent to the server as quickly as possible. This is the basis of DoS. A Web Application Firewall (WAF) is the recommended mitigation approach for large spike HTTP flooding attacks. An effective strategy against HTTP flooding DoS attacks is rate limiting requests from a single IP address via the WAF. Commercial platforms offer WAF services that implement safety against OWASP's top 10 security risks [8] and provide rate limiting rules such as AWS,[5] Google Cloud,[6] and Microsoft Azure.[7] These rules are difficult to set. A hard limit will protect against short time span (large spike in requests) HTTP flooding but at the risk of producing false positives in the event of actual real traffic spikes. Adaptive rules could be deployed but would become a target for long term false normative baseline inducing attacks (Section 4.5).

Rate limiting also exists within the API Gateway services themselves. These are generally limits that show off the service's ability to handle large traffic (AWS API Gateway can handle 10000 requests per second [12]). Unlike rate limiting rules on the firewall, if the API rate limit is reached, this could incur a DoS result as resources are taken up. Efforts to stop the effect of DoW by limiting the massive scaling factor of serverless applications could instead cause DoS as limits are reached.

4.3. *Long time span attack*

There are a number of "slow" DoS attacks, as described by Zargar et al. [13], such as Slowloris and R-U-Dead-Yet (RUDY). These attacks, through various means, aim to send high workload requests that lock up connection resources i.e. sockets. Analysis on such attacks has proved they are a threat [14].

A new strategy may be employed for long term DoW attacks that differs from the previous interpretation of "slow" attacks. We propose the idea of "leeching" as a potential form of DoW. A leech being a malicious program that continually triggers API endpoints that invoke functions. These leeches operate indefinitely and at a rate that would not be detected by reasonable WAF rate limiting rules. The key difference between DoW and DoS is that DoW does not need to execute all at once to consume all available resources to cause damage. As such, a leech DoW attack could be executed over a long time span and appear as legitimate traffic. We demonstrate the potential damage a leech DoW attack could create in Section 8.2.

4.4. *Distributed denial of wallet*

A long time span attack would be the most simple method of bypassing WAF rules and implementing a DoW attack. However, a single attacker or "leech" may not inflict financial damage on a scale that would affect larger organisations. Multiple leeches could be deployed onto a botnet either willingly, such as through IRC chat as seen in Operation Payback [15], or via malware that makes unknowing victims hosts for the leeches. A distributed approach to DoW would ultimately prove difficult to detect given how closely it would resemble real traffic using a long time span attack protocol.

4.5. *Adaptive mitigation deception*

In DoS mitigation systems that use an adaptive rule set i.e. can change the firewall rules base on fluctuation in "deemed to be real" traffic [6], a DoW attack can again utilise the fact that it can be executed over a long time span. Unlike a rigid firewall rule where it will be forever limited, if the attacker gently increases the intensity in such a way that the mitigation system adapts to it, it will then establish new false normative baselines.

4.6. *Serverless exploitation*

As well as API triggers, serverless functions can also be invoked through uploading to a storage service (such as S3 buckets for AWS Lambda). This introduces the concept of serverless specific attack patterns such as function input parameter exploitation, by flooding a storage service with images of varying size depending on the limits imposed by the function.

To fully realise the potential harm a DoW attack may be capable of, it should not be simply seen as a slow burning DoS attack. Instead it should fully exploit any oversights the developer may have made. If there is no limit imposed upon functions that process files, the run time of a function will increase. We show this effect in Section 8.1.

Similarly on Microsoft Azure Functions, the developer does not configure the memory allocation of a function unlike its competitors. This could similarly be abused by forcing a function to scale to a higher memory allocation to cope with larger inputs.

4.7. *Fake users*

A novel approach that could be taken towards performing a DoW attack would be the mass generation of fake users on a user subscription web application. Varol et al. [16] found that between 9% and 15% of Twitter accounts are bot accounts. That equates to roughly 49.5 million accounts based off the current monthly active user statistics [17]. Using a tool such as Selenium,[8] which automates browser activity, a mass of clients could perform a site walkthrough creating many fake users. Not only would this trigger many functions involved with profile creation but also, as discussed in Section 4.6, uploading the largest images possible (as profile pictures) can be used to increase run time. A further result of the mass generation of fake users would be access to other API endpoints such as page querying or other site functionality further falsely invoking more functions causing more damage.

[5] https://aws.amazon.com/waf/

[6] https://cloud.google.com/armor

[7] https://azure.microsoft.com/en-us/services/web-application-firewall/

[8] https://www.selenium.dev/

In recent studies, there have been three main approaches to fake user detection: graph-based, crowd sourcing and machine learning.

Graph-based detection is the use of graphs constructed from the social network of users to understand the network information and the relationships between edges or links across accounts to detect bot activity. There are many approaches to graph-based bot detection, a popular one being the use of “random walks”, for example SybilRank [18], Criminal account Inference Algorithm (CIA) [19] and SybilWalk [20]. SybilWalk claims a more robust classification with 99% of the top 80,000 nodes ranked in order of likeliness to be a Sybil, being correctly classified Sybils. However, in the ranking lists produced by SybilRank and CIA, only 0.3% and 30% are Sybils, respectively. Accounts are ranked by performing a random walk based movement on an undirected social graph. The idea of the random walk method is to label human users with benignness scores and Sybil users with badness scores. The score is then used in ranking the users.

Crowd sourcing is the use of humans to label whether a given user is a Sybil or a human. They identify, evaluate, and determine behaviors that would point towards an account being a bot. Alarifi et al. [21] recruited 10 volunteers deemed to have expert knowledge of Twitter (all BSc Computer Science graduates and active Twitter users) to rate and label 2000 random accounts captured via a Tweet streaming API. Using this, a ground truth data set was established for the further training and comparison of various machine learning algorithms such as C4.5 and Random Forest.

The machine learning method involves developing algorithms and statistical methods that can develop an understanding of the revealing features or behavior of social network accounts in order to distinguish between human- and computer-led activity. Ersahin et al. [22] used a Naïve Bayes based classification algorithm on a data set that had undergone preprocessing via Entropy Minimisation Discretisation (EMD). The results were 86.1% correct classification before EMD and 90.9% after.

5. Denial of wallet scenarios

5.1. Attack motivation

Motivations for attacks such as DoS are largely the same; some interested party wishes to cause a disruption to a victim’s service as a form of revenge, ransom, threat, retaliation or simply a desire to cause chaos. DoW is different in this regard. however, due to our proposed “slow-burn” and deceptive attack methods, we believe that the motivations to cause such damage are slightly more nuanced. We akin the motivation to perform DoW on a victim to that of espionage or sabotage. One party wishes to covertly effect another in a negative way thus giving the attacking party the upper hand when a critical matter arises.

One example we propose that best exemplifies this is that of a party wishing to stifle the growth of a startup company. A startup company offers some service powered by serverless computing. This service has potential to pull clientele from the attacking party. The attacking party may then initiate a DoW “leech” that will gradually ramp up fictitious usage of the startup’s service. This will drive up the startup’s serverless usage bill for no tangible return, unbeknownst to them. The startup will continue to pay for their serverless platform and may also invest in other areas given their new found usage. The attacker then ceases the DoW reducing the startup’s usage to a much lower level that does not justify the capital invested. The startup finds themselves in a position where they have spent a large amount of their budget on the serverless platform’s bills and may have taken loans backed by their previous fake success that they cannot pay. This will cause the stifling or even shut down of the startup.

This example can be applied to any situation where a smaller entity is vulnerable to their serverless powered service being targeted for DoW. Given that the attack’s most powerful trait is their ability to go unnoticed, it could be used as a tool for larger politically interested entities to silence or weaken any opposition. For example, finances spent on inflated serverless bills is money not spent on organising demonstrations etc.

5.2. Attack execution

For DoW to occur, a malicious entity must gain access to function endpoints. This can be broadly achieved via knowledge that certain services use serverless functions e.g. Seattle times image loading.[9] More targeted approaches could be taken to extract specific endpoints of functions via web scrapers looking for direct API links that follow the typical naming convention e.g.

AWS—https://xxxxx.execute-api.xx-xxxx-x.amazonaws.com/xxxxx/xxxxx

Azure—https://xxxxx.azurewebsites.net/api/xxxxx

GCF—https://xx-xxxxx-spherical-plane-xxxxx.cloudfunctions.net/xxxxx

It is good practice amongst industry professionals not to leave API URLs visible like this, however, such a vulnerability could still exist through human error.

The core mechanism of a DoW attack will be to spam function triggers. We consider two likely scenarios for this to occur.

Scenario 1: Function API endpoint URLs have been scraped or leaked. These URLs are loaded into spamming software that continuously hits these endpoints with HTTP requests. The rate of attack can vary depending on the security of the application. In the presence of no DDoS security (rate limiting etc.), fast paced attacks can take place but will raise suspicion when an application owner sees a sudden spike. Slow rate attacks, “Leeches”, are effective for DoW due to their difficulty of detection. They are akin to “Bad Bot Traffic”, fictitious traffic generated by bot nets intended to give false analytics. As such, slow rate DoW has a double impact of driving up operational costs with function invocations and spoofing application owners with falsified traffic (potentially causing further financial damage when used to justify increased spending).

Scenario 2: Developers have successfully hidden or obscured API URL endpoints or the endpoint is a different trigger such as upload to a storage system. Attacks will need to be performed as a fake application user. Web application testing software such as Selenium[10] can automate interactions with a web application. Applying a similar tactic to a bot net will produce fictitious traffic as with the URL example. The downside to this is that it requires greater computing resources and knowledge of which applications use serverless functions and how they are triggered.

6. Isolation zone for denial of wallet attack testing

As DoW is a financial attack, it is impractical to test such on real commercial platforms. Given that there are no known historical examples of DoW, we must have a means of generating data for training of mitigation systems and then validating such systems. To facilitate this, an isolated serverless platform was devised that will emulate the cost damage of DoW. This consisted of four desktop PCs running in the same network. One machine was designated the “victim”. It would play host to the serverless platform and cost emulator being targeted by our attacks. The other three machines were designated as “attackers”. It is from these machines that we will launch a variety of attack strategies to determine the best candidacy for DoW. The topology of our test bed is illustrated in Fig. 1.

[9] https://aws.amazon.com/solutions/case-studies/the-seattle-times/
[10] https://www.selenium.dev/

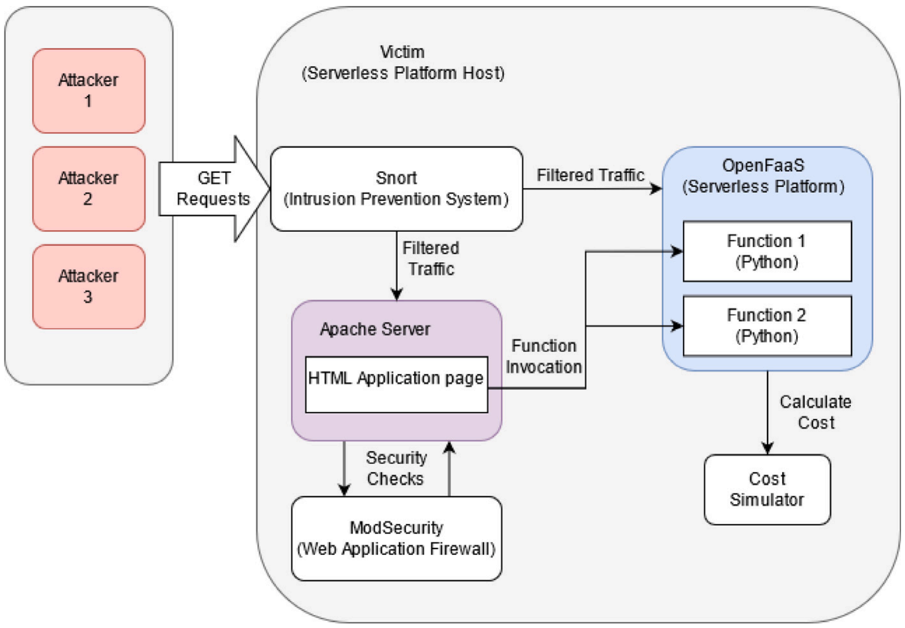

Fig. 1. Test bed of an isolated serverless platform for attack simulation.

6.1. Configuration

The "victim" machine is a Linux desktop running the Ubuntu 18.04 operating system. The serverless platform used to execute functions is OpenFaaS.[11] OpenFaaS is a framework for building serverless functions with Docker and Kubernetes which has first class support for metrics. Any process can be packaged as a function enabling you to consume a range of web events without repetitive boiler-plate coding. The mock application that will trigger the functions on OpenFaaS runs on Apache Server 2.[12] The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. In order to perform more meaningful experiments on DoW, our environment has a twofold security system. Where commercial platforms offer one Web Application Firewall (WAF) that can run on all its products e.g. AWS WAF,[13] finding a single catch-all solution to secure both our application server and serverless platform proved difficult. Our solution was to secure the Apache server with the ModSecurity v2[14] WAF and use the Intrusion Prevention System (IPS) Snort v2[15] to perform WAF actions on the OpenFaaS deployment.

6.2. Serverless deployment

OpenFaaS was chosen as the serverless platform for executing function code. Some of its highlights include;

- Ease of use through UI portal and one-click install
- Write functions in any language for Linux or Windows and package in Docker/OCI image format
- Portable—runs on existing hardware or public/private cloud with Kubernetes or containerd
- CLI available with YAML format for templating and defining functions
- Auto-scales as demand increases

[11] https://www.openfaas.com/
[12] https://httpd.apache.org/
[13] https://aws.amazon.com/waf/
[14] https://modsecurity.org/
[15] https://www.snort.org/

OpenFaaS runs on the "PLONK" stack. PLONK is a Cloud Native stack for building applications which stands for:

- Prometheus—metrics and time-series
- Linux—OS or service mesh
- OpenFaaS – management and auto-scaling of compute – PaaS/FaaS, a developer-friendly abstraction on top of Kubernetes. Each function or microservice is built as an immutable Docker container or OCI-format image.
- NATS—asynchronous message bus/queue
- Kubernetes—declarative, extensible, scale-out, self-healing clustering

The function gateway can be accessed through its REST API, via the CLI or through the UI. All services or functions get a default route exposed, but custom domains can also be used for each endpoint. Prometheus collects metrics which are available via the function gateway's API and which are used for auto-scaling. Functions are invoked from the application front end via an API gateway. This also exposes the functions to direct invocation (as is normal in FaaS). The front end and OpenFaaS communicate via HTTP requests and JSON.

6.3. Web application server deployment

The Apache server hosts the application front end. It is a powerful, flexible, HTTP/1.1 compliant web server that implements the latest protocols, including HTTP/1.1 (RFC2616). It is highly configurable and extensible with third-party modules, which we make use of when implementing our security solution. Apache runs on Windows 2000, Netware 5.x and above, OS/2, and most versions of Unix, as well as several other operating systems, meaning we could easily pair it with our OpenFaaS deployment on Ubuntu 18.04. This server will be used to host mock applications for future experiments.

6.4. Security

As previously mentioned, where commercial platforms offer a single WAF that protects all their products, this proved unrealistic for our environment. We split the security measures to 1. protect the application server and 2. protect the serverless platform.

The application server is protected by ModSecurity. It is a toolkit for real-time web application monitoring, logging, and access control. Its capabilities include:

- Real-time application security monitoring and access control
- Full HTTP traffic logging
- Continuous passive security assessment
- Web application hardening

We use this as the primary means of securing the application from attacks such as XML or Shell injection etc.

A highly recommended means of mitigating HTTP Flood attacks (which is the core of worst case scenario DoW attacks) is rate limiting based on IP address. For this, all inbound traffic gets filtered through Snort. Snort is an open source network intrusion detection system (NIDS). Snort is a packet sniffer that monitors network traffic in real time, scrutinising each packet closely to detect a dangerous payload or suspicious anomalies. Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IP traffic sniffers and analyzers. Through protocol analysis and content searching and matching, Snort detects attack methods, including DoS, buffer overflow, CGI attacks, stealth port scans, and SMB probes. Running in "inline" mode, Snort becomes an intrusion prevention system. Inline mode is where Snort sits between two bridged network interfaces, analyzing network packets as they come in. When suspicious behavior is detected, it will drop any packets that trigger certain rules. Otherwise it will pass them to the Apache server or OpenFaaS back end (depending on the function invocation method).

6.5. *Serverless platform pricing emulator*

We developed the Serverless Platform Pricing Emulator in order to compare the effects of DoW across multiple commercial platforms. It is a program that sits on top of an OpenFaaS serverless deployment that allows you to emulate how much its function executions would cost on AWS Lambda, Google Cloud Functions, IBM Cloud Functions and Microsoft Azure Functions. It is written in Python and converts usage metrics gathered by Prometheus on the OpenFaaS platform to a comparable cost had those functions been invoked on the four largest commercial serverless platforms. The cost calculator queries Prometheus for the total number of successful function invocations and the cumulative execution time. As all functions are configured to 128MB, delineation between function executions is not required and a total cost can be calculated as per each platform's pricing guidelines. The source code[16] is available for use by the community who wish to perform similar experiments on serverless computing financial effects.

6.6. *Attacking nodes*

The attack machines execute a python script that sends multiple GET requests to the functions and change the IP address of the machine (simulating any number of attack nodes required). For DoW, slow rate attacks are the primary concern, however, DoS style of attacks may also be executed.

Attacker may also be configured with the Kali Linux operating system[17] for use of existing attack software. In Section 7, we do so and utilise the DoS tool Low Orbit Ion Cannon.

7. Demonstrating denial of wallet

Using our test bed, we can demonstrate the mechanism of DoW attacks. Firstly, we utilise an existing DoS tool, Low Orbit Ion Cannon (LOIC), a DoS tool made famous by the "Operation Payback" attacks in defense of Wikileaks in 2010, that can perform HTTP flooding attacks [23]. We configured a single node with Kali Linux and ran LOIC with a single serverless function on our OpenFaas platform as the target. We allowed LOIC to execute for one hour, continuously spamming direct function invocations to our OpenFaaS deployment. Our platform gathers metrics such as function invocation rate, container scaling, total invocations and execution time using Prometheus and Grafana. We observed that over 130,000 function invocations could be triggered in this time span (Fig. 2). Average invocation rate and run time per 20 s interval is also recorded(Figs. 3 and 4). This run demonstrated how a serverless platform scales to accommodate large volumes of requests by creating additional containers for function execution (Fig. 5) However, this example is a worst case scenario where a single attacker is allowed to continuously spam requests. A Web Application Firewall (WAF) can mitigate simple GET request flooding using rate limiting. Using LOIC again and activating our intrusion prevention system (IPS) that acts as a WAF to our OpenFaaS deployment, we can limit the damage any one node can do with an HTTP Flooding attack by rate limiting requests from a single source. This would suggest that short time span, single node attacks are not suitable for DoW.

Table 1
Effect on run time of increasing image size.

Image size $l \times w$ (px)	Test 1 (ms)	Test 2 (ms)	Test 3 (ms)	Test 4 (ms)	Test 5 (ms)	Average (ms)
540	248.27	230.51	254.77	239.32	236.22	241.818
1080	371.18	368.43	361.15	385.22	409.24	379.044
2160	1045.59	1051.62	1083.57	1046.05	1059.11	1057.188
4320	2945.17	2958.46	2954.22	2963.07	2933.42	2950.868
8640	9901.85	9843.55	9924.92	9877.68	9881.59	9885.918

From this demonstration, we can infer that an unprotected serverless function is susceptible to DoW from a single node. Implementing rate limiting protects against the issue. However, as previously discussed in Section 4.3, DoW can be executed over a much longer time span than a DoS attack, thus rate limiting will be ineffective on such cases.

8. Theoretical damage analysis

8.1. *Serverless function input parameter exploitation*

A common use case of serverless functions is in image re-processing, such as creating a thumbnail from a user uploaded image [24]. We use AWS Lambda as an example to demonstrate how one could exploit input parameters to a serverless function by uploading large images that must be resized to a thumbnail size. For this, an AWS Lambda function was created that resizes images uploaded to an AWS S3 bucket to a 128 × 128 px thumbnail [25]. The thumbnail image is then stored in its own separate bucket. Images of increasing size were uploaded and we observe the effect on function run time. The function memory allocation was kept constant at 512MB as we found lower memory allocations could not handle the largest images. Results are shown in Table 1.

As expected, there is an increase in run time with an increase of image size. In this scenario, it would fall to the developer to impose a hard limit on image size. However, as smartphones are producing higher resolution images with each new flagship device, for example, the baseline of an acceptable maximum image size gets pushed higher. This makes even well thought out functions susceptible to input exploitation.

8.2. *Cost analysis of excessive function invocation*

As a reference, an Apple iPhone X takes photos producing an image of 4032 × 3024 px [26]. Cross-referencing our prior experiments on input exploitation (Section 8.1), we can take a run time of 2950 ms of a memory allocation of 512MB as a benchmark for a legitimate function that a malicious user would consider targeting for a Denial-of-Wallet attack.

[16] https://github.com/psykodan/openfaas-commercial-platform-emulator
[17] https://www.kali.org/

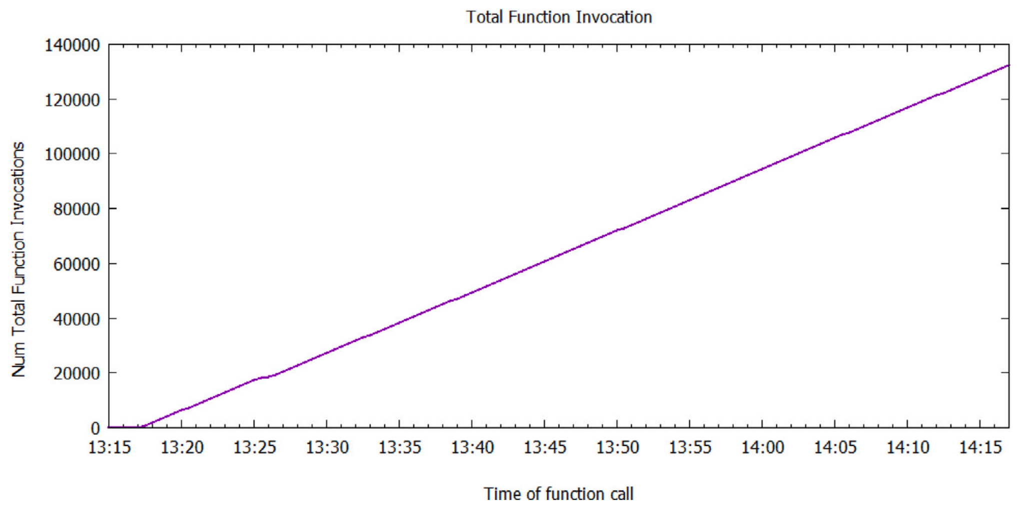

Fig. 2. Total function invocations gathered from HTTP flood attack with no protection.

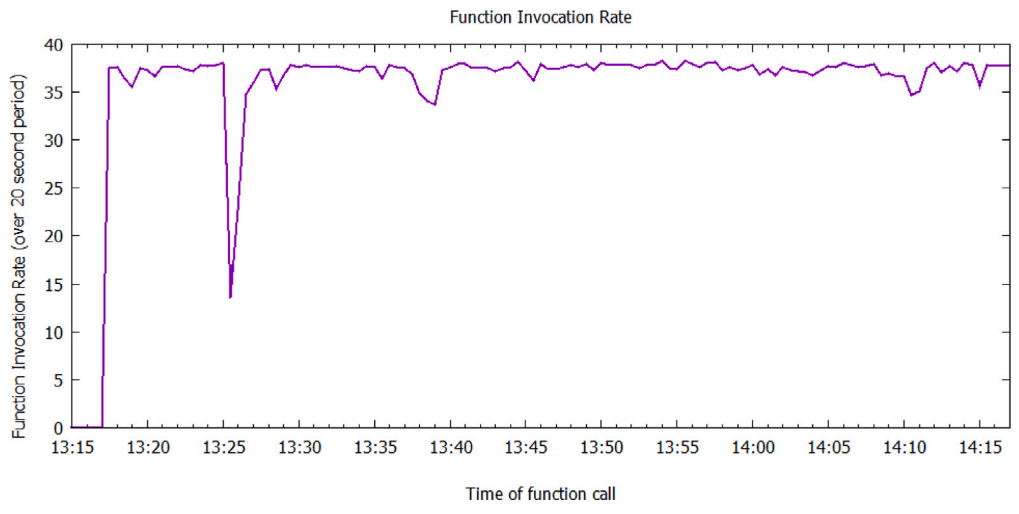

Fig. 3. Function invocation rate per 20 s interval gathered from HTTP flood attack with no protection.

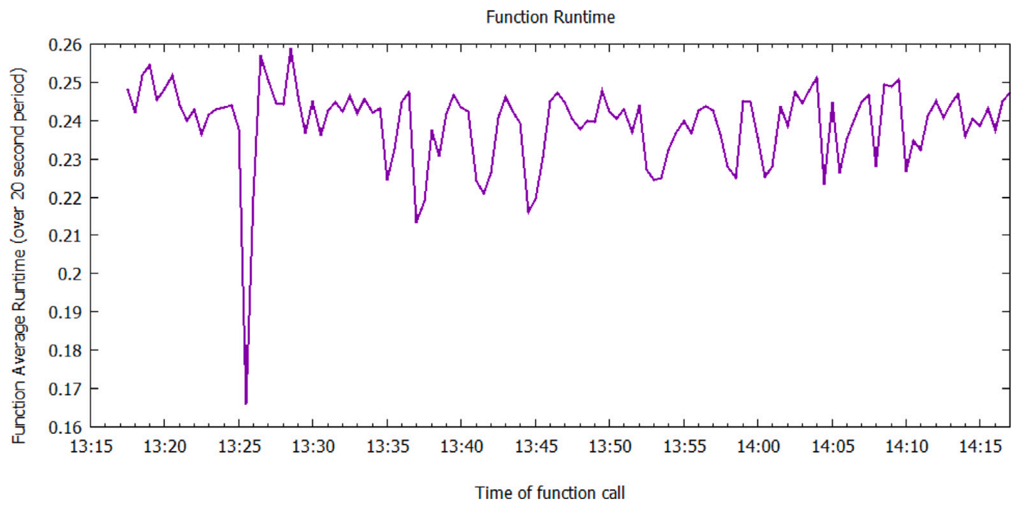

Fig. 4. Average function run time per 20 s interval gathered from HTTP flood attack with no protection.

A cost analysis of this function on the four largest commercial serverless platforms; AWS Lambda, Google Cloud Functions, Microsoft Azure Functions and IBM Cloud Functions, was performed for increasing number of function invocations using the Serverless function cost

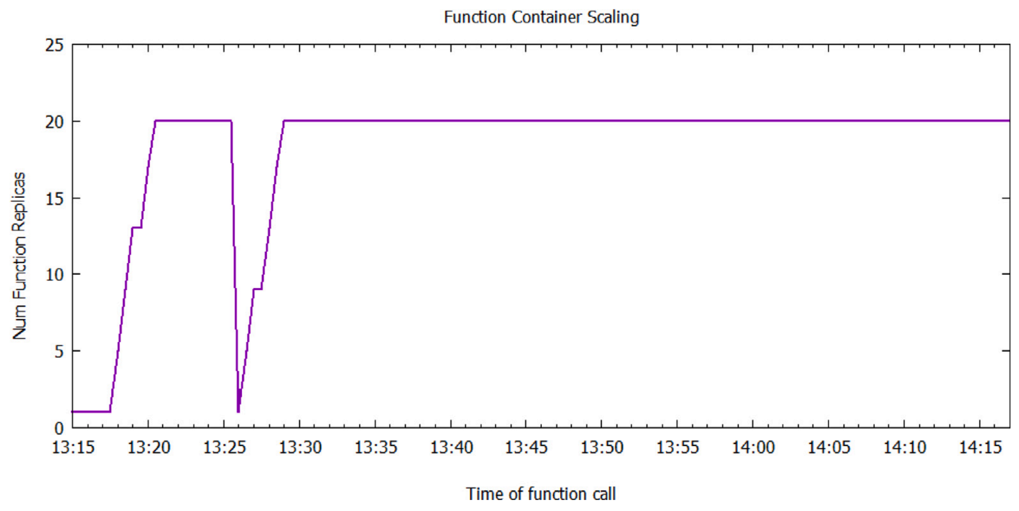

Fig. 5. Metrics gathered from HTTP flood attack with no protection. Function replica generation.

calculator we devised as the mock application on our isolated test bed (Section 6).

This calculator takes the number of function requests, the run time of each of those functions and the memory allocation of the function as inputs and returns the total cost of executing those functions inclusive of all free allowances granted by each platform and additional costs such as API call costs. The official cost guides for each platform were used; AWS Lambda,[18] Google Cloud Functions,[19] Microsoft Azure Function[20] and IBM Cloud Functions.[21]

The three graphs show a linear increase of cost incurred with Google Cloud functions resulting in the greatest charges followed by AWS Lambda, IBM Cloud Functions and lastly Azure Functions. With a modest bot-net of 1000 nodes, a slow rate attack of 2000 requests per hour will cost an application owner roughly $40,000 after one month and between $400,000 and $500,000 if left unchecked for a year. A bot-net of 10,000 nodes will do the same damage in one month that 1000 nodes would do in a year.

9. Discussion

DoW is a strong potential threat, as shown by our preliminary experiments on the subject. It is a rare opportunity to be ahead of a threat, as such it is necessary to momentarily become malicious and devise how these attacks may be conducted; “If you know the enemy and know yourself, you need not fear the result of a hundred battles”. Using previously designed attacks and our own research on the potential damage of DoW we can theorise the following would lead to an effective attack:

- Use Sybils to interact with a serverless application in order to trigger its functions.
- Do not aim to hit an application with DoS levels of requests i.e. HTTP Flooding.
- Use slow rate/long term attacks to naturally raise the invocation count of functions.
- Make traffic look as natural as possible, the aim is to fool an application owner into believing they have many users. Not only does this cause damage via DoW over serverless but the additional damage caused by falsely believing in a larger user base resulting in additional spending.
- Super-long term attacks yield high damage but are susceptible to discovery. Aim between one and six months for an attack. Discovery is more likely to occur when finances are being analyzed.

Mitigation against such attacks will prove difficult. We believe approaching it from the same mindset as bot detection rather than DoS will be appropriate. Some potential avenues for consideration are:

- Constructing common usage graphs to be used in traffic analysis for flagging suspicious users
- Strategic honey-pots that only malicious web-scrapers would find
- Applying reinforcement learning to graphs generated by user interaction to determine how suspicious they are

The value of the damage inflicted need not be astronomical to be effective. As mentioned in the motivations for DoW attacks (Section 5.1) the most likely targets are smaller entities. The results in Fig. 6 highlight that a modest bot-net of 1000 nodes could inflict up to $50,000 of costs in a month. This would cause significant finance issues to a small startup company.

10. Conclusions and future work

From this initial investigation on the emerging threat of DoW attacks, we have concluded that there is potential for significant financial damage should a malicious entity wish to exploit unique selling points of serverless computing. Future work on the mitigation of such attacks will need to tackle the following concerns:

1. Further research on fictitious request classification.
2. Securing serverless applications from our proposed threat of “leeches”.
3. Further investigations into unique weaknesses of serverless computing for unique attacks.
4. The detection of fake users and mitigation of their actions.

Meaningful data must be gathered in the absence of historical attacks, rather than wait for large scale attacks to occur. The barrier to obtaining such data is the creation of realistic test rigs that can be put under various attack simulations without incurring real world financial damage. Such a barrier is what makes research of this topic currently more difficult to test than traditional cybersecurity research topics, where a test bed can be created in isolation with relative ease. Our implementation of an isolated serverless platform to emulate the billing practices of the four major commercial platform is a substantial step towards furthering research on this topic.

[18] https://aws.amazon.com/lambda/pricing/
[19] https://cloud.google.com/functions/pricing/
[20] https://azure.microsoft.com/en-us/pricing/details/functions/
[21] https://cloud.ibm.com/functions/learn/pricing

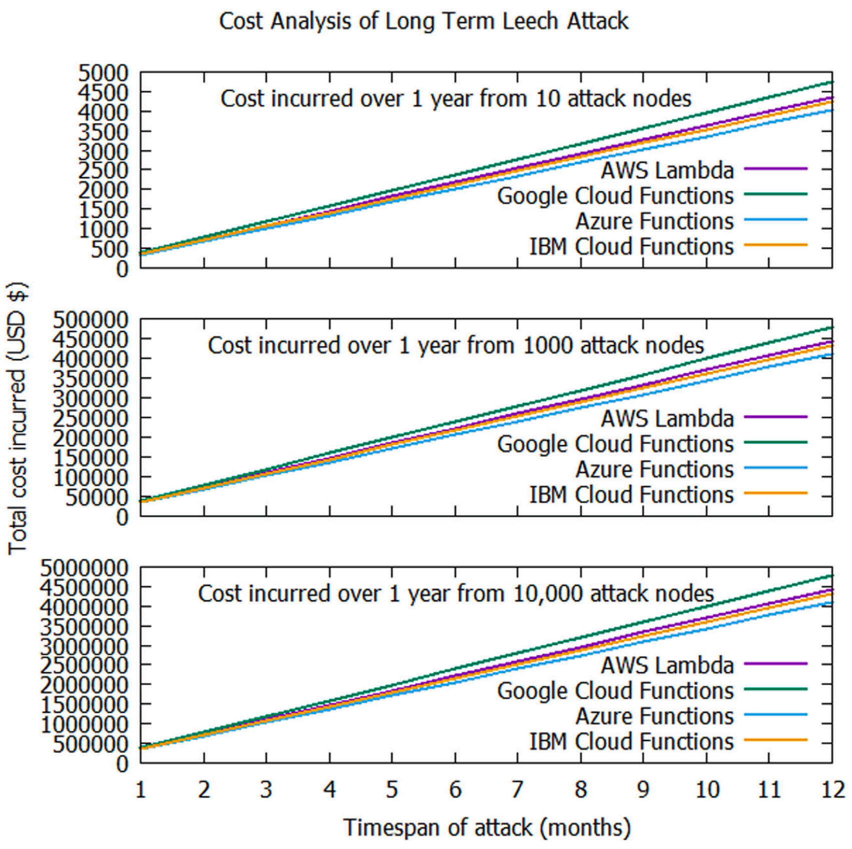

Fig. 6. Cost incurred when an increasing number of nodes send 2000 requests every hour.

We conclude that effective DoW attacks will certainly be of a distributed nature. A single source is not capable of delivering severe damage in realistic scenarios where proper security procedures have been followed. It is then in conjunction with existing and future Distributed Denial of Service research that Distributed Denial of Wallet will be investigated.

In this paper, we propose that DoW attacks are one of the latest threats in cybersecurity. With the ever increasing adoption of serverless computing, it is a threat that must be considered. Our work has provided a baseline for further research to be carried out in the field. The simulation experiments we performed highlight the financial damage that could occur as a result of DoW. Our simple attacks incurred theoretical damage of over one million dollars. While these numbers seem outlandish, it is an important baseline for worst case scenarios that can be built upon to prevent such attacks becoming a reality.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Dooley D. Serverless, shadow APIs and Denial of Wallet attacks. 2019, URL https://www.helpnetsecurity.com/2019/03/29/serverless-challenges/.

[2] Sachenko R. Severe truth about serverless security and ways to mitigate major risks. 2020, URL https://hackernoon.com/severe-truth-about-serverless-security-and-ways-to-mitigate-major-risks-cd3i3x6f.

[3] Rahic A. Fantastic serverless security risks, and where to find them. 2018, URL https://www.serverless.com/blog/fantastic-serverless-security-risks-and-where-to-find-them/.

[4] Pursec. The ten most critical security risks in serverless architecture. 2018, https://www.puresec.io/hubfs/SAS-Top10-2018/PureSec-SASTop10-2018.pdf.

[5] Datta P, Kumar P, Morris T, Grace M, Rahmati A, Bates A. Valve: Securing function workflows on serverless computing platforms. In: Proceedings of the web conference 2020. 2020, p. 939–50.

[6] Barna C, Shtern M, Smit M, Tzerpos V, Litoiu M. Model-based adaptive dos attack mitigation. In: 2012 7th international symposium on software engineering for adaptive and self-managing systems. IEEE; 2012, p. 119–28.

[7] Mukhopadhyay D, Oh B-J, Shim S-H, Kim Y-C. A study on recent approaches in handling ddos attacks. 2010, arXiv preprint arXiv:1012.2979.

[8] OWASP. OWASP top 10 (2017) interpretation for serverless, Tech. rep., URL https://owasp.org/www-pdf-archive/OWASP-Top-10-Serverless-Interpretation-en.pdf.

[9] Douligeris C, Mitrokotsa A. DDoS attacks and defense mechanisms: classification and state-of-the-art. Comput Netw 2004;44(5):643–66.

[10] OWASP. Regular expression Denial of Service - ReDoS. URL https://owasp.org/www-community/attacks/Regular{_}expression{_}Denial{_}of{_}Service{_}-{_}ReDoS.

[11] Morgan TD, Ibrahim OA. Xml schema, dtd, and entity attacks: A compendium of known techniques. 2014, Online, http://www.vsecurity.com/download/papers/XMLDTDEntityAttacks.pdf.

[12] AWS. Amazon API Gateway quotas and important notes. URL https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html.

[13] Zargar ST, Joshi J, Tipper D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Communications Surveys & Tutorials 2013;15(4):2046–69.

[14] Tripathi N, Hubballi N. Slow rate denial of service attacks against HTTP/2 and detection. Computers & Security 2018;72:255–72.

[15] Sauter M. “LOIC Will Tear Us Apart”: The impact of tool design and media portrayals in the success of activist DDOS attacks. Am Behav Sci 2013;57(7):983–1007. http://dx.doi.org/10.1177/0002764213479370.

[16] Varol O, Ferrara E, Davis CA, Menczer F, Flammini A. Online human-bot interactions: Detection, estimation, and characterization. In: Eleventh international AAAI conference on web and social media. 2017.

[17] Aslam S. Twitter by the numbers: Stats, demographics & fun facts. In: Omnicore. 2020, URL https://www.omnicoreagency.com/twitter-statistics/.

[18] Cao Q, Sirivianos M, Yang X, Pregueiro T. Aiding the detection of fake accounts in large scale social online services. In: Presented as part of the 9th {USENIX} symposium on networked systems design and implementation ({NSDI} 12). 2012, p. 197–210.

[19] Yang C, Harkreader R, Zhang J, Shin S, Gu G. Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter. In: Proceedings of the 21st international conference on world wide web. 2012, p. 71–80.

[20] Jia J, Wang B, Gong NZ. Random walk based fake account detection in online social networks. In: 2017 47th annual IEEE/IFIP international conference on dependable systems and networks. IEEE; 2017, p. 273–84.

[21] Alarifi A, Alsaleh M, Al-Salman A. Twitter turing test: Identifying social machines. Inform Sci 2016;372:332–46.

[22] Erşahin B, Aktaş Ö, Kılınç D, Akyol C. Twitter fake account detection. In: 2017 International Conference on Computer Science and Engineering (UBMK). IEEE; 2017, p. 388–92.

[23] Roman J, Radek B, Radek V, Libor S. Launching distributed denial of service attacks by network protocol exploitation. In: Proceedings of the 2nd international conference on applied informatics and computing theory. 2011, p. 210–6.

[24] Akiwatkar R. 10 AWS lambda use cases to start your serverless journey. URL https://www.simform.com/serverless-examples-aws-lambda-use-cases/.

[25] Kumar N. Resize an image in AWS S3 using a lambda function. 2019, URL https://levelup.gitconnected.com/resize-an-image-in-aws-s3-using-lambda-function-dc386afd4128.

[26] Lim J. Here's how much the iPhone's camera has changed over the years. In: Mashable South-East Asia. 2019, URL https://sea.mashable.com/tech/6504/heres-how-much-the-iphones-camera-has-changed-over-the-years.