



DoWTS – Denial-of-Wallet Test Simulator: Synthetic data generation for preemptive defence

Daniel Kelly¹ · Frank G Glavin¹ · Enda Barrett¹

Received: 2 May 2022 / Revised: 1 August 2022 / Accepted: 4 August 2022 /
Published online: 19 August 2022
© The Author(s) 2022

Abstract

The intentional targeting of components in a cloud based application, in order to artificially inflate usage bills, is an issue application owners have faced for many years. This has occurred under many guises, such as: Economic Denial of Sustainability (EDoS), *Click Fraud* and even secondary effects of Denial of Service (DoS) attacks. With the advent of commercial offerings of serverless computing circa 2015, a variant of the EDoS attack has emerged, termed, *Denial-of-Wallet* (DoW). We describe our development of a simulation tool as safe means to research these attacks as well as to generate datasets for the training of future mitigation systems to combat DoW. We believe that DoW may become increasingly prevalent as applications further utilise services based on a pay-per-invocation cost model. Given that the damage caused is purely financial, such attacks may not be disclosed as application users are not directly effected. As such, we believe that the development of an attack simulator and specific testing of security measures against this niche attack will be able to provide previously unavailable data and insights for the research community. We have developed a prototype DoW simulator that can emulate multiple months worth of API calls in a matter of hours for ease of training data generation. Our aspiration for the future of this work is to provide a system and starting point for research on this form of attack. We present our work on such a system Denial-of-Wallet Test Simulator (DoWTS) - a system that allows for safe testing of theorised DoW attacks against serverless applications via synthetic data generation. We also expand upon prior research on DoW and provide an analysis on the lack of specific safety measures for DoW.

Keywords Denial-of-wallet · Serverless computing · Function-as-a-service · Attack modelling · Attack simulation · Synthetic data generation · Synthetic network traffic

✉ Daniel Kelly
d.kelly69@nuigalway.ie

Frank G Glavin
frank.glavin@nuigalway.ie

Enda Barrett
enda.barrett@nuigalway.ie

¹ School of Computer Science, National University of Ireland, Galway, University Rd,
H91TK33 Galway City, Galway, Ireland

1 Introduction

Serverless computing has emerged as a powerful paradigm for application development. The appeal of decreased time to deployment, lack of servers to manage and the pay-per-use cost model of the functions that execute the business logic has accelerated its adoption by many application owners. These serverless function driven applications are highly scalable, such that a weak attempt at a flooding style DoS attack may be absorbed with no disruption to service. However, such a capability also leaves serverless functions vulnerable to an evolution of the EDoS attack. This evolution has been named *Denial-of-Wallet* Kelly et al. (2021). DoW can describe any abuse of a pay-per-use cloud product to cause an inflated bill. However, in the scope of this research, we will focus on the continual triggering of serverless functions in an attempt to induce greater operation costs for the application owner.

Previous work conducting the first academic analysis of DoW on serverless computing Kelly et al. (2021), attempted to give DoW a standard definition and presented a rudimentary analysis of potential attack vectors and worst case scenario damage figures. This work entirely focused on serverless DoW, however, DoW is a broad term and such an attack could apply to other systems where large bills may be incurred as a result of nefarious actions. The term DoW potentially came into being when it was used on Twitter in 2013 to describe excessive bills posted by a user for leaving a BizTalk VM running Ross (2013). The term has since become popular to describe any instance of runaway usage bills following some unexpected circumstance (e.g. infinite loops or VMs being left active when not in use). DoW as a targeted attack has been on the conscience of developers even before the advent of Function-as-a-Service (FaaS).

In order to gain insights into and tackle related cybersecurity threats that target public facing applications, the inspection of network traffic is required. However, such data can raise privacy concerns as network traffic logs may reveal personal information of users such as IP address, browsing data and in the cases of ecommerce data; purchase history. Therefore, it can be difficult to obtain realistic data for the training of mitigation systems. There are additional factors that increase the difficulty of obtaining the required data for training serverless application-based threat detection systems. In general, there is a lack of traffic data available that log interactions with function triggers that make up a serverless application. As the serverless paradigm undergoes continued adoption, this lack of data will no doubt be remedied. However, in this early stage of serverless specific security research it remains a large issue. An additional factor that makes DoW research specifically more difficult is that this attack targets the capital of application owners. As such, it is a costly and tedious ordeal to investigate DoW on *in-the-field* deployments of serverless applications due to the requirement to literally pay for any investigations into various attacks. The solution to these issues is to utilise synthetic data generators in order to create the required datasets for further use in mitigation system training.

In this paper, we present our work on a tool for generating synthetic usage data on a serverless application, called Denial-of-Wallet Test Simulator (DoWTS). This tool serves as a means of generating normal usage data in the absence of historical data. We detail extensively our heuristics for generating such data as well as perform an in depth evaluation of the validity of our synthetic “normal” usage generator. Such validity checks include: visual analytics, quantitative comparison and a number of statistical tests used for assessing the synthetic data similarity to real data. We surmise three basic attack vectors that will stand as an initial investigation into how an adversary may conduct DoW attacks. As this

topic is still in the early stages of research, we take the opportunity to further the general knowledge on DoW by conducting a deeper literature review on related attacks and mitigation approaches. Specifically, we look into the security of AWS based serverless applications. Following the safety standards outlined in a recent whitepaper on serverless development AWS (2019), we assess those safeguards with an investigation of known historical attacks such as EDoS, DoS, *Click Fraud* and API specific vulnerabilities.

This main contributions of this paper are as follows:

1. Building upon previous research on DoW Kelly et al. (2021), providing improved background research and definition for DoW as it pertains to serverless computing (Section 2).
2. Highlighting potential attack vectors and exposure on the lack of specific safety measures for DoW, reinforcing the need for future research on novel detection and mitigation methods (Section 3).
3. *DoWTS* - a synthetic data generator that allows for safe testing of theorised DoW attacks against serverless applications (Section 4).
4. An exhaustive evaluation of the validity of the data produced by DoWTS (Section 5).

2 Related work

2.1 Economic denial of sustainability detection and defence

On elastically balanced VMs, DoW or Economic Denial of Sustainability (EDoS), as termed in the article Hoff (2008), was theorised as a major threat in the result of mass scaling of the service. Research on EDoS has lead to the development of a number of detection and mitigation techniques for elastically load balanced cloud applications. This work is important as a starting point for DoW mitigation research.

Kumar et al. (2012) propose a model that operates in two modes depending on the load imposed on the server; *Normal Mode* and *Suspect Mode*. When the threshold is reached for resource consumption then the service provider activates *Suspect Mode* in anticipation of a Distributed Denial of Service (DDoS) attack. In this mode, the client machine must now solve a mathematical puzzle when a request is made.

Khor and Akihiro (2009) also propose a puzzle-based mitigation approach with self-verifying Proof of Work (sPoW). sPoW was devised in such a way that it could be implemented onto existing cloud technology. It returns a puzzle with an encryption key when a client tries to make a connection to a server. The client communicates with the server through plugins embedded in a static webpage. The server plugin handles the puzzle generation by creating a temporary encryption channel and sends back a puzzle for the client to solve. If successful, then the connection can proceed, otherwise it will return a more difficult puzzle in order to re-establish the connection.

Idziorek and Mark (2011) proposed two detection methodologies: Zipf's Law and Entropy detection. Zipf's Law is used to detect anomalous patterns in incoming traffic, whereby anomalous traffic will not fit the frequency distributions of *ranked data* being the expected traffic. Entropy based detection analyzes individual user traffic and calculates the entropy of session lengths over a specific time.

Sgalli et al. (2011) devised *EDoS Shield*. It consists of virtual firewalls (VFs) and verifier nodes (V-nodes). Client requests are sent to the VF which forwards the request to a

V-node. The V-node sends a graphical Turing test to the client, such as Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). If successful, the client's IP address is updated to the whitelist in the VF. Failing the test leads to the source IP address being added to the blacklist with subsequent packet requests being dropped.

2.2 Click fraud

Perhaps the most prevalent form of cyber attack that solely targets the finances of an application owner is *Click Fraud*. It is an issue that has plagued companies with an online advertising presence long before the introduction of FaaS or emergence of EDoS. *Click Fraud* is the act of illegally clicking on pay-per-click (PPC) ads to increase site revenue or to exhaust a company's advertising budget. *Click Fraud* is different from invalid clicks (those that are repeated or made by the ad's host/publisher) in that it is intentional, malicious, and has no potential for the ad to result in a sale. *Click Fraud* may fall into two categories Wilbur and Yi (2009):

Inflationary Click Fraud: Third parties may click on advertisements hosted on their service in order to inflate their revenues.

Competitive Click Fraud: Where rivals fraudulently click on their competitions advertisements in order to drive up their advertising costs thus exhausting their budget.

Click Fraud schemes are either executed by low-wage workers from developing countries or large botnets Kshetri (2010). These botnets execute ever increasingly sophisticated algorithms that mimic the behaviour of legitimate users. In 2016-2017, it was found that over 76% of fraud detections were from machines that hosted both a bot and a human Ana (2017). This makes the task of detecting such illegitimate traffic much more difficult. Such methods could be utilised in causing DoW and, as such, it is important to approach DoW detection not just from the point of view of DoS style flooding attacks but also more covert attack mechanisms that go unnoticed as with *Click Fraud*. Such detection approaches for *Click Fraud* may involve activity monitoring such as mouse movement Firebrand (2017) or defining geographical areas of suspicion for incoming traffic Kshetri (2010). *Click Fraud* was estimated to cost \$6.5 billion dollars globally despite 80% of brands (participating in the study), deploying some form of fraud countermeasure Firebrand (2017).

2.3 API targeted attacks

Serverless functions can be invoked by various triggers. By far the most common method is via Application Programming Interface (API) endpoints. Some platforms such as Google Cloud will create a URL endpoint for a function once it is deployed, whereas others such as AWS require you to link functions to an API Gateway. The result is a means to execute functions via a request to the endpoint. This intrinsic link with serverless functions makes API the largest attack surface in which DoW could manifest.

There are unique vulnerabilities that may target APIs, as outlined in OWASP's top ten risks to APIs OWASP (2019). These vulnerabilities serve as an entry point for an attacker to cause DoW. Specifically in the cases of *Lack of Rate Limiting* and *Insufficient Logging and Monitoring*, an attacker could flood API endpoints with requests quickly driving up costs. However, all these vulnerabilities may be exploited with the goal of causing DoW.

Efforts to combat API vulnerabilities are largely based around training developers to identify the issue and implement a fix. To this end, a number of training schemes that

gamify vulnerability exploration by rewarding users for correct exploitation of known vulnerabilities on purpose-built insecure applications have been developed (Su et al., 2019; Maki et al., 2020; Idris et al., 2021).

2.4 Training of mitigation systems using machine learning

As cyber attacks become ever more complex and powerful, the need for equally effective mitigation systems increases. To this end, the use of machine learning (ML) algorithms trained on datasets of previous attacks for classification of suspicious traffic has become an important step forward in cybersecurity.

Niyaz et al. (2016) use deep learning for feature reduction of a large set of features derived from network traffic headers for detection of DDoS. They utilise a Stacked Auto-Encoder (SAE) that consists of stacked sparse auto-encoders and a softmax classifier for unsupervised feature learning and classification respectively. A sparse auto-encoder is a neural network with three layers; input and output containing M nodes and a hidden layer containing N nodes. M nodes represent a record with M features. For training, output is the identity of function of the input. The sparse auto-encoder networks calculates optimal weights of matrices and bias vectors while trying to learn an approximation of an identity function using back propagation. Multiple sparse auto-encoders are stacked so the output of one feeds into input of the next while also reducing in dimension. Finally, the last hidden layer is fed into the softmax classifier. Normal traffic data was collected by recording use on a home network over 72 hours. Attack data was generated using *hping3* in an isolated setup. *Tcpreplay* was used on a software defined network environment for training and testing by replaying the recorded normal traffic and generated attack traffic. DDoS attacks were detected with a 95.65% accuracy using this system.

He et al. (2017) compare multiple supervised and unsupervised ML algorithms to classify outgoing traffic from a cloud platform as DDoS. They train the algorithms to detect the attacks from the source rather than on the victim end. The attacks they chose for detection were: Secure Shell (SSH) Brute Force, Domain Name System (DNS) Reflection, Internet Control Message Protocol (ICMP) Flood and Transmission Control Protocol (TCP) Synchronise (SYN) Flood. They trained the following algorithms:

- **Supervised** - Linear Regression, Support vector machines w/ Linear, polynomial and radial basis function kernels, Decision Tree, Naïve Bayes and Random Forest
- **Unsupervised** - K-Means and Gaussian Mixture Model for Expectation Maximisation

They ran these trained algorithms on a test cloud running OpenStack for virtual machine provision and found that Random Forest had the best accuracy at 94.96%.

Priya et al. (2020) also found that Random Forest had the best accuracy, training the algorithm on existing datasets.

Ko et al. (2020) utilised netflow data on an Internet Service Provider (ISP) and BoNeSi Markus-Go (2008) for generating attacks. A dynamic feature selector was devised for use with Self-Organising Maps in training an attack detection system.

Training these mitigation systems requires reliable data. Once such method of acquiring said data is the generation of synthetic network traffic. Including this paper, there are a number of recent approaches to this topic.

Cordero et al. (2021) create a tool to allow for standardised replication of network intrusion detection research by generating synthetic attack traffic by replicating background

traffic properties. They demonstrate the ability to inject attack traffic using their tool rather than requiring the release of specific datasets for that attack.

Xu et al. (2021) create a tool for generating network traffic. They run an exhaustive suite of tests for validating the data which includes likelihood testing; negative log likelihood (NLL) and performance in machine learning tasks.

In summary, this related work serves as a useful starting point for DoW research. DoW is a variant of EDoS, in which the damage caused is financial as a result of unwanted operations on an application. DoW and *Click Fraud* exhibit similar characteristics, such as the ability to cause damage per click/invoke and that it is not necessary to perform flooding style attacks to have an effect. API endpoints are the largest attack surface for DoW due to REpresentational State Transfer (REST) APIs being the most common trigger for serverless functions. As such, understanding the security concerns of APIs is important. Finally, as this paper proposes the use of synthetic data for ML classifier training, it is beneficial to look towards existing research on the subject.

3 Denial-of-wallet on serverless functions

As outlined in Section 2, the purposeful targeting of an individual's service with the aim of causing undue usage bills has existed under various guises in the past. In this paper, we approach the topic of DoW from the perspective of specifically targeting serverless functions. To date, this perspective has received very little attention and, as such, ground should be broken on this latest avenue for attackers to exploit.

"Serverless DoW attacks are defined as the intentional mass, and continual, invocation of serverless functions, resulting in financial exhaustion of the victim in the form of inflated usage bills" Kelly et al. (2021). Such attacks seek to exploit FaaS platforms massive capability for scaling in response to increased traffic loads or take advantage of the cost per invocation pricing model by continually hitting an endpoint that triggers functions. Both scenarios are also susceptible to exploitation of any input parameters a function may require, leading to the increased runtime of the function.

3.1 Possible attack methods

The leech attack Kelly et al. (2021), is a theorised slow rate DoW attack where a malicious program repeatedly sends requests to an API endpoint or performs some action that would invoke a serverless function. Leeches may run for long time spans over a month and at a rate that would not be detected by reasonable Web Application Firewall (WAF) rate limiting rules. The leeches will then appear as regular users. If there are enough leeches then the damage increases. Unless you have some reason to believe this is illegitimate traffic then the traffic will be allowed to continue and the bill will be paid.

The pay-as-you-go pricing model that commercial serverless platforms operate is the vulnerability that allows for such an attack. If the load of a DoS attack was distributed over a month, the attack would be ineffective as the load on the system would never become high enough to bring it down. However, with FaaS, the bill continually rises. This attack plays on the same vulnerabilities as *Click Fraud* does in that it is up to the accounts manager to spot the anomaly in the billing, which as shown through the prevalence of *Click Fraud*, happens at a large scale.

In DoS mitigation systems that use an adaptive rule set i.e. can change the firewall rules based on fluctuation in *deemed to be real* traffic Barna et al. (2012), a leech attack can again utilise the ability that it can be executed over a long time span to take advantage of such a system. Leeches can increase in number and intensity over their long run time to reduce suspicion and establish new false normative baselines. Boiling the frog is an apt analogy for this form of attack, where you slowly increase the heat such that the victim does not notice the danger until it's too late.

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 vectors such as function input parameter exploitation. For example, an attacker could trigger a function that uploads to a storage service with images of the largest size that incurs the greatest runtime possible out of the function execution; that is, keep increasing the size of the image Kelly et al. (2021) until you reach some function timeout then use that as the basis of your attack going forward.

3.2 Motivation for DoW

Attacks like DoW and previous examples of EDoS differ from destructive attacks like DoS in that resulting damage has a large range of values in which it can be effective. A successful DoS attack will put a service down and be immediately noticed by the victims. Whereas with DoW, the financial damages may accrue over a long period of time depending on how perceptive the application owner is. To this end, DoW may effect smaller companies and solo application developers more due to limited funding. The ability to potentially remain undetected for longer is also a strong advantage to using such an attack.

In a recent report conducted by Trend Micro Fuentes (2020), the price of botnet procurement has dramatically decreased since 2015. Generic DDoS botnets can be acquired from \$50 a day and generic botnets can be rented from \$5 a day. It is therefore highly feasible that despite the initial investment, an attack could net more damage.

DoW is an attack that may be chosen for its covertness and suitability for sabotage. Rather than an attack seeking to *take out* a victim, they can potentially weaken them. This could be a tactic employed where a larger entity is interested in stifling the growth of a smaller one. It may also be used to cause customers to distrust the application if it is regularly attacked.

Another possible motivation would be as a form of ransom attack. Attackers may cause a DoW then contact the victim looking to extort some payout with threat of continuing to cause DoW or even contact in advance with threats of a DoW in order to receive a payout.

3.3 Current safeguards in big cloud providers

Securing a serverless function from threats not limited to DoW falls in the hands of the application developer. AWS highlight a *shared responsibility model*, where the developer is responsible for security of the code and explicitly states that poorly configured code resulting in adverse repercussions may not receive sympathy when reported AWS (2019). Steps to protect applications are highlighted in their whitepapers for developing well architected frameworks and the security overview of AWS Lambda AWS (2021, 2019). The relevant *pillars of a well architected framework* that should be implemented to protect a serverless application from DoW (specifically for AWS based applications) include:

1. *Operational Excellence Pillar* – Developers should set individual alarms for separate Lambda functions so that you can catch invocations running for longer than usual. AWS Xray tracing should also be utilised for monitoring of interactions with the service.
2. *Security Pillar* – Applications should undergo risk assessment and have appropriate mitigation strategies. Identity and access management is an important factor when implementing secure applications. As serverless functions are often invoked via API triggers, the following mechanisms are recommended:
 - *AWS IAM authorisation* – Only allow users within AWS environments access with IAM credentials (protect against internal attack by a compromised user). Always give least amount of access
 - *Cognito user Pools* – Built in user management or integration with socials login. Uses OAuth scopes.
 - *API Gateway Lambda authoriser* – Use a lambda function to interact with some Identity Provider
 - *Resource policies* – Resource policy can block/allow specific AWS accounts, IP ranges, Classless Inter-Domain Routing (CIDR) Blocks, virtual private clouds etc via JavaScript Object Notation (JSON) policy file
3. *Cost Optimisation Pillar* – Application owners should appropriately monitor costs to further refine return on investment. With many more functions and API endpoints, it is recommended to allocate costs from bills to individual resources to gain a more granular view. This allows for quick identification of services generating elevated costs.

While these safeguards are described specifically to AWS's offerings, the same principles apply to all platforms.

On the development side the following considerations should be taken at certain levels:

- *Identity and Access* - Implement some form of authentication to hide function interaction endpoints from the public facing web.
- *Code* - Use of secrets to maintain privacy of credentials and business critical data. Input validation to minimise risks associated with API requests. Vigilance in monitoring and dependency vulnerabilities.
- *Data* - Encryption in Transit, such as Secure Sockets Layer (SSL).
- *Infrastructure* - Configuration of usage plans to help against effects of DoW. Implementation of a WAF for protection against other forms of cyber attack.
- *Logging and Monitoring* - Appropriate usage of both to detect suspicious usage on application.

The listed safeguards serve as a possible first defence against the possibility of being affected by a DoW attack. However, these safeguards are not apparent for *new-to-serverless* developers. Official sample applications that such a developer would follow when learning how to create serverless applications AWS (2017); Beswick (2020a) are susceptible to API vulnerability attacks such as capturing an endpoint URL (Fig. 1) and replaying the request with varying header and body fields to force repeat invocations. Serverless DoW relies on repeatedly triggering functions, the most common method being via API endpoints. Requests can be spammed using trivial programmes that quickly loop REST commands. It is not required that an attacker use a vast number of





200	OPTIONS	 cwhtund5n9k0.execute-api.us-east-1.amazonaws.com
200	POST	 cwhtund5n9k0.execute-api.us-east-1.amazonaws.com
200	OPTIONS	 cwhtund5n9k0.execute-api.us-east-1.amazonaws.com
200	GET	 cwhtund5n9k0.execute-api.us-east-1.amazonaws.com

Fig. 1 API endpoints that trigger functions can be sniffed on the network by searching for URLs that match the platform's template (unique ID blurred for privacy)

nodes to cause damage, like in DDoS. The attack can cause financial damage with only a modest sized attack source. However, it is worth noting that in the event of such an attack, if the application owner becomes aware and decides to shut down an endpoint or even the whole application, this is essentially a successful DoS with a fraction of the required firepower. These unique aspects that define DoW on serverless computing are what lead us to create the DoWTS system to aid in future investigation on this potential attack.

4 DoWTS - Denial-of-wallet test simulator

Denial-of-Wallet Test Simulator (DoWTS), is a simulated serverless platform that will emulate the cost damage of DoW and create pseudo timestamped datasets of request traffic Kelly (2022). DoWTS generates synthetic baseline request traffic via a system of heuristics modelled on existing datasets of requests on ecommerce websites Kechinov (2020, 2021). This serves as background usage upon which any number of theorised DoW attacks can be launched. DoWTS calculates the current cost of serverless function invocations per simulation time step for four of the largest commercial platforms. It also generates usage log data of every function invocation with a label denoting whether it was a bot or legitimate traffic. Such labelled data will be used in future research on classifying legitimate traffic.

DoWTS is written in Golang in order to take advantage of the multi-threading capabilities of *goroutines*. It is composed of three main components; Serverless Platform Emulator (SPE), Usage Generator and a MongoDB database (Fig. 2).

DoWTS is operated via interaction with the usage generator as per the sequence diagram in Fig. 3. The user specifies the combination of attack method and user identification spoofing as per the attack traffic generator. The scenario is programmed with predefined values for the parameters listed in the synthetic traffic generator. Finally, the scenario is executed.

DoWTS creates log files for each of the simulated platform's cost, compute time and number of function invocations per run. This is to accompany the generated dataset of traffic to the serverless functions.

4.1 Serverless platform emulator

We developed the Serverless Platform Emulator in order to compare the effects of Denial-of-Wallet across multiple commercial platforms. It includes go-packages each for AWS Lambda, Google Cloud Functions, Azure Functions and IBM functions. Utilising the openly available pricing formulae for each platform, the emulator keeps count of: total

Denial of Wallet Test Simulator (DoWTS)

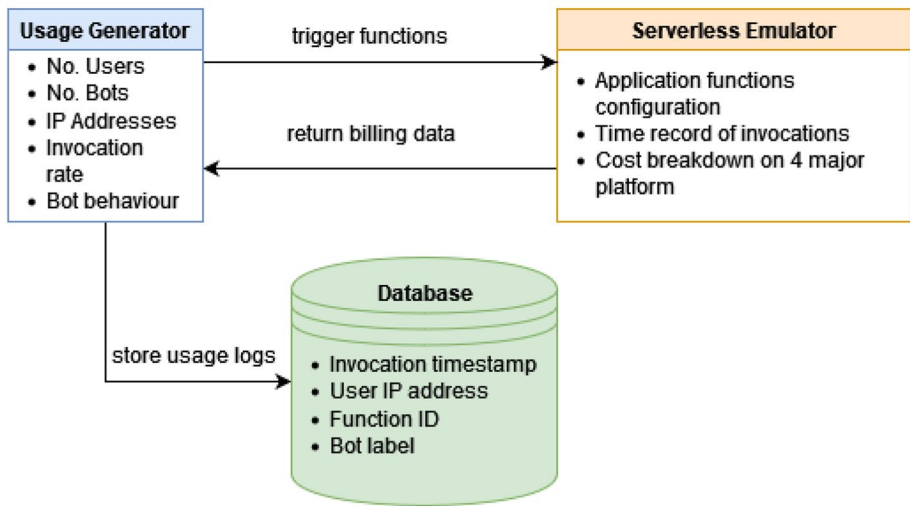


Fig. 2 DoWTS usage flow

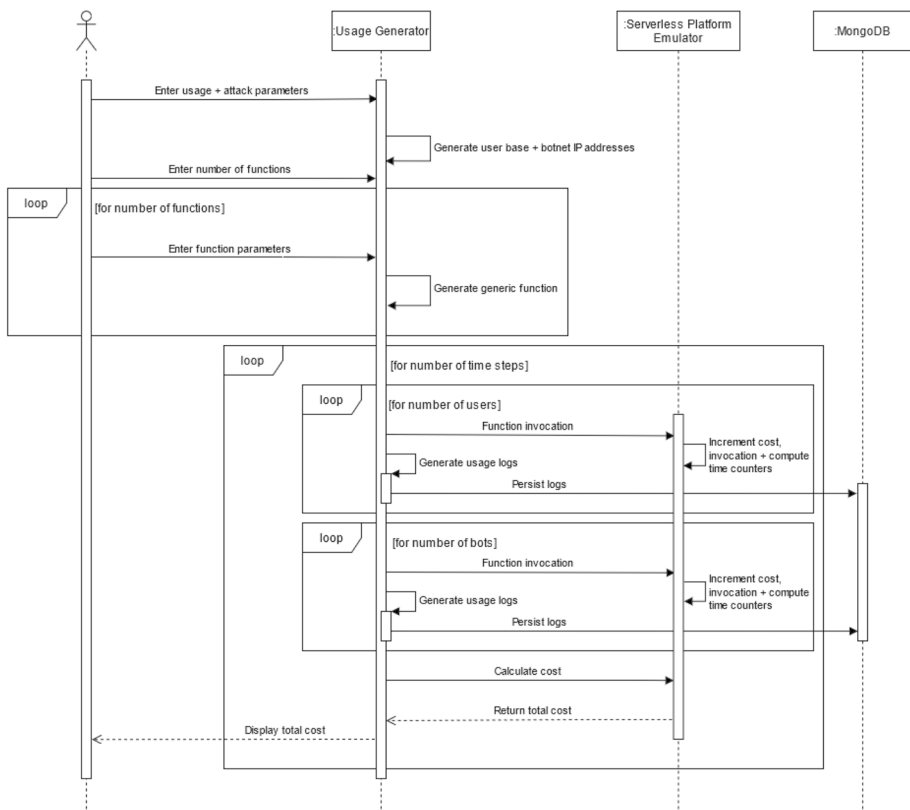


Fig. 3 Sequence diagram of DoW attack simulator and dataset generator

cost, total invocations and total runtime of the functions on that platform. DoWTS can be used to model the expected expenditure on the serverless components of an application across the four largest platforms. Such information is useful in the decision making process for which platform to choose as the host of the serverless functions. However, while our simulator does indeed offer this utility, its primary purpose is the generation of datasets for DoW detection research and as a means to safely study a variety of attack vectors. DoWTS can be configured with specific instances of serverless applications by varying the input parameters, such as usage metrics and function configuration.

4.2 Usage generator

The usage generator comprises an attack traffic generator, synthetic traffic generator and dataset based traffic generator. The role of the usage generator is to manage the timings of function invocations to the SPE, log the function invocations and persist them to the database. All three generators have a standardised output format for each pseudo requests comprising of: *timestamp*, *IP address*, *function ID* and a label denoting whether the request came from the attack generator or not.

4.2.1 Dataset based traffic generator

There is a lack of historical data of both attacks and general usage on serverless applications available to the public. However, it is feasible to utilise certain datasets that capture usage on traditional web applications where the endpoints of an interaction are recorded. For example, the following datasets Kechinov (2020, 2021) are request event histories recorded over five month periods each. The data was collected via Open CDP REES46 Technologies (2022) on a fashion and an electronics ecommerce store respectively. These datasets contain a field *event_type* that corresponds to what URL endpoint a user interacted with. If this is to be translated to a similar mechanism on serverless applications, these can be interpreted as API triggers to serverless functions. As such, the recorded traffic, in these datasets, can be streamed via the dataset based traffic generator as the legitimate traffic in the DoWTS system. The result of translating this traffic to the DoWTS system allows for the fine-tuning of our synthetic data generator.

4.2.2 Synthetic traffic generator

The synthetic traffic generator takes in a number of parameters from the user, in a given usage scenario, that define:

- **User base size** - The total number of users that access an application
- **Botnet size** - The number of bots attacking the application
- **Time step** - The granularity of time for the simulation run (seconds, minutes, hours, days, weeks, months). This will affect the data logs generated
- **Number of time steps** - The length of the simulation
- **Users per time step** - The number of users accessing the application in a given time step
- **Requests per time step** - The number of function requests in a given time step

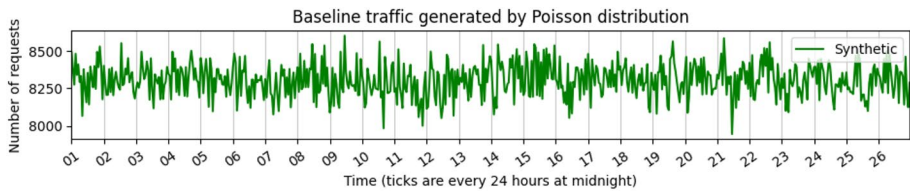


Fig. 4 Baseline traffic generated by Poisson distribution

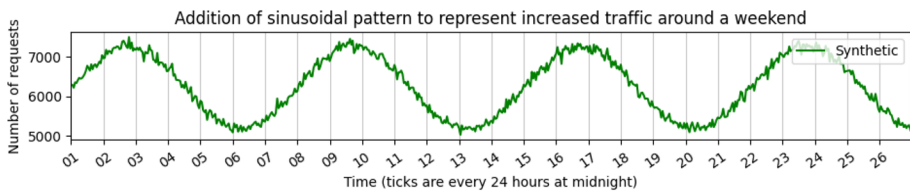


Fig. 5 Addition of sinusoidal pattern to represent increased traffic around a weekend

Once these parameters have been entered, DoWTS generates IP addresses for the users and botnet. The user is then prompted to enter in the configuration of however many functions are present in their application i.e. memory allocation and function execution time. Functions may then be grouped into chains of functions that logically execute one after another. These chains contain a parameter that allows for the distribution of traffic to certain chains more than others for instances where there are functions that are more commonly used than others.

Traffic Synthesis The timing of requests is dictated by a system of heuristics devised in reference to the output of the ecommerce datasets Kechinov (2020, 2021). The hourly count of requests is used as the metric in which we base these heuristics. Baseline traffic is generated by a random number generator utilising a Poisson distribution, where the *users per time step* value is used as lambda in equation (1). The resulting traffic is noisy and random about that value (Fig. 4).

$$P(x) = \frac{e^{-\lambda} \lambda^x}{x!} \quad (1)$$

Through visual analysis of the ecommerce datasets' number of requests per hour over the course of a month long period, we observed a number of patterns in the data. Firstly, there appeared to be fluctuations in traffic amounts roughly every seven days. We can infer this corresponds with increased traffic over the weekends. In order to achieve this fluctuation, a sinusoidal pattern with a period of seven days was added to the Poisson distribution traffic (Fig. 5). This subtly causes the traffic to oscillate.

The next pattern we observed, was obvious dips in traffic during late night hours. Again a sinusoidal pattern was added to the traffic timings in order to achieve stark fluctuations in day/night traffic (Fig. 6).

Day time traffic did not follow the regular repeating values of a sine wave in the ecommerce data. We observed clipping of the peaks and the traffic at that clipping

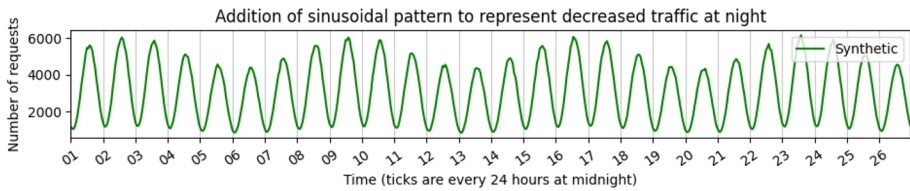


Fig. 6 Addition of sinusoidal pattern to represent decreased traffic at night

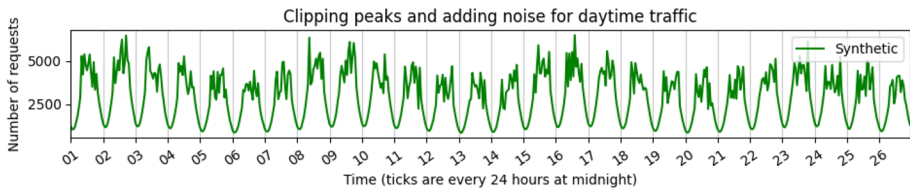


Fig. 7 Clipping peaks and adding noise for daytime traffic

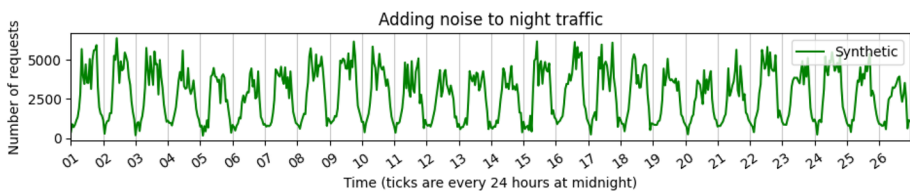


Fig. 8 Adding noise to night traffic

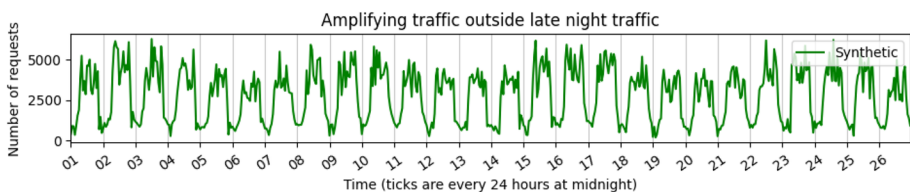


Fig. 9 Amplifying traffic outside late night traffic

threshold was noisy. We synthesise this behaviour by setting the traffic to another randomly generated number based on a Poisson distribution when the ratio of *pre-augmented* traffic to *augmented* traffic (that is traffic before and after the day/night sine augmentation) is less than 2 (Fig. 7).

Similarly, in order to reduce traffic that followed a perfect sine wave, minor noise was introduced to the troughs of the wave (Fig. 8).

The final observation of our graphical analysis of the ecommerce datasets, was that there was less downtime than up time in traffic i.e. the peaks are wider than the troughs.

By amplifying the traffic between the zones of clipping the peaks and adding noise to the troughs, the effect can be achieved (Fig. 9).

Finally, a smoothing algorithm was applied that interpolates the distance between subsequent values should the difference between them be too much. This is required to avoid overly jagged traffic (Fig. 10).

4.2.3 Attack traffic generator

There are no documented instances of targeted DoW on serverless to date. Therefore, it is required that our research take on the role of devising potential attack patterns that may cause DoW in an effort of preemptive defence. This paper primarily focuses on presenting DoWTS as a system for future attack research. As such, the attack generator posed here is to serve as a proof-of-concept. We surmise three rudimentary “leech” attacks Kelly et al. (2021) in order to demonstrate DoWTS’ capability at generating attack data. We have modelled three potential attack vectors for a simple long term leech:

1. Constant rate – eg. 2000 requests per hour (rph) per bot
2. Exponential rate – start low (eg. 10 rph) then increase by factor (eg. 1.001) of rph
3. Random rate – attack with randomly varying numbers of requests

On top of the three attack vectors, DoWTS also implements a number of methods of recycling and changing the IP addresses of the botnet to either emulate a changing botnet or spoofing IP address. These methods are:

1. No IP address change
2. Change bot IP address every x number of requests
3. Change bot IP address every x amount of time steps
4. Pool bot IP addresses for reuse over time

These serve as a starting point from which we begin the challenge of training classifiers to detect varying attack strategies for DoW. Initial attack vectors will be simplistic but over time we will investigate increasingly difficult patterns to detect, utilising DoWTS to generate data.

4.3 Execution example

The configuration parameters for DoWTS can be arbitrarily chosen or selected in order to replicate a known serverless application. In this example, we choose the usage

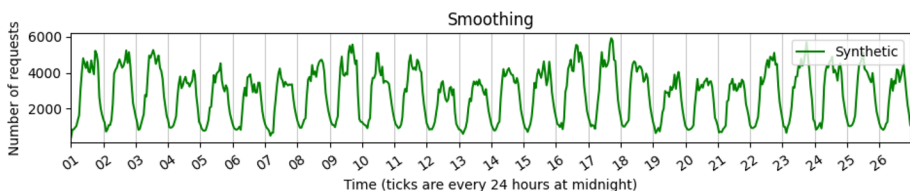


Fig. 10 Smoothing traffic

values described in an AWS serverless application load testing guide Beswick (2020b). It describes a geo-location question and answer based application along with its expected traffic load. From this we can set the following parameters for normal operation:

- **User base size** - 1,000,000
- **Time step** - 1 hour
- **Number of time steps** - 730 (1 month)
- **Users per time step** - 1,500
- **Requests per time step** - 61,000

The *time step* and *requests per time step* are the most pertinent values in this example as they set the expected load on the application. The usecase also provides information on how functions should be chained (Fig. 11). These are the four API requests a user can make when interacting with the application (GET/POST Questions and Answers). Each API request triggers a chain of Lambda functions. For the purpose of DoWTS, we are only interested in the number of Lambda functions and in what order they execute. Understanding of the mechanism of the example application is not required for generating traffic data with DoWTS.

From this we can infer the following function chains:

- **POST Question:** *PostQuestions* → *ProcessQuestion* → *Publish*
- **POST Answer:** *PostAnswers* → *ProcessStarAnswers* \cup *ProcessGeoAnswers* → *Aggregation* → *Publish*
- **GET Answer:** *GetAnswers* → *Aggregation* → *Publish*
- **GET Question:** *GetQuestions* → *Publish*

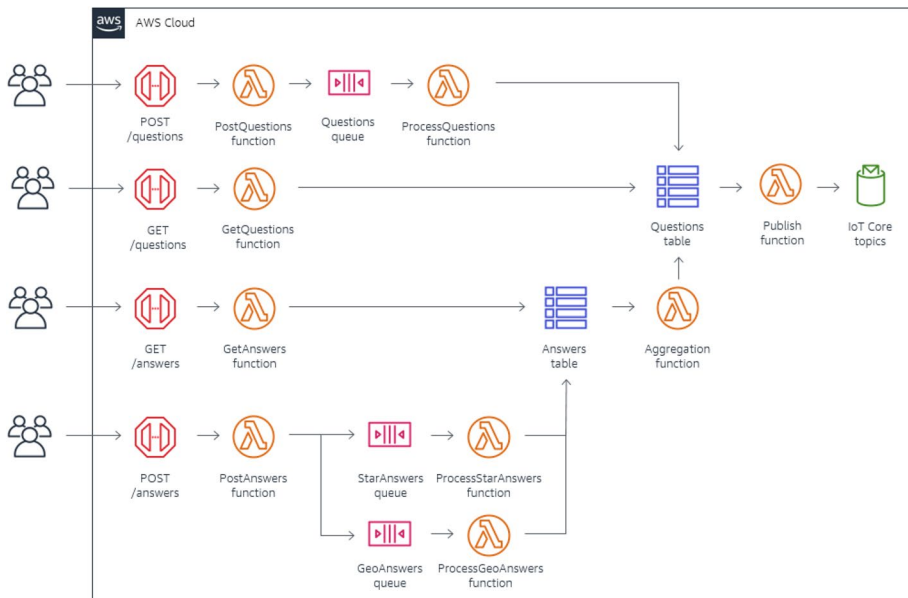


Fig. 11 Architecture diagram of application in AWS load testing example Beswick (2020b)

Also we know the expected load on each endpoint, so we can set the factor on each function chain that determines the ratio of the total traffic it should receive.

Running DoWTS on this configuration gives us a result for normal operation usage in this scenario. We can visualise the traffic through a count of requests per hour (Fig. 12).

We will use this configuration as the background traffic upon which we can perform three additional runs with each of the attack patterns activated. The first run will be a continuous rate leech attack of 100 bots performing 200 requests each throughout each hour. This is a very low rate that would not trigger any sensible rate limiting rules on a WAF. The second run will be an exponential rate leech attack of 100 bots that will start at 10 requests each and increase by a factor of 1.005 each time step. The goal is to lull the application owner into thinking they are receiving a steady increase in legitimate traffic. Finally, a random rate leech attack of 100 bots will be performed by randomly selecting a number of requests to send from 0 to 400. Figure 13 visualises the affect on traffic these attacks have.

DoWTS also calculates the cost of these function invocations. Table 1 shows how the attacks drive up operational costs of the application via the end of the month bill. Even at low scale we can see that leech attacks may double the costs.

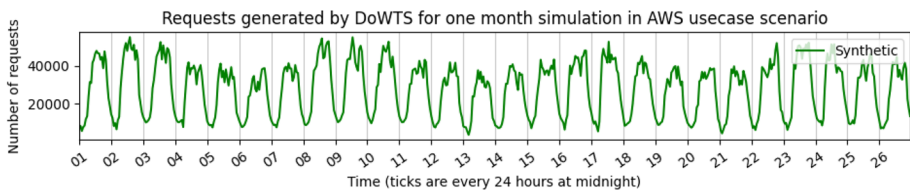


Fig. 12 Normal traffic on usecase application

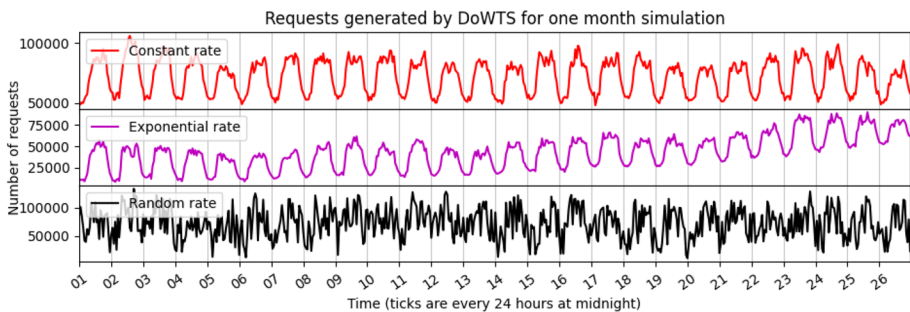


Fig. 13 Three leech attack variants on usecase application

Table 1 Operational cost of application in each scenario for one month of each serverless platform

	AWS	Google	Azure	IBM
No attack	\$20.69	\$26.62	\$20.02	\$17.15
Constant rate	\$77.40	\$85.92	\$74.71	\$68.44
Exponential rate	\$49.49	\$56.65	\$47.78	\$43.25
Random rate	\$79.18	\$87.74	\$76.42	\$70.07

5 Synthetic normal data evaluation

A major challenge with regard to synthetic data generation is ensuring that the generated data closely correlates to real world data as much as possible. A simulation platform is only useful if it can emulate the system it models and give confidence that any classifiers trained using the simulation data will be robust when deployed in the real world. In order to assess the usability of DoWTS in future work on training classification algorithms, we must evaluate the validity of the data produced by our synthetic usage generator. We utilise the SynthGauge Data Science Campus (2022) framework for performing a statistical analysis and also conducted our own visual analysis of the data.

SynthGauge is a Python library that provides a framework for evaluating the utility and privacy of synthetic datasets, using a range of metrics and visualisations. It is the result of a collaboration between the UK Office for National Statistics (ONS) and The Alan Turing Institute. Our evaluation focuses on the timings of arrivals of requests, from which we bin traffic to achieve an hourly count of requests. Upon this data, we compute the following to assess the validity of our synthetic data:

- **Quantitative Comparison** - calculation of useful statistical values such as: Mean, Standard Deviation, Min/Max, Inter-quartile Range and Median. These values are useful for initial comparison of real and synthetic datasets in order to determine they are similar in scale (number of data entries).
- **Kolmogorov Smirnov (KS) Test Statistic** Scipy (2022a) - calculates the maximum difference between the cumulative distribution functions of the traffic count in the real and synthetic datasets. If the returned statistic is small, then it can be said that the datasets come from the same distribution.
- **Wasserstein Distance** Scipy (2022b) - or Earth Mover's distance, can be thought of as calculating the amount of work required to move from the distribution of the synthetic data to the distribution of the real data. The distance is zero if the distributions are identical, and increases as they become less alike. We compare the Wasserstein distances between multiple months of real data and compare the distance between real and synthetic data with that.
- **Jensen Shannon Divergence** Scipy (2022c) - describes the difference between the real and synthetic distributions of the traffic count in terms of entropy. We can think of the Jensen Shannon divergence as the amount of information, or entropy, encoded in the difference between the real and synthetic distributions of the traffic count. The distance is zero if the distributions are identical, and is bounded above by one if they are nothing alike.

For consistency in comparison, the range of data from each month is reduced to 27 days in order to accommodate the lack of data from the month of February. Henceforth, the fashion and electronics ecommerce datasets shall be referred to as *dataset 1* and *dataset 2* respectively.

5.1 Quantitative comparison

Our first measure of validation is a comparison of quantitative metrics for each dataset. We compute the mean, standard deviation, minimum value, maximum value, inter-quartile

Table 2 Quantitative comparison of dataset 1 on number of requests per hour

Month	Mean	StD	Min	25%	50%	75%	Max
October	5544.091346	2824.513974	673.0	3010.25	6081.0	7492.25	15201.0
November	6245.689103	3601.325623	694.0	2970.25	6626.5	8212.75	23257.0
December	5131.977564	2555.463735	675.0	2702.75	5649.0	7126.25	10912.0
January	5533.810897	2858.003096	415.0	2637.25	6054.0	7902.50	11473.0
February	5986.722756	3011.134544	94.0	3136.25	6672.0	8429.75	14427.0

Table 3 Quantitative comparison of synthetic data based on dataset 1 on number of requests per hour

	Mean	StD	Min	25%	50%	75%	Max
Synthetic	5415.012821	2685.997338	771.0	2578.00	5937.5	7633.50	10750.0

Table 4 Quantitative comparison of dataset 2 on number of requests per hour

Month	Mean	StD	Min	25%	50%	75%	Max
October	210.346154	107.983838	21.0	104.5	225.0	296.00	450.0
November	270.397436	134.136591	0.0	133.0	299.0	380.25	548.0
December	208.987179	101.698607	8.0	110.0	236.5	289.00	428.0
January	253.105769	122.933362	23.0	130.5	280.5	354.25	545.0
February	251.246795	111.162065	30.0	136.0	284.5	339.00	460.0

Table 5 Quantitative comparison of synthetic data based on dataset 2 on number of requests per hour

Month	Mean	StD	Min	25%	50%	75%	Max
Synthetic	235.451923	118.611568	33.0	111.0	260.5	331.0	494.0

range and median of the number of requests per hour. By computing these values on each of the five months of real data in both real datasets we can determine a range of values that should constitute acceptable readings for our synthetic data (Table 2 and Table 4). This will suggest that the combination of input parameters to DoWTS, in conjunction with the synthetic usage generator heuristics, output traffic inline with expected values based on historic readings. Table 3 and Table 5 confirm that the synthetic data falls within an acceptable range of the real data it is based off.

5.2 Statistical testing

Our next measure of validity is a number of statistical tests chosen to determine the similarity between the real and synthetic datasets. Our quantitative comparison confirmed that our datasets contain similar values. As such we can apply the three aforementioned tests

Table 6 Statistical evaluation of dataset 1 on number of requests per hour against itself to establish baseline values

	KS Test statistic	p-value	Wasserstein Distance	Jensen Shannon Divergence
Oct - Nov	0.1202	0.0002	730.4824	0.0264
Nov - Dec	0.1875	< 0.0001	1113.7660	0.0546
Dec - Jan	0.1218	0.0002	427.7308	0.0218
Jan - Feb	0.0946	0.0075	464.5721	0.0091

Table 7 Statistical evaluation of synthetic data against dataset 1 on number of requests per hour

	KS Test statistic	p-value	Wasserstein Distance	Jensen Shannon Divergence
Oct - Synth	0.1073	0.0014	393.5753	0.0467
Nov - Synth	0.1153	0.0004	948.3461	0.0517
Dec - Synth	0.1137	0.0006	435.7339	0.0451
Jan - Synth	0.0961	0.0062	271.7403	0.0280
Feb - Synth	0.125	0.0001	710.1907	0.0290

Table 8 Statistical evaluation of dataset 2 on number of requests per hour against itself to establish baseline values

	KS Test statistic	p-value	Wasserstein Distance	Jensen Shannon Divergence
Oct - Nov	0.2932	< 0.0001	60.1891	0.0757
Nov - Dec	0.3509	< 0.0001	61.4647	0.1123
Dec - Jan	0.2644	< 0.0001	44.1185	0.0799
Jan - Feb	0.0785	0.0426	13.0929	0.0155

Table 9 Statistical evaluation of synthetic data against dataset 2 on number of requests per hour

	KS Test statistic	p-value	Wasserstein Distance	Jensen Shannon Divergence
Oct - Synth	0.1378	< 0.0001	25.6730	0.0376
Nov - Synth	0.1778	< 0.0001	36.1410	0.0269
Dec - Synth	0.1826	< 0.0001	27.6826	0.0573
Jan - Synth	0.0865	0.0186	18.0961	0.0130
Feb - Synth	0.0929	0.0090	19.1153	0.0104

to determine whether they are distributed in a similar manner. These tests were first performed on each month of real data on its following month (Table 6 and Table 8). These values are to confirm the validity of the tests as we know that these datasets are similar. Therefore if our synthetic dataset is equally similar, then it would pass as legitimate traffic. The suite of tests are run against each month of each real dataset and the synthetic dataset (Table 7 and Table 9). The results indicate that the generated data is suitably distributed when compared to the real dataset it is modelled on.

5.3 Visual analysis

Finally, a sanity check is performed via visual analysis of traffic data. The traffic count is plotted with respect to time invoked for all datasets and grouped for visual comparison. Figures 14 and 15 clearly show obvious day/night cycles, noise at the peaks and troughs and a slight variation in traffic over a period of seven days.

5.4 Evaluation

The results presented in Section 5.1 demonstrate that DoWTS generates appropriate volumes of requests when given a set of input parameters. This shown by the similarity in scale of value when comparing the real datasets and synthetic dataset. The mean value of requests in dataset 1 is between 5,131 and 6,245 requests across the five months of observed traffic. When given similar usage parameters to this dataset, DoWTS produces 5,415 requests on average. Similar confirmations are produced across the quantitative comparison fields, suggesting the suitability of DoWTS to produce realistic data.

The results presented in Section 5.2 should be interpreted as follows:

- **KS Test Statistic** - The KS Test is utilised to determine goodness of fit between two samples. It produces two results both bounded between 0 and 1: the KS statis-

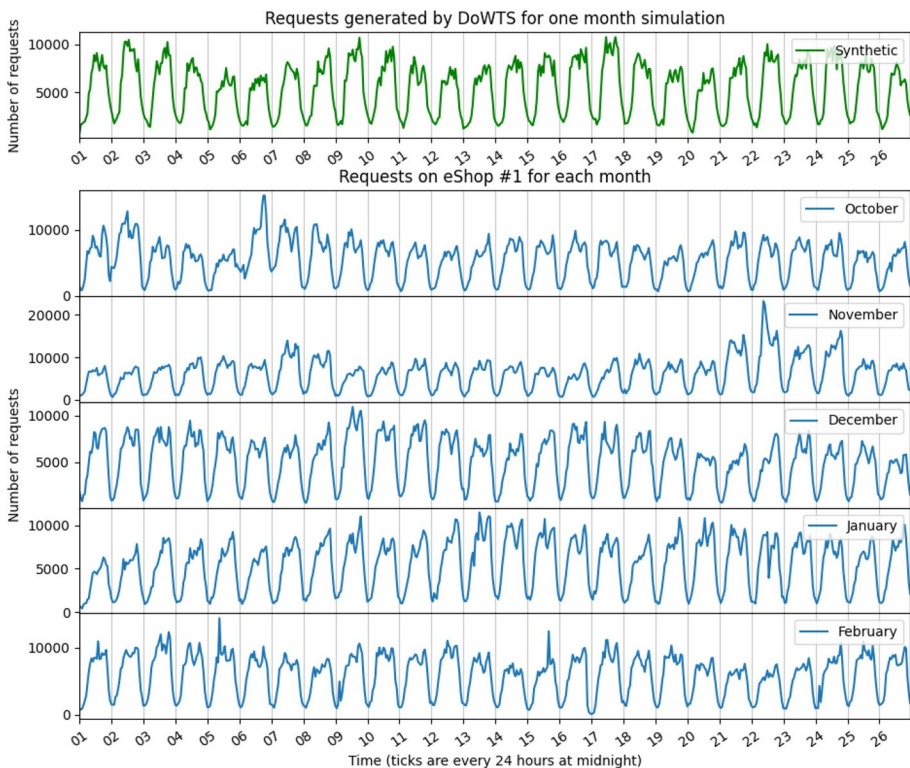


Fig. 14 Visual comparison of requests per hour on DoWTS and real dataset 1

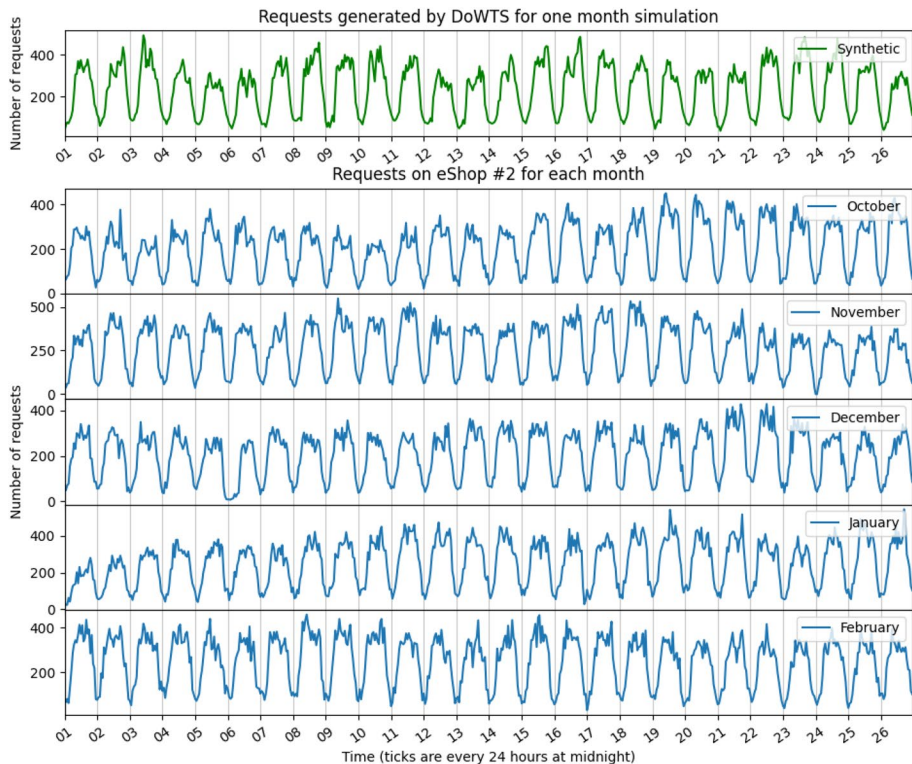


Fig. 15 Comparison of requests per hour on DoWTS and real dataset 2

tic and the p-value. The null hypothesis of the KS Test is that the distributions of datasets are identical if the statistic value is low and the p value is high. In our tests we observe both values to be low. This suggests that we reject the null hypothesis as the distribution of the datasets is not identical. DoWTS does not aim to over fit the data it produces to the two real datasets utilised in this paper, in order to ensure a good variety of produced data. This coupled with the large sample size used in our tests will lead to the KS Test performing overly strict when calculating the p-value, which results in the recorded low values. However, the calculated statistic of the KS Test (the maximum difference between the distributions) suggests that the recorded values in the datasets are at least similarly distributed, although not exact. The KS Test statistic on each month of real data returns values between 0.09 and 0.19 in dataset 1. DoWTS produces datasets with similar KS Test statistics when compared to each month of real data. This suggests that the data generated by DoWTS is no less believable than any other sample of real data. When coupled with the additional tests, we believe the KS Test statistic to be a valuable marker of the suitability of the data DoWTS produces.

- **Wasserstein Distance** - The value obtained from calculating the Wasserstein Distance has no upper limit. As such, interpretation of the results presented should be in relation to each dataset independently. The values calculated when comparing real data are similar to those comparing real to synthetic data.

- **Jensen Shannon Divergence** - The range for Jensen Shannon Divergence is between 0 and 1. The results of our tests demonstrate with suitably low values that DoWTS produces synthetic data similar to the real datasets used for comparison in this paper. No comparison of our synthetic data with real data yielded a Jensen Shannon Divergence greater than 0.06.

From our analysis, we can determine that the synthetic usage generator performs suitably well at replicating real traffic. As such we are confident in utilising it for generating background traffic upon which hypothesised DoW attacks will be launched for use in training of detection and mitigation systems via machine learning.

6 Conclusions and future work

In this work, we furthered an early investigation Kelly et al. (2021) into the threat of DoW attacks on serverless functions. We have further added to the formal definition of DoW in an absence of academic discussion on the topic. We focus primarily on leech attacks and how a preemptive stance can be taken against such attacks in the face of a lack of historical data. The development of DoWTS allows for the rapid generation of attack data that would take too long to run in real time thus accelerating the research process. We have contributed this tool that has been thoroughly evaluated for its suitability to generate realistic data.

Future work utilising the data generated by our simulator in this work will encompass the training of detection algorithms and development of mitigation systems that will be deployed to serverless platform simulators that function in real time Kelly et al. (2021). Such work will further not only knowledge of serverless computing security and DoW prevention but also serve to assist research on bot and *Click Fraud* detection given the attack vectors discussed.

Author contributions The first author wrote the main manuscript. All authors reviewed the manuscript.

Funding Open Access funding provided by the IReL Consortium. The research conducted in this publication was funded by the Irish Research Council under project ID GOIPG/2021/288

Data availability All code will be made available on the first author's Github Kelly (2022).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

Ana (2017) Bot baseline fraud in digital advertising 2016-2017. Report. <https://www.ana.net/getfile/25093>. Accessed 31 Jan 2022

- AWS (2017) Wild Rydes. <https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/>
- AWS (2019) Serverless applications lens aws well-architected framework. Report. Accessed 7 Feb 2022
- AWS (2021) Security overview of aws lambda. Report
- Barna, C., Mark, S., Michael, S., Vassilios, T., Marin, L. (2012) Model-based adaptive dos attack mitigation. IEEE
- Beswick, J. (2020a) Building a location-based, scalable, serverless web app. <https://aws.amazon.com/blogs/compute/building-a-location-based-scalable-serverless-web-app-part-1/>. Accessed 22 Jun 2022
- Beswick, J. (2020b) Load testing a web application's serverless backend. <https://aws.amazon.com/blogs/compute/load-testing-a-web-applications-serverless-backend/>. Accessed 22 Jun 2022
- Cordero, C. G., Vasilomanolakis, E., Wainakh, A., Mühlhäuser, M., & Nadjm-Tehrani, S. (2021). On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM Trans Priv Secur*, 24(2), 8. <https://doi.org/10.1145/3424155>.
- Data Science Campus (2022) Synth Gauge. <https://github.com/datasciencecampus/synthgauge>. Accessed 23 Jun 2022
- Firebrand (2017) Bot Traffic Detection Method Teases Real Website Traffic from Fake. <https://firebrand.net/blog/bot-traffic-detection-tool/urldate=2020-07-01>. Accessed 31 Jan 2022
- Fuentes, M.R. (2020) Shifts in underground markets past, present and future. Report, Trend Micro. https://documents.trendmicro.com/assets/white_papers/wp-shifts-in-the-underground.pdf. Accessed 31 Jan 2022
- He, Z., Zhang, T., Lee, R.B. (2017) Machine learning based ddos attack detection from source side in cloud. In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)* (pp. 114–120). IEEE
- Hoff, C. (2008) Cloud Computing Security: From DDoS (Distributed Denial Of Service) to EDoS (Economic Denial of Sustainability). <https://www.rationalsurvivability.com/blog/2008/11/cloud-computing-security-from-ddos-distributed-denial-of-service-to-edos-economic-denial-of-sustainability/>. Accessed 31 Jan 2022
- Idris, M., Syarif, I., Winarno, I. (2021) Development of vulnerable web application based on owasp api security risks. In: *2021 International Electronics Symposium (IES)* (pp. 190–194). IEEE
- Idziorek, J., Mark, T. (2011) Exploiting cloud utility models for profit and ruin. IEEE
- Kechinov, M. (2020) eCommerce Events History in Cosmetics Shop. <https://www.kaggle.com/datasets/mkechinov/e-commerce-events-history-in-cosmetics-shop>. Accessed 22 Jun 2022
- Kechinov, M. (2021) eCommerce events history in electronics store. <https://www.kaggle.com/datasets/mkechinov/e-commerce-events-history-in-electronics-store>. Accessed 22 Jun 2022
- Kelly, D. (2022) DoWTS - Denial of Wallet Test Simulator. <https://github.com/psykodan/DoWTS>. Accessed 22 Jun 2022
- Kelly, D., Glavin, F. G., & Enda, B. (2021). Denial of wallet—defining a looming threat to serverless computing. *Journal of Information Security and Applications*, 60, 102843.
- Khor, S.H., Akihiro, N. (2009) Spow: On-demand cloud-based eddos mitigation mechanism
- Ko, I., Chambers, D., & Barrett, E. (2020). Feature dynamic deep learning approach for ddos mitigation within the isp domain. *International Journal of Information Security*, 19(1), 53–70.
- Kshetri, N. (2010). The economics of click fraud. *IEEE Security & Privacy*, 8(3), 45–53.
- Kumar, M. N., Sujatha, P., Kalva, V., Nagori, R., Katukojwala, A. K., & Kumar, M. (2012). Mitigating economic denial of sustainability (edos) in cloud computing using in-cloud scrubber service. *Fourth international conference on computational intelligence and communication networks* pp. 535–539. <https://doi.org/10.1109/CICN.2012.149>
- Maki, N., Nakata, R., Toyoda, S., Kasai, Y., Shin, S., & Seto, Y. (2020). An effective cybersecurity exercises platform cyxec and its training contents. *International Journal of Information and Education Technology*, 10(3), 215–221.
- Markus-Go (2008) BoNeSi. <https://github.com/Markus-Go/bonesi>. Accessed 22 Jun 2022
- Niyaz, Q., Sun, W., Javadi, A.Y. (2016) A deep learning based ddos detection system in software-defined networking (sdn). arXiv preprint [arXiv:1611.07400](https://arxiv.org/abs/1611.07400)
- OWASP (2019) Owasp api security project. Report. <https://owasp.org/www-project-api-security/>. Accessed 11 Apr 2022
- Priya, S.S., Sivaram, M., Yuvaraj, D., Jayanthiladevi, A. (2020) Machine learning based ddos detection. In: *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)* (pp. 234–237). IEEE
- REES46 Technologies (2022) Open CDP. <https://rees46.com/en/open-cdp>. Accessed 23 Jun 2022

- Ross, G. (2013) @richorama Denial of Wallet attack! Twitter https://twitter.com/gepeto42/status/331756195574587392?s=20t=SITXovUz_JhYkWQM89FhdQ. Accessed 22 Jun 2022
- Scipy (2022a) Kolmogorov Smirnov Test. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html. Accessed 28 July 2022
- Scipy (2022b) Wasserstein Distance. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wasserstein_distance.html. Accessed 28 July 2022
- Scipy (2022c) Jensen Shannon Distance. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.jensenshannon.html>. Accessed 28 July 2022
- Sqalli, M.H., Fahd, A.-H., Khaled, S. (2011) Edos-shield-a two-steps mitigation technique against edos attacks in cloud computing. IEEE
- Su, J.-M., Cheng, M.-H., Wang, X.-J., Tseng, S.-S. (2019) A scheme to create simulated test items for facilitating the assessment in web security subject. In: *2019 Twelfth International Conference on Ubi-Media Computing (Ubi-Media)* (pp. 306–309). IEEE
- Wilbur, K. C., & Yi, Z. (2009). Click fraud. *Marketing Science*, 28(2), 293–308.
- Xu, S., Marwah, M., Arlitt, M., Ramakrishnan, N. (2021) Stan: Synthetic network traffic generation with generative neural models. *Deployable Machine Learning for Security Defense* (pp. 3–29). Springer

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.