Journal of Cybersecurity, 2024, tyae004
https://doi.org/10.1093/cybsec/tyae004

Research Paper

DoWNet—classification of Denial-of-Wallet attacks on serverless application traffic

Daniel Kelly *, **Frank G. Glavin**, **Enda Barrett**

School of Computer Science, University of Galway, University Rd, H91TK33 Galway, Ireland

*Corresponding author. E-mail: daniel.kelly@universityofgalway.ie

Received 11 January 2023; revised 11 December 2023; accepted 25 February 2024

Abstract

Serverless computing is an ever-growing programming paradigm being adopted by developers all over the world. Its highly scalable, automatic load balancing, and *pay for what you use* design is a powerful tool that can also greatly reduce operational costs. However, these advantages also leave serverless computing open to a unique threat, Denial-of-Wallet (DoW). It is the intentional targeting of serverless function endpoints with request traffic in order to artificially raise the usage bills for the application owner. A subset of these attacks are *leeches*. They perform DoW at a rate that could go undetected as it is not a sudden violent influx of requests. We devise a means of detecting such attacks by utilizing a novel approach of representing request traffic as heat maps and training an image classification algorithm to distinguish between normal and malicious traffic behaviour. Our classifier utilizes convolutional neural networks and achieves 97.98% accuracy. We then design a system for the implementation of this model that would allow application owners to monitor their traffic in real time for suspicious behaviour.

Keywords: Denial-of-Wallet; serverless functions; Function-as-a-Service; machine learning; deep learning; neural networks; image classification; heat map

Introduction

Serverless computing, or Function-as-a-Service (FaaS), has continued to grow and become adopted as a popular paradigm for software development [1]. It enables the abstraction of back-end infrastructure management such that the developer can simply write the business logic code and have it be invoked via some trigger. This code is referred to as a function. Functions only execute when invoked and application owners are only billed for that runtime at a rate predetermined by the function's configuration. This *pay-as-you-go* model of billing has the potential to drastically reduce operational costs. Serverless functions also boast the trait of being highly scalable. Functions are deployed to a large pool of virtual machines (VMs) across the providers cloud infrastructure. This means that serverless applications can handle high loads of bursty traffic without risk of downtime of the application.

These highly attractive selling points open serverless computing to a unique form of abuse known as the Denial-of-Wallet (DoW) [2]. It is the intentional targeting of serverless functions (or other pay-as-you-go services) for excessive invocation with an aim to elevate the usage bill of the application owner. Recent research on DoW has further defined it to be split into *internal* and *external* DoW [3]. This research suggests that external DoW can be detected and mitigated by traditional Denial of Service (DoS) mitigation techniques. However, unlike DoS, DoW may use alternative attack patterns that do not trigger DoS detection parameters as they may not be volumetric attacks, i.e. generating large traffic volumes that seek to cause damage in a short amount of time. Instead, these attacks execute over long periods of time in order to not raise suspicion like a traditional DoS flooding attack (leech attacks).

In this paper, we present our novel approach to representing these attacks for use in a detection system that utilizes deep learning of a convolutional neural network (CNN) for classification of suspicious traffic on an application. The contributions of this body of work are as follows:

(1) Development of a novel approach to translating large volumes of request traffic data into summary image representations and then generating a new dataset of the said representations via data synthesis.
(2) Analysis of the feasibility of approaching DoW detection as an image classification problem and subsequent training of a CNN for malicious traffic classification.
(3) Creation of a system for applying the traffic classification model that would allow application owners to evaluate their traffic

in real time and alert them when potential attacks have been detected.

Related work

DoW in literature

Prior to our initial investigation on DoW [2], the state of focussed research on the attack was non-exhaustive. A number of blogs drew attention to the issue [4–6], although they only go as far as to describe what DoW is, rather than conduct in-depth analyses of the issue. Pursec [7] released a list of 10 security risks that face serverless computing with DoW featured on that list. They do not go into detail on DoW specifically, instead focussing on DoS attacks and suggesting mitigation techniques such as the following:

(1) write efficient serverless functions that 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 Application Programming Interface (APIP) calls;
(5) enforcing proper access controls to serverless functions;
(6) using APIs, modules, and libraries that are not vulnerable to application layer DoS attacks such as Regular expression Denial of Service (ReDoS) and Billion-Laughs-Attack.

In 2017, Open Web Application Security Project (OWASP) included DoW in their risks to serverless report [8]. This is one of the first major reports that categorically lists DoW as a threat. Amazon Web Services (AWS) also released two whitepapers around this time, discussing the security of its Lambda serverless platform [9] and serverless architecture in general [10], which have seen subsequent revisions; however, no mention of DoW is made.

The available literature on DoW was non-exhaustive, with little to no testing to further understand the potential damage this threat could inflict. In 2021 and 2022, we published two papers on the subject. The former being an initial investigation on DoW in order to give it a formal definition in academia [2], and the latter furthering our investigation and polishing our definition along with development of a tool to aid further research [11]. These publications have sparked what appears to be a growing interest in DoW. The Register published an article summarizing our work [12] that garnered good discussion on the article itself and was widely shared on social media. Two articles published in 2022 by two seperate research groups specifically focussed on DoW, one being a continuation of our work [13] and the other investigating a new realm of DoW termed *internal DoW*, where the attack originates within the applications network [14].

Serverless attack surface

Serverless functions can be invoked by various triggers. By far the most common method is via 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 requires to link functions to an API Gateway. The result is a means of executing functions via an Hypertext Transfer Protocol (HTTP) or Representational State Transfer (REST) request to the endpoint. This intrinsic link with serverless functions makes API endpoints the largest attack surface in which DoW could manifest.

Datta *et al.* [15] propose a system to mitigate the flow of indirect functions by monitoring the activity of every API trigger of the function. This research was directed at preventing information theft. However, analysis of all exposed API endpoints is necessary for DoW mitigation as any unprotected function triggers will serve as a point of attack.

There are unique vulnerabilities that can target APIs, as outlined in the OWASP's top 10 risks to APIs [16]. These vulnerabilities serve as an entry point for an attacker to cause DoW. Specifically in the cases of a *Lack of Rate Limiting* and *Insufficient Logging and Monitoring*, an attacker could flood API endpoints with requests quickly driving up costs. However, all of these vulnerabilities can be exploited with the aim of causing DoW.

Efforts to combat API vulnerabilities are largely based on 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 [17–19]. OWASP examples of these applications include:

WebGoat: a deliberately insecure application that allows developers to test vulnerabilities commonly found in Java-based applications that use common and popular open source components (https://owasp.org/www-project-webgoat/).

Juice Shop: a sophisticated insecure web application used in security training, awareness demos, and as a test environment for security tools. Juice Shop encompasses vulnerabilities from the entire OWASP Top 10 along with many other security flaws found in real-world applications (https://owasp.org/www-project-juice-shop/).

Image classification

Image classification is the labelling of images by a pretrained model. The research presented in this paper utilizes such in order to determine whether an attack is likely occurring. In recent years, neural networks have consistently been at the top of the image classification section in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [20]. The ILSVRC is an annual computer vision competition developed upon a subset of a publicly available computer vision dataset called ImageNet [21]. In particular, the use of CNNs has paved the way forward in image classification. The following is a selection of notable CNNs produced for the ILSVRC that achieved high positions in their time.

Visual Geometry Group

Simonyan *et al.* [22] of the Visual Geometry Group (VGG) investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Their submission to the ILSVRC in 2014 received first and second in the localization and classification sections, respectively. They utilized small convolution filter sizes of 3×3 in architectures of increasing weight layers, testing from 11 to 19 layers. They found that the networks with 16 layers (13 convolution layers and 3 fully connected layers) and 19 layers (16 convolution layers and 3 fully connected layers) achieved 74.4% top-1 classification accuracy on the ImageNet dataset.

Residual networks

He *et al.* [23] developed a model that utilizes *residual learning* in order to drastically increase the number of weighted layers from VGG19's 19 layers to 152 layers without the cost of increased complexity. This is achieved by creating *shortcut connections* between stacked layers where the residual representation of the network is generated and then used in the subsequent number of stacked layers. The author's Residual Network with 50 layers (ResNet50) achieved a 75.3% top-1 classification accuracy on the ImageNet dataset.

SqeezeNet

Iandola *et al.* [24] sought to replicate AlexNet [25] level classification accuracy with a more lightweight network that is easier to train and deploy. This is achieved via the following three strategies:

(1) replace 3 × 3 filters with 1 × 1 filters;
(2) decrease the number of input channels to 3 × 3 filters;
(3) downsample late in the network so that convolution layers have large activation.

SqueezeNet achieved 60.4% top-1 classification accuracy on the ImageNet dataset. However, the model was 510× smaller than non-compressed AlexNet.

Xception

Chollet *et al.* [26] aimed to build upon the GoogLeNet (later InceptionV3 [27]) model by re-interpreting *inception modules*, which are a means of making the process of a convolutional kernel mapping cross-channel and spacial correlations easier by treating those channels as separate spaces. The hypothesis is that cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly. Xception takes this hypothesis to the extreme by basing its architecture on entirely depthwise separable convolution layers. Xception achieved 79% top-1 accuracy on the ImageNet dataset.

MobileNet

Sandler *et al.* [28] employ the techniques demonstrated by the previously mentioned models, namely residual representations of the network with shortcut connections and depthwise separable convolution layers. They achieved 74.7% top-1 accuracy with MobileNetV2 on the ImageNet dataset.

Applications of machine learning in cybersecurity

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

Niyaz *et al.* [29] use deep learning for feature reduction of a large set of features derived from network traffic headers for the detection of Distributed Denial of Service (DDoS). They use a stacked autoencoder (SAE), which is a type of Artificial Neural Network (ANN) that consists of multiple layers of autoencoders. An autoencoder is an unsupervised learning algorithm that learns to encode and decode data in an efficient way. The goal of an autoencoder is to learn a compressed representation of the input data, called the latent space, which can be used to reconstruct the original input data. This SAE consists of stacked sparse autoencoders and a softmax classifier for unsupervised feature learning and classification, respectively. A sparse autoencoder 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 function of the input. The sparse autoencoder networks calculate optimal weights of matrices and bias vectors while trying to learn an approximation of an identity function using backpropagation. Multiple sparse autoencoders are stacked, so the output of one feeds into the input of the next while also reducing in dimension. Finally, the last hidden layer is fed into the softmax classifier. Normal traffic data were collected by recording usage on a home network over 72 h. The attack data were generated using *hping3* in an isolated environment. *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 an accuracy of 95.65% using this system.

He *et al.* [30] compare multiple supervised and unsupervised machine learning 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 Brute Force, Domain Name System (DNS) Reflection, Internet Control Message Protocol (ICMP) Flood, and Transmission Control Protocol (TCP) Synchronize 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 Maximization

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

Priya *et al.* [31] also found that Random Forest had the best accuracy, training the algorithm on attacks gernerated by *hping3*.

Ko *et al.* [32] utilized netflow data on an Internet Service Provider and BoNeSi [33] for generating attacks. A dynamic feature selector was devised for use with self-organizing maps in training an attack detection system.

Examples of cyber threat detection utilizing image classification are predominantly on malware detection. Malware can be reduced to greyscale images where a given malware binary is read as a vector of 8 bit unsigned integers and then organized into a 2D array. This can be visualized as a greyscale image in the range [0,255] (0: black, 255: white) [34]. Local binary patterns (LBP) can be extracted from such images in order to classify the malware [35]. This idea is further enhanced by the creation of MalNet-Image [36]. It eases the past difficulty of researching the topic with a large dataset, offering 24× more images and 70× more classes than existing databases of binary image representations of malware.

The role of image classification in cyber threat detection grows as more methods of visualizing attack data are devised. Azab *et al.* [37] advance malware classification from binary images and searching for LBPs to representation of malware as spectroscopy images and uses CNNs for classification.

Wang *et al.* [38] create image representations of raw traffic from *pcap* files in order to detect malware traffic. A four-step procedure is used to trim, sanitize, and anonymize the *pcap* file before translating the byte code into an image. They subsequently train a CNN to identify features representing malware and achieved a 99.17% accuracy in their 20 class classifier.

Taheri *et al.* [39] perform a similar process of creating image representations of raw traffic *pcap* files and CNN training. However, the aim of this research is botnet detection. A classification accuracy of 99.98% was achieved on the CTU-13 dataset [40].

McCullough *et al.* [41] propose a novel method of representing aggregated TCP traffic from the Internet of Things (IoT) as heat maps and subsequently training a CNN to detect DDoS traffic patterns. We take inspiration from this work, adapting the use of heat map representations of traffic for 1 month's worth of HTTP requests to FaaS endpoint triggers, rather than egress TCP traffic from an IoT devices (Fig. 1). They were able to mitigate DDoS by detecting if there was a compromised IoT device on the network that performed the attack and removing them. Their detection approach used the same CNN algorithms that were previously discussed above. The authors reported excellent classification accuracy utilizing existing models of ∼99% and F1 scores of ∼97%.

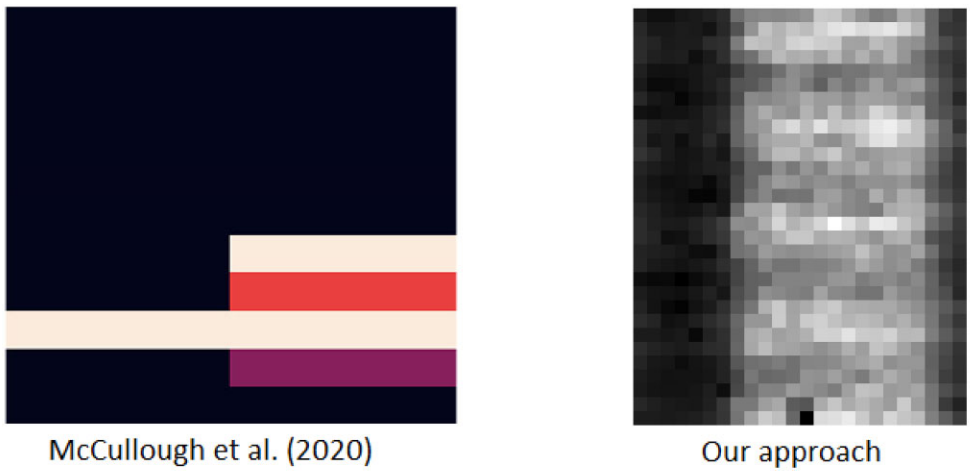


Figure 1. Difference between McCullough *et al.* [41] approach to heat map representation of traffic and ours. The former shows ingress traffic in the left column and egress in the right with each row representing a time step. Our approach maps a month's worth of incoming requests to a 24 × 30 grid.

With the continued persistence of CNNs in all areas of image classification, including cyber threat detection, the use of existing models should be considered, where these models have performed with high accuracy in other domains. The previously discussed selection of notable CNNs produced for the ILSVRC heavily inspired the work presented in our approach to attack detection. Given their success in other domains of computer vision, we concluded that a CNN will serve as the backbone of our detection strategy. We perform an evaluation of the models listed above in this paper and further take inspiration from their underlying architecture in the development of our own model, DoWNet.

DoW attack data generation

Currently, there are no existing publicly available training datasets of traffic on serverless functions. As the aim of DoW is to cause the application owner to receive excessive usage bills, the users of the application are not the ones being affected by the attack. Therefore, if there was to be an attack, it would not serve the victim any advantage to publicly disclose such an attack. This, along with the relative recency of the development and uptake of serverless computing, is potentially why there are no documented large-scale DoW attacks that can be used as training data. We utilized DoWTS [11] in order to generate many different attack scenarios with varying parameters for attack configuration.

DoW test simulator

DoW test simulator (DoWTS) is capable of synthesizing standard normal traffic on a number of function endpoints. It keeps track of total function invocations, total compute time on system, and the cost of such on four of the most popular serverless platforms: Amazon Web Services, Google Cloud, Microsoft Azure, and IBM Cloud [42]. On top of this normal use, DoWTS can launch three types of theorized DoW attack:

(1) Linear request increase: A fixed number of bots deliver a set number of requests per hour (rph) each to the function endpoints, causing the number of requests total to increase linearly.
(2) Geometric request increase: The rph of the bots is multiplied by a set increase factor each timestep, causing the number of requests to increase exponentially.
(3) Random request increase: The rph of the bots is chosen randomly each timestep, causing the number of requests to increase randomly.

These attacks can be configured for high volume sudden traffic (flooding attack) or low volume long time span traffic (leech attack [2]). DoWTS allows for the rapid generation of request traffic logs without actually having to attack a deployed honey pot application and incur real financial losses.

Image representation of function request traffic

The logs generated by DoWTS cover 1 month of traffic, which is the billing cycle of a serverless application. These logs can vary in size depending on the normal background usage and the intensity of the attack. For the normal traffic use case used by default in DoWTS, these files can be expected to be approximately 4GB in size with no attack execution on top. The subsequent attacks can increase this to 20GB in cases where the attack is particularly aggressive (high rate of requests). As such, a method of representing the data to minimize storage space and computation time to process is required.

The traffic data are represented as a heat map, with hour of the day on the x-axis, day of the month on the y-axis, and number of requests relative to the month's worth of requests as the intensity of the square on the grid (as demonstrated in Fig. 2). These heat maps are 24 × 30 pixels in size and single channel greyscale images in order to reduce storage and processing time. The intensity of the traffic is represented as a value of 0–255 relative to the traffic in that month. This procedure is outlined in Fig. 3. The actual heat maps produced for the three types of attack modelled by DoWTS are shown in Fig. 4. Their pixels are not as clearly visible because of scaling up from their original small size.

Dataset characteristics

Using DoWTS to run various attack scenarios and then converting those logs into heat maps, we generated the following dataset:

- In total, 10 000 images were generated.

Total 4000 normal traffic heat maps: traffic modelled by DoWTS based on historic web application use data.

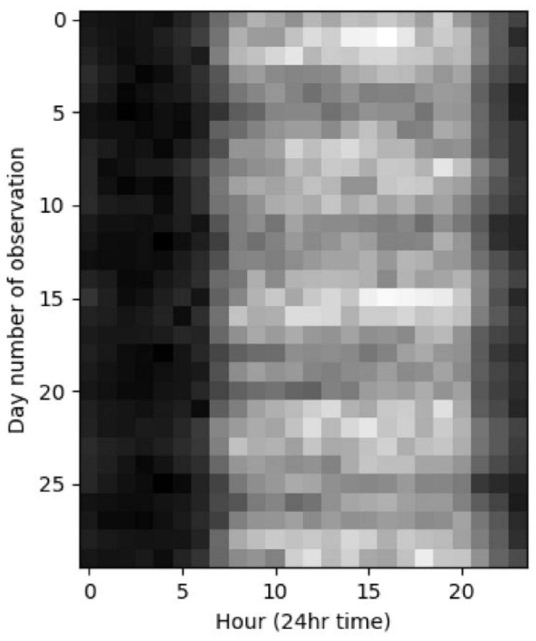


Figure 2. Representation image of a produced heat map with clearly visible pixels where (*x*, *y*) co-ordinate of pixel corresponds to (hour, day) of observation.

Three attack types: randomly choosing a start time as the hour in the month and the duration of the attack in hours.

- Total 2000 linear attack heat maps: randomly chosen from range, fixed rph for each run. Range between 0 and 3000 rph.
- Total 2000 geometric attack heat maps: variable fixed request increase factor. Range between 1.001 and 1.01 increase factor per hour.
- Total 2000 random attack heat maps: variable rph each hour. Range between 0 and 3000 rph.

This dataset is fully labelled for the four classes; normal, linear, geo, and rand. There is also an accompanying log file that lists the attack parameters and the damage caused. However, it is not used for our detection system.

In summary, our dataset consists of 10 000 labelled images of heat maps. These heat maps are labelled for the traffic they represent on a serverless application, i.e. the number of function invocations an hour. The traffic representations are: normal use traffic, linear increase attack traffic, geometric increase attack traffic, and random rate increase attack traffic. The data produced cover a range of attacks intensity for training of a robust detection model.

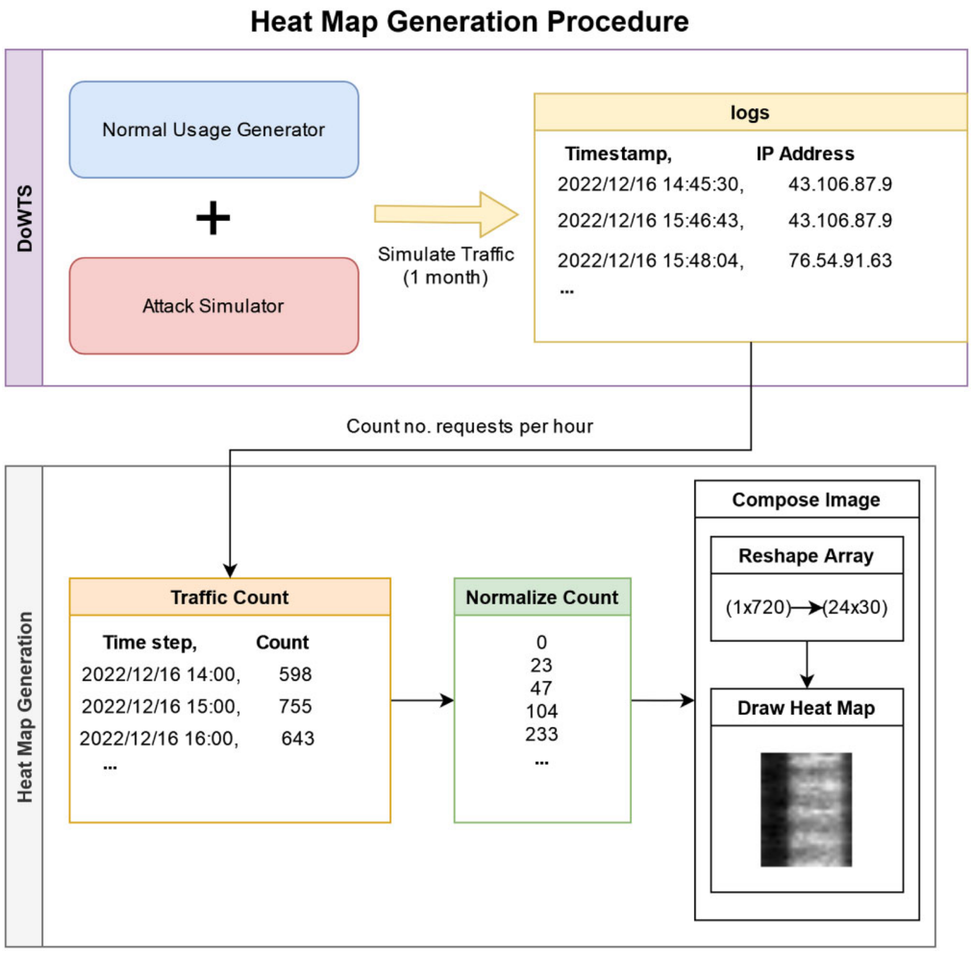


Figure 3. Full life-cycle of data generation from DoWTS to heat map creation.

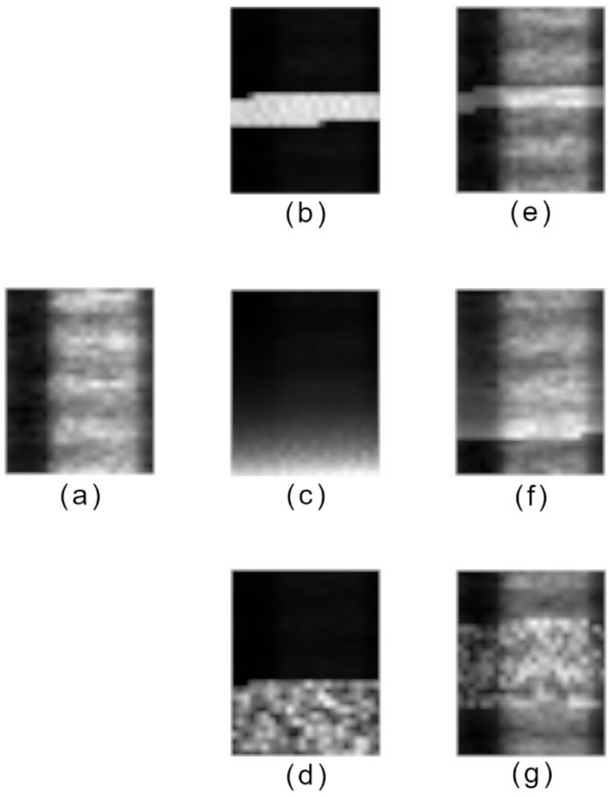


Figure 4. Heat map representation of request traffic: (a) normal traffic, (b) linear rate attack, (c) geometric rate attack, (d) random rate attack, (e) difficult to detect linear rate attack due to lower attack intensity, (f) difficult to detect geometric rate attack due to lower attack intensity, and (g) difficult to detect random rate attack due to lower attack intensity. Note that the images are scaled up from 24 × 30 pixels for visibility.

Table 1. Comparison of preexisting image classification models and two non-CNN based models. Highest values are in bold.

Model	Accuracy (%)	F1 (%)	Precision (%)	Recall (%)
MobileNetV2 [28]	95.66	95.79	95.85	95.66
ResNet50 [23]	98.03	97.29	98.13	97.93
SqueezeNet [24]	**98.44**	97.12	**98.49**	**98.44**
VGG16 [22]	97.48	96.92	97.62	97.43
Xception [26]	98.34	**97.78**	98.43	98.24

Image classification model for detection of attacks

Analysis of popular CNNs for image classification

In the section "Image classification," we outline a number of image classification CNNs that achieved notable results on the ImageNet dataset. These networks are publicly available as Keras Applications [43]. As such, we perform an evaluation of this selection of networks on our dataset for the DoW attack detection problem domain. Table 1 lists the classification accuracy, F1 score, precision, and recall of each model as defined in equations 1, 2, 3 and 4, respectively, where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

All models were trained on 1500 images of normal, linear attack, geometric attack, and random attack traffic each. The images were scaled up by a factor of three in order to meet the minimum input size for the models. They were also loaded as three-channel images. An 80/20 training to validation split was used on this data, where the validation set is used for tuning of hyperparameters in the model. A final dense layer was added to each network as the output layer for classification of the four labels using a *softmax* activation. Each model was trained for 10 epochs to ensure a reasonable comparison. A holdout dataset was used for evaluation containing 2000 images of even ratio between the four classes with 500 examples from each class. The model is then trained again on an 80/20 split with the samples shuffled a further nine times. This process is visualized in Fig. 5.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{1}$$

$$\text{F1} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}, \tag{2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4}$$

From the results shown in Table 1, all five CNNs achieve high results across the four metrics. Any of the five predesigned networks would make good candidates for implementation in a deployable detection system, given the high results demonstrated. However, we observe that the network with a much more simplistic architecture, VGG16, performs almost as well as three of the higher complexity networks and better than MobileNetV2. The tradeoff of complexity to finer accuracy is relevant as in the following sections we will demonstrate a deployment of the final model on a serverless application where a lighter weight model will be necessary.

Creation of a domain specific model

As the problem domain of detecting DoW attacks, at present, is limited to the three proposed basic attack patterns [11], it is unnecessary to employ the networks discussed above. We deduce that the use of one of these networks designed to classify 1000 classes (from the ImageNet dataset) is excessive. As such, in this section, we outline the design and performance of a model of significantly lower complexity.

We shall henceforth refer to our proposed model as *Denial-of-Wallet Network (DoWNet)*. The network architecture is shown in Fig. 6. Inspired by the architecture of VGG16, DoWNet comprises three convolution layers and four dense layers. Pooling is performed after each convolution layer, and a dropout layer is included before the dense layers to prevent overfitting. Each convolution layer uses Rectified Linear Unit activation and filters of size 3. The output layer uses softmax activation for multiclass classification. Images are loaded in their original 24 × 30 size and as a single channel image. This further reduces complexity by not requiring the search for cross-channel correlation.

The results shown in Table 2 demonstrate that DoWNet performs almost as well as the top three classification models in the section "Analysis of popular CNNs for image classification," and outperforms VGG16 and MobileNetV2. We do not claim that DoWNet is a better performing model than these models. However, since this classification task is not as difficult as the ILSVRC, we have shown that a more simple model can achieve similar performance. The advantage of this is that there is less time required to train and use the model. SqueezeNet was the model that performed best in our tests with a classification accuracy of 98.44%. It took 777.79 s to train the model. Whereas, DoWNet with an accuracy of 97.68% took 11.8 s.

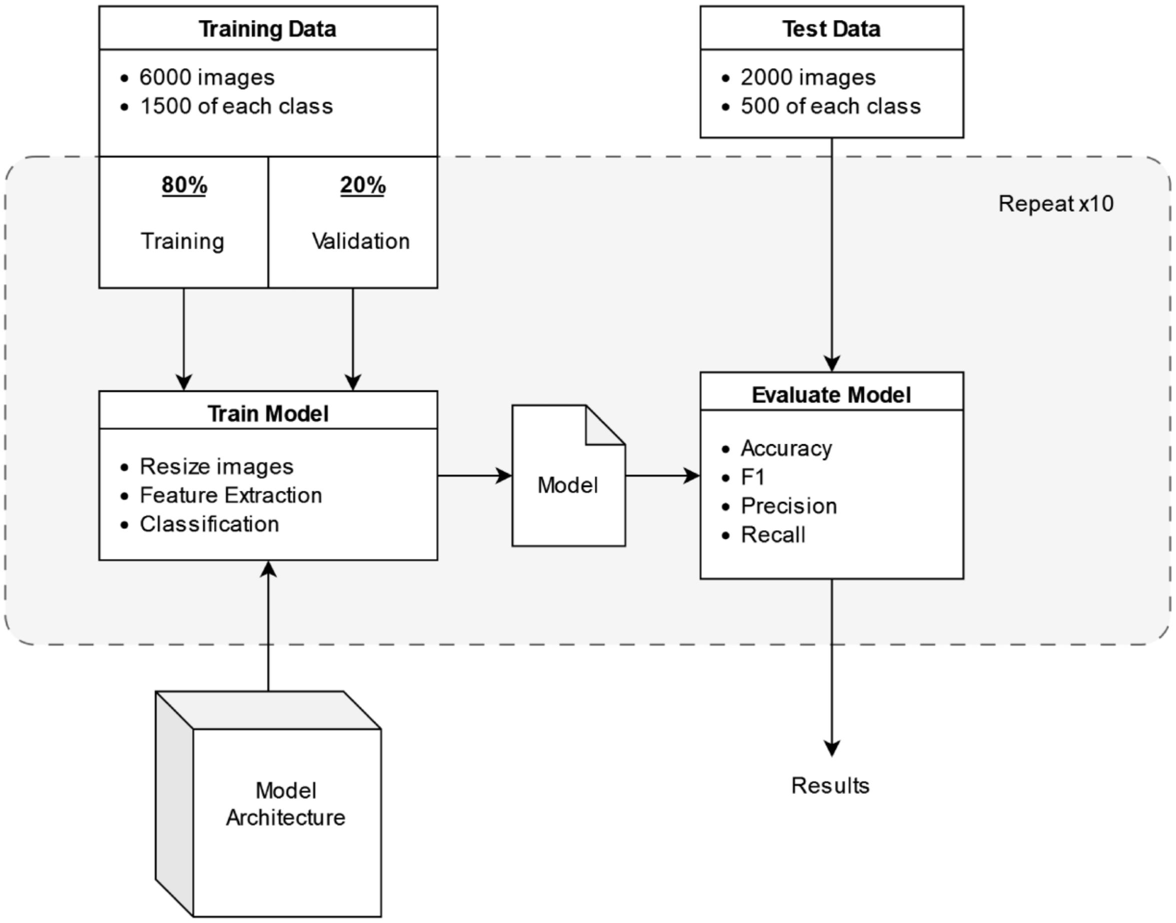


Figure 5. System of training and evaluation of models.

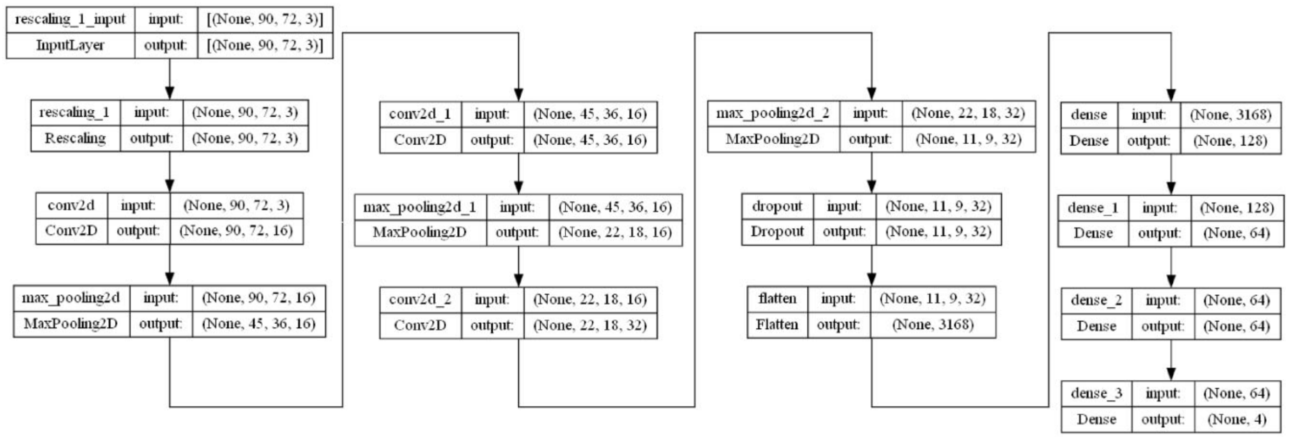

rescaling_1_input | InputLayer | input: [(None, 90, 72, 3)] | output: [(None, 90, 72, 3)]
rescaling_1 | Rescaling | input: (None, 90, 72, 3) | output: (None, 90, 72, 3)
conv2d | Conv2D | input: (None, 90, 72, 3) | output: (None, 90, 72, 16)
max_pooling2d | MaxPooling2D | input: (None, 90, 72, 16) | output: (None, 45, 36, 16)
conv2d_1 | Conv2D | input: (None, 45, 36, 16) | output: (None, 45, 36, 16)
max_pooling2d_1 | MaxPooling2D | input: (None, 45, 36, 16) | output: (None, 22, 18, 16)
conv2d_2 | Conv2D | input: (None, 22, 18, 16) | output: (None, 22, 18, 32)
max_pooling2d_2 | MaxPooling2D | input: (None, 22, 18, 32) | output: (None, 11, 9, 32)
dropout | Dropout | input: (None, 11, 9, 32) | output: (None, 11, 9, 32)
flatten | Flatten | input: (None, 11, 9, 32) | output: (None, 3168)
dense | Dense | input: (None, 3168) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 64)
dense_2 | Dense | input: (None, 64) | output: (None, 64)
dense_3 | Dense | input: (None, 64) | output: (None, 4)

Figure 6. DoWNet layer architecture.

Table 2. DoWNet performance metrics.

Model	Accuracy (%)	F1 (%)	Precision (%)	Recall (%)
DoWNet	97.98	97.92	97.98	97.98

There is great importance in these short training times, as this attack is so new, it is constantly evolving, and retraining will be continuous beyond the attack patterns demonstrated in this paper. The size of the files created when saving the models for use in further applications was also notably different, with SqueezeNet generating a 1081-kb file and DoWNet generating a 231-kb file. The reduced size and time overhead make our simple model more suitable for use in a lightweight detection system that would sit on top of a serverless application. Figure 7 demonstrates DoWNet's classification on the holdout test dataset.

Where there were misclassifications, they were predominantly heat maps that had such little attack traffic that it looks like a normal traffic heat map. These low attack rate heat maps were generated as

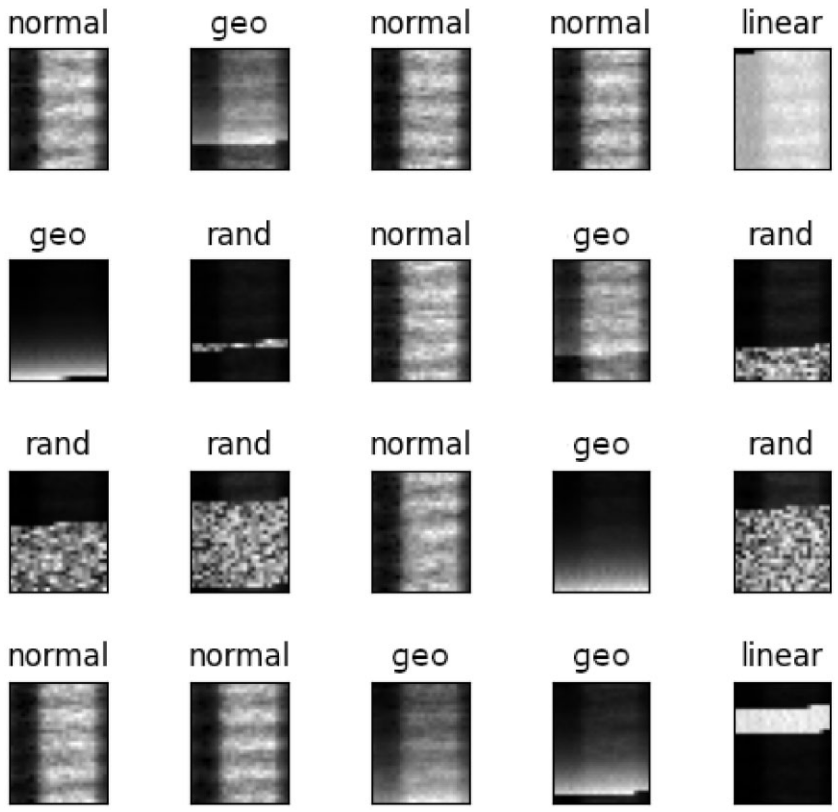


Figure 7. DoWNet successful classification of test data.

a result of the range of rph for attacks starting at 0. We believe these examples are important as they make the model more robust and capable of detecting attacks that are not just incredibly high rate (that emulate DoS flooding attacks). We relate this to cat vs. dog classification where there may be images of dogs that look like cats and visa versa.

Implementation of model in deployed application scenario

DoWNet is capable of accurate classification on heat maps that represent 1 month's worth of request traffic. As such, this approach is limited to checking whether an application is under attack once a month, as you would need to have gathered the request log data to compose a new heat map. In order to address this limitation, we propose a method of utilizing DoWNet that allows for daily assessment of request traffic for signs of attack.

Pixel streaming heat map

As DoWNet is trained on images of a month's worth of request traffic represented as a heat map, it must be supplied with a full image to perform classification. Our solution is to continually update the image pending new values from the recorded traffic of the previous hour. We term this *pixel streaming* as our images are 24 × 30 pixels images where each pixel represents an hour's traffic intensity at a specific time during the month. By continually streaming in new values for pixels in a sliding-window fashion, we can update the image to represent the traffic trends in the heat map. There are two key aspects that allow this to work.

(1) There must be a baseline month of data for the first use. A heap map must be generated for the updates to take place on. Upon this heat map, the streaming will commence and from then will continue to use the previously streamed data as the historical traffic values.
(2) All values for each pixel in the image are re-normalized with each timestep. The actual count of rph that dictate the image's pixels is separate from the normalized values represented in the heat map. As new requests come in on each hour, those values are updated, and a new normalized array is generated for composition of the heat map.

Figure 8 demonstrates how pixel streaming and DoWNet are used together for DoW classification. The top-left heat map is determined to be normal traffic. In this example, this happens to be the initial baseline traffic required for the pixel streaming. However, it could also be the product of months of streaming where there were no attacks taking place. The top-right heap map is the result of 10 days of streaming. As each pixel represents an hour, there have been 243 iterations of updates and normalization to the heat map. Visually, we can see that a faint line is developing and DoWNet is now classifying the traffic as a random increase rate attack. In this example, a geometric increase attack is being launched at this time. The bottom-left heat map shows further updates to the heat map after 18 days. DoWNet had previously been classifying the attack as random and then linear increase rate until the features present were enough to be classified as geometric. The final heat map is after the full run of 1 month with the geometric attack.

This solution serves as an early warning system that an attack is taking place. At the 10 day mark where DoWNet first became suspicious of the traffic, an application owner can take appropriate mea-

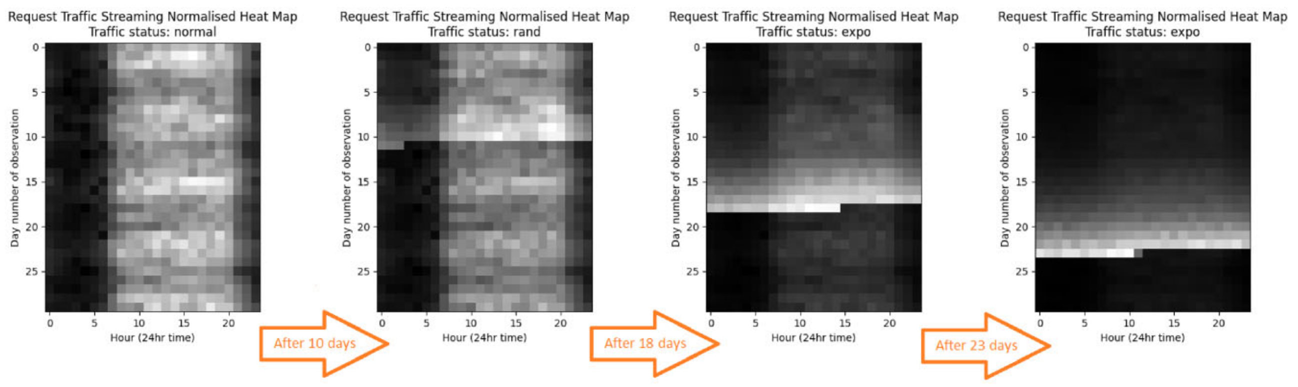


Figure 8. Progression of streaming the total number of requests for each hour and normalizing the data to create a new heat map for analysis. The scenario shown is an geometric attack that starts roughly on day 5 and increases the rate of requests for 20 days. The detection model initially classifies the attack traffic as random until more data are available before it is correctly identified as an geometric attack.

sures to investigate and secure their endpoints before the full damage of the attack can occur, such as the mitigation approaches discussed later in this paper.

Application of solution on deployed service

We believe that an appropriate deployment of DoWNet should have as little impact on the architecture of an application as possible, since it should only run once an hour. As such, it is a prime candidate for being written as a serverless function itself. Deploying DoWNet into a function that lives within the same space as the rest of the application means that there is no requirement to parse data to a third party, thus mitigating potential breaches in privacy. Another great advantage is that the whole system runs on the developers cloud platform, allowing for ease of maintenance and setup.

We present a proof-of-concept system that operates on the OpenFaaS serverless platform. Our testbed follows the isolation zone for serverless attack testing outlined in our previous work [2]. An additional MongoDB database was used for persistent storage of the traffic count at each hour.

DoWNet was written into a function that executes Python code. The function was written in such a way that it could be translated to other serverless platforms, remaining as *platform agnostic* as possible. The operation of the function is as follows:

Prerequisites

(1) NoSQL database; each major platform offers its own solution; our example uses MongoDB;
(2) data collection of 720 h (24 hours × 30 days) of traffic counts; this is required to generate the first heat map;
(3) document that contains the current hour number and previous total invocations.

Function

(1) connect to the database and get the current hour and the previous total for the new value overwrite;
(2) utilizing the platform's provided metrics system, fetch the total count of function invocations; OpenFaaS uses Prometheus, whereas AWS uses its own product, CloudWatch; these metrics are available via API requests;
(3) calculate the total for that hour using the previous total; update hour invocation count, new total, and new current hour in database;
(4) shape the data with the new hour count into the 24 × 30 grid and normalize into heat map;
(5) apply the DoWNet model to heat map;
(6) report results.

The operation of this system is shown in Fig. 9

Results

An experimental testbed was set up that consisted of the following:

- a dummy application to represent an application;
- the deployment of DoWTS;
- a traffic generator utilizing the same synthetic traffic generator as in DoWTS;
- a preloaded collection of a months traffic with the usage set to not exceed 2500 requests per hour. This is a low-scale example to avoid the need to use a botnet for sending attack requests. As DoWNet is trained on normalized data, the reduced scale should not affect the results.

Two control tests were run to establish that the traffic generator when executing only normal traffic does not trigger false positive detections. The system ran for 24 h with the synthesized traffic starting on hour 48 (in order to emulate a system that has already been running for a period of time). Figure 10 shows the change in the heat maps over 24 h in both tests. Figure 11 shows the prediction values for each traffic classification over the 24-h period.

An attack was then launched on the dummy application, starting on hour 115 of the month, and used the random rate increase attack vector. The attack was detected successfully after 5 h. Figure 12 demonstrates the effect of the attack on the generated heat map and Fig. 13 graphs the prediction values of DoWNet.

The results show a clear success in detecting DoW attacks. The function running DoWNet executes in under 2 s, thus requiring little to no cost over head to run given its low-frequency invocation (once an hour).

Discussion

Challenges of DoW detection

As discussed in this paper and in previous literature [2,11], *external* DoW attacks need not simply be DoS style flooding attacks. The threat of lower intensity leeches would not be detectable by traditional DoS mitigation's such as rate limiting or packet analysis as

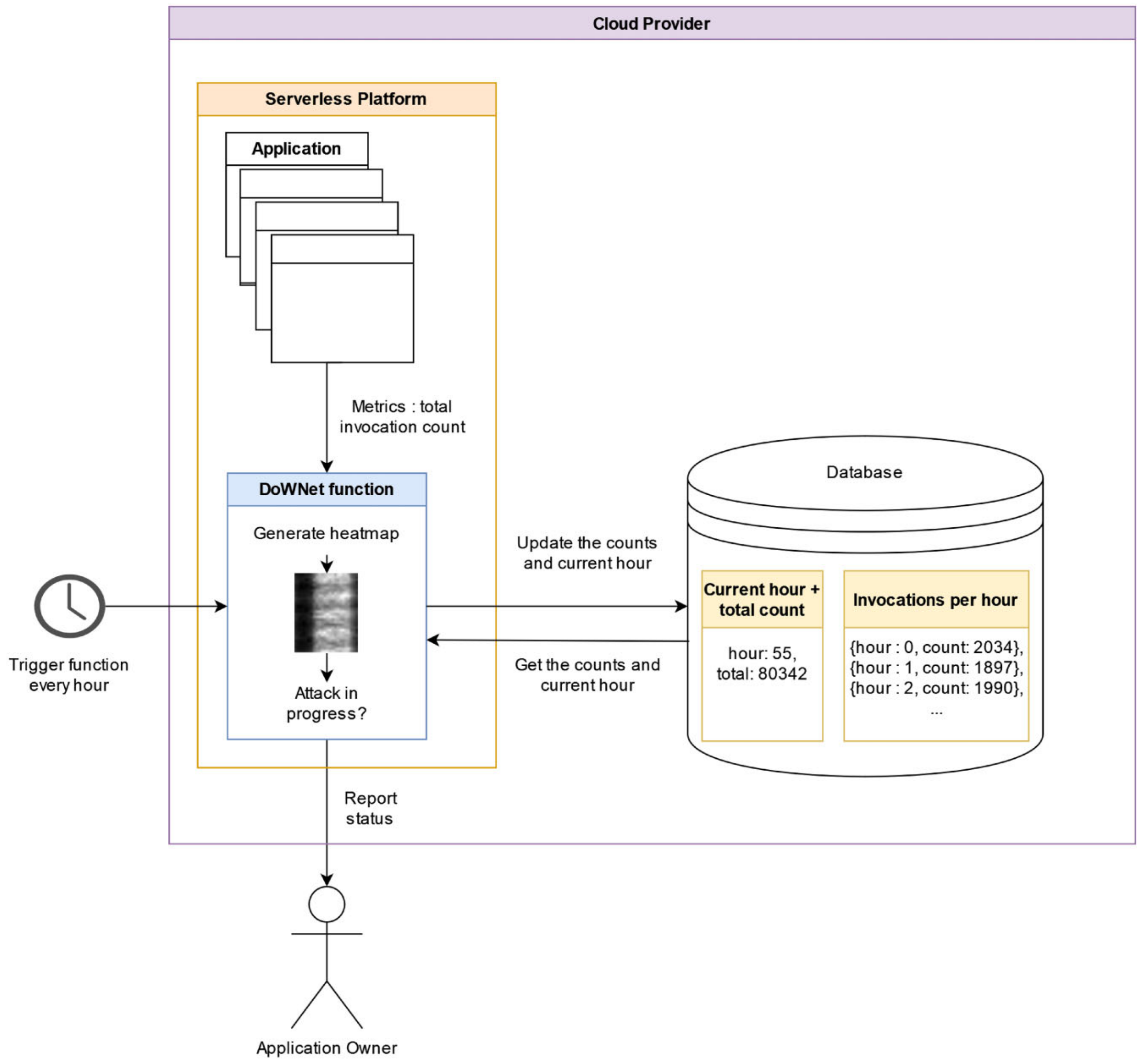


Figure 9. Operation of proof-of-concept deployment of DoWNet.

individually they would be indistinguishable from normal traffic. Therein lies the initial challenge with DoW detection that we have set out to solve.

Through the translation of traffic into small heat map images, we were able to effectively represent long spans of data to facilitate training our CNN. As with leech attacks, they are most effective over these long time spans. The heat map representation is also robust in that since the values are normalized, DoWNet is trained to recognize patterns rather than suspicious values. This would be the case if we treated the requests individually. Reoccurring IP addresses could be easily detected via machine learning (ML) using a clustering algorithm. However, this would not be a robust system for further attack detection as it could lead to blacklisting entire regions from access to an application.

Our proposed detection system using DoWNet is successful in providing early warning to an application owner. It is also lightweight in that it is only ever analysing one image. Since that one image is continuously updated, there is no need for storage of multiple heat maps.

Effective use of DoWNet

DoWNet need only be used where the pixel streaming analysis takes place. We recommend that it should also be deployed in a serverless function to be executed every hour. It may gather request traffic data via the serverless platform's inbuilt accessor functions. However, in order to avoid vendor lock-in, a small helper script can be appended to each function in the application that will update a document containing the number of invocations of the functions. This document should then be accessible to the pixel streaming code. We advocate for the deployment of DoWNet on a serverless function operating within the existing application's serverless infrastructure rather than centralizing the detection within the firewall offering of

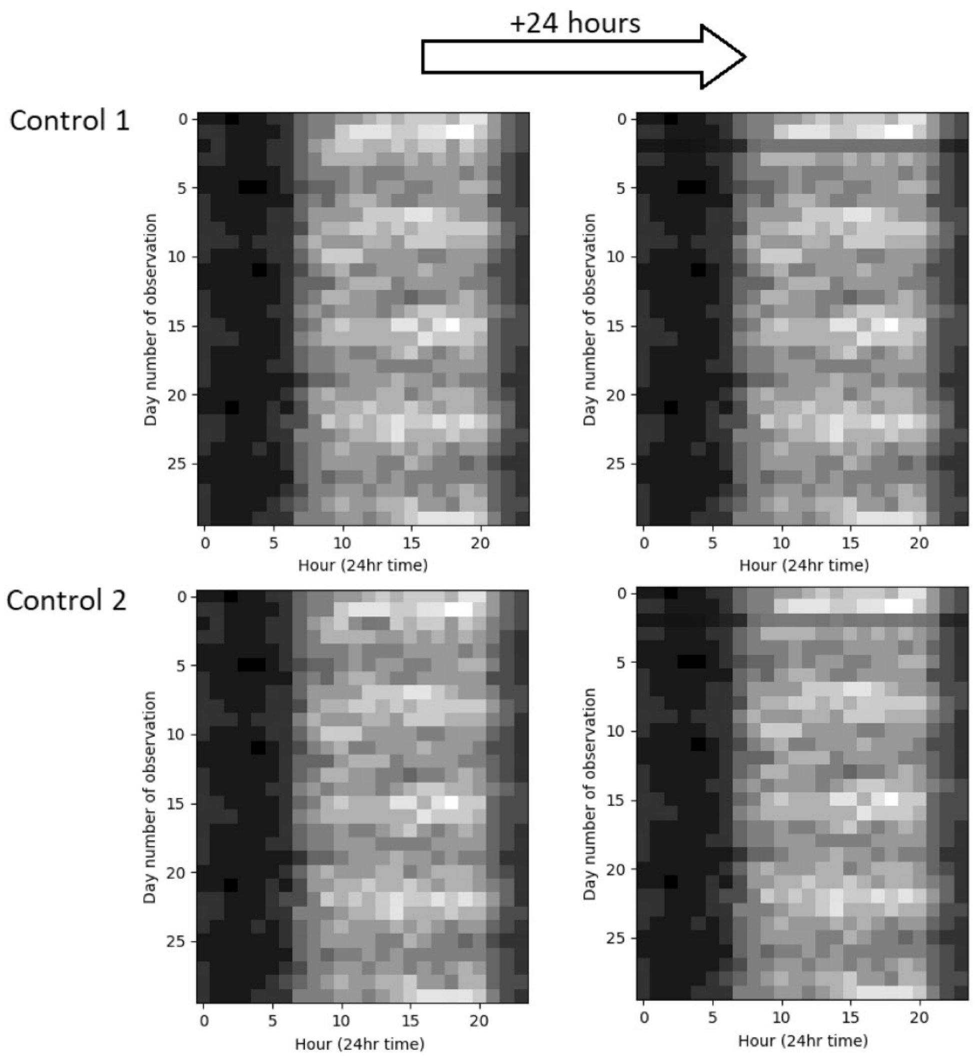


Figure 10. Change in the heat maps over 24 h in control runs.

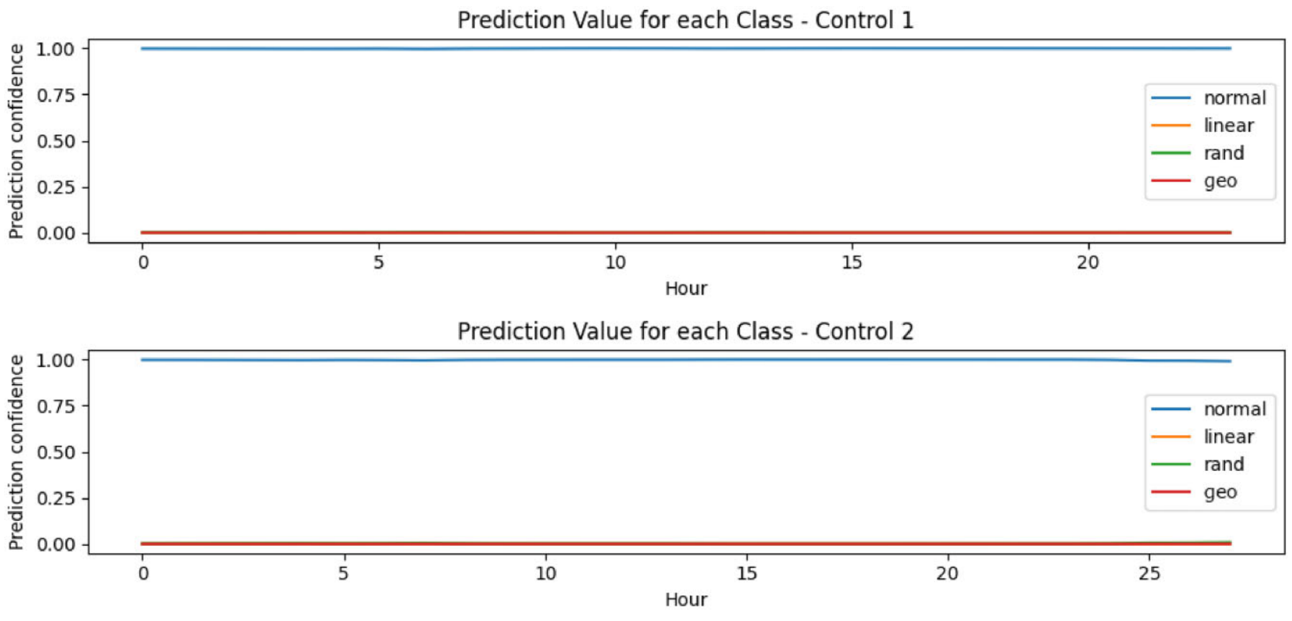


Figure 11. Prediction value of each class over 24 h in control runs.

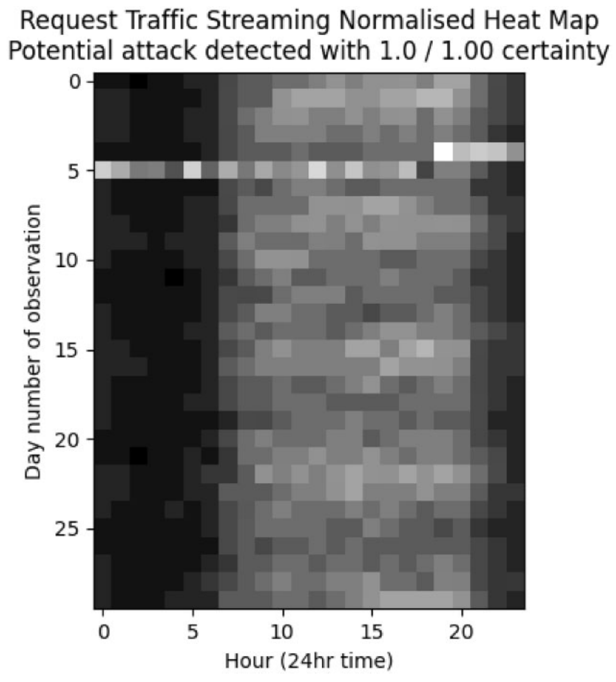


Figure 12. Resulting heat map after the 24 h attack scenario. Note: 1.0/1.00 is how Tensorflow represents 100% when displaying results from classification.

the cloud provider. We believe in security driven development, where developers should be coding with security in mind. By providing an open source Python package that is easy to deploy, we are putting the power in the hands of the developer, allowing them to be in control of their security without the need to enlist third party services for scrubbing of traffic, storing data, etc. This is especially important as serverless computing becomes increasingly popular and is adopted by solo-operator developers as the paradigm of choice when creating their products. A free to use open source package will serve these budget conscious users to secure their functions. DoWNet is a detection system for early warning of potential attacks. Its value is that it provides objective interpretation of the traffic to the application owner to avoid cases where they may have been fooled into thinking that the traffic is normal or are not monitoring it rigorously enough. A detection from DoWNet should be followed up with appropriate mitigation depending on the serverless platform in operation and the requirements of the application owner. Mitigation can vary from shutting the functions down to wide-scale IP address blocking. The former, however, is effectively DoS, and the latter may be difficult if a large botnet with changing IP addresses is being used.

Mitigation of DoW

Securing a serverless function from threats, not limited to DoW, falls in the hands of the application developer. AWS highlights a *shared responsibility model*, where the developer is responsible for the security of the code and explicitly states that poorly configured code resulting in adverse repercussions may not receive sympathy when reported [10]. Steps to protect applications are highlighted in their whitepapers to develop well-architected frameworks and the security overview of AWS Lambda [9,10]. 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 the following:

(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 X-ray tracing should also be utilized 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 in the implementation of secure applications. As serverless functions are often invoked via API triggers, the following mechanisms are recommended:
   - *AWS IAM authorization*: only allow users within AWS environments access with IAM credentials (protect against internal attack by a compromised user). Always give the least amount of access.
   - *Cognito user Pools*: built in user management or integration with social login. Uses Oauth scopes.
   - *API Gateway Lambda authorizer*: 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 Optimization 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 that generate elevated costs.

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

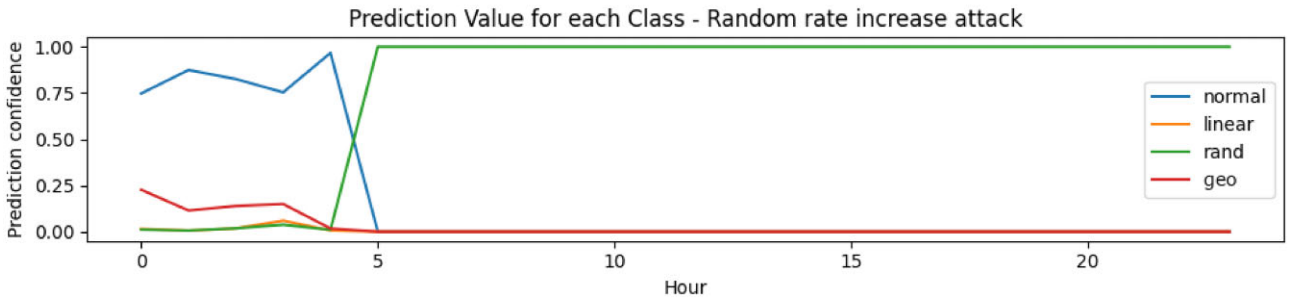
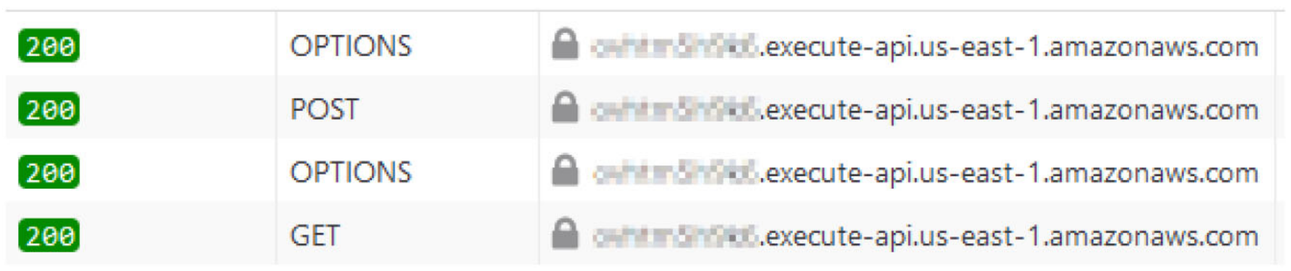


Figure 13. Prediction value of each class for 24 h in attack scenario.



200	OPTIONS	[redacted].execute-api.us-east-1.amazonaws.com
200	POST	[redacted].execute-api.us-east-1.amazonaws.com
200	OPTIONS	[redacted].execute-api.us-east-1.amazonaws.com
200	GET	[redacted].execute-api.us-east-1.amazonaws.com

Figure 14. API endpoints that trigger functions can be sniffed on the network by searching for URLs that match the platform template (unique ID blurred for privacy).

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 the privacy of credentials and business critical data. Input validation to minimize risks associated with API requests. Vigilance in monitoring and dependency vulnerabilities.
- *Data*: encryption in Transit, such as Secure Sockets Layer.
- *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 the 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 [44,45] are susceptible to API vulnerability attacks, such as capturing an endpoint URL (Fig. 14) 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. An attacker is not required to use a large number of nodes to cause damage, as 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.

Further to the best practices discussed, there are a number of *Cloud Security Services* that provide DDoS protection. These are either offering of the cloud platform itself, such as AWS Shield (https://aws.amazon.com/shield/) or Google Cloud Armor (https://cloud.google.com/armor), or external services that operate as a middleman to analyse traffic on an application, such as Cloud Flare (https://www.cloudflare.com/en-gb/ddos/) or Akami (https://www.akamai.com/solutions/security/ddos-protection). While DDoS protection would do little to prevent leech attacks given their vastly different attack rate, these services are an effective safety measure by entrusting the handling of an application's security to a tailor made tool and/or expert third party. These DDoS protection services are also prime candidates for deploying DoW recognition algorithms as they are already monitoring all the traffic to an application. We believe that such an offering, whether it is built on the findings of our research or not, will soon become mainstream.

Limitations

Despite the contributions outlined above, this work does have limitations, as outlined below.

Synthetic data robustness

The presented system for generating synthetic usage traffic, DoWTS, utilizes a heuristic for best approximation of traffic loads over a 24-h period, given a known rough usage base. The heuristic is based on examples of traffic to real ecommerce websites, where there were clear trends in usage, with dips at night and peaks during the day. We concluded that our synthetic data closely follow this example data. However, the load patterns DoWTS replicates are not conducive of an exhaustive sample of *normal* traffic. Therefore, in the training of DoWNet, we are aware that its classification of *normal* traffic may not be sufficient.

We believe that DoWTS provides a perfectly usable dataset for development of mitigation strategies. However, in order to create more robust classifiers, reliance on said synthetic data is not ideal. Instead, a wider sample of example data of normal traffic patterns is required, upon which the use of more complex forms of synthetic data generation such as GAN and VAE should be considered.

Willingness of data sharing

Building on the previous limitation, datasets of normal usage on a serverless application and suspected DoW attacks are required for further research. Normal usage data may become more available over time, just as there are traffic logs of traditional web applications.

However, the biggest hurdle in terms of data acquisition will be recordings of suspected attacks. Unlike DoS, where customers are impacted by the attack and therefore a statement must be made by the service owner, DoW only affects the owner. As such there would be no requirement to publicly disclose that they were a victim as this may make investors and other stakeholders uneasy. Data would need to be willingly published for the research community in a selfless pursuit of furthering knowledge on the subject.

Feasibility of attack

The greatest limitation of this research is the double edged nature of what makes it unique. By being a preemptive analysis of an unrealized attack, it brings into question the viability of the attack entirely. This research has categorically proven that the attack is possible and is indeed a threat to serverless computing if it is executed correctly. However, with a lack of historical examples, the feasibility of the attack comes into play. We are of the opinion that regardless of this limitation, research on the topic should be conducted so that if the attack is realized, there is a body of work to fall back on when devising means of defence. We have outlined potential motivations to conduct DoW and rudimentary attack vectors that bad actors may utilize. Also, it is important to look at existing attack types like DDoS. While no one has publicly admitted to performing a *DDoS* attack on serverless functions to cause a DoW, it is entirely feasible to have happened. Until there is a major occurrence made public, this limitation will remain at the forefront of future researchers minds.

Future work

In summary, we have applied the novel method of representing request traffic as heat maps for training a DoW classification algorithm. This method overcomes the hurdles obstructing the application of ML to DoW detection that are, enormous volumes of data and difficulty in distinguishing individual requests as malicious or benign. The CNN we trained for the purpose of DoW classification, DoWNet, achieved consistent results in detecting hypothetical attacks proposed [11], with accuracy, F1 score, precision, and recall of ∼97%. This model was then successfully applied to a system that streams incoming request traffic numbers for the redrawing of heat maps with respect to historical values. These heat maps undergo analysis by DoWNet, which will report on the status of the traffic at a given hour to serve as an objective early warning system for application owners. This system was then shown to be capable of being deployed within a serverless function. This lowers the barrier of difficulty in implementing such a detection system on an existing application as it would not require any adaption of existing functions.

Continued research will need to migrate to actual deployments of applications. Our isolated zone for testing serves as a great starting point for mitigation system validation. However, before any product of research is accepted for use in the field, it will need to prove itself on commercial platforms. To this end, collaboration with the commercial platforms is imperative so that the financial damage caused through testing is not a factor that would cause research teams to avoid this area.

The issue of data acquisition must be addressed. If there are suspected attacks taking place they must be reported. Whether an anonymous means of reporting is required or collaboration with large companies utilizing serverless computing needs to be setup, somehow the veil must be lifted on DoW in order for the research community to properly tackle the attack.

This research has shown that image classifiers are a powerful tool for cyber threat detection. A great deal of information can be conveyed within an image and given the current spike in the effectiveness of image classification algorithms (as demonstrated by the ILSVRC), this method of threat detection shows strong promise.

Image classification proved especially powerful in our use case, as we were translating log files that would total over 20GB in size into images that are barely over a kilobyte. This drastically more efficient means of storing information would prove useful for analysis of other attacks, especially those that execute over long time spans. It may also prove effective in related fields such as bot detection, click fraud, and espionage detection with potential additions to the image generation process such as Gramian angular field generation [46].

As image recognition packages become more streamlined, it brings with it the possibility of utilizing image classification cyber threat detectors in low processing power deployments, such as serverless functions. This may be an avenue that could disrupt the current cybersecurity market where clients are reliant on outsourcing their data to an external service for processing, rather than running their security solution on their hardware or cloud.

Acknowledgments

The authors thank the anonymous reviewers for their valuable suggestions.

Author contributions

Daniel Kelly (Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Writing – original draft), Frank G. Glavin (Project administration, Writing – review & editing), and Enda Barrett (Project administration, Writing – review & editing).

Conflict of interest: No competing interest is declared.

Funding

The research conducted in this publication was funded by the Irish Research Council under project ID GOIPG/2021/288.

References

1. Ot A. The Serverless Computing Market in 2022. 2022. https://www.enterprisestorageforum.com/cloud/serverless-computing-market/. (14 March 2024, date last accessed)
2. Kelly D, Glavin FG, Barrett E. Denial of wallet—defining a looming threat to serverless computing. *J Inf Sc* 2021;**60**:102843. https://doi.org/10.1016/j.jisa.2021.102843.
3. Shen J, Zhang H, Geng Y. *et al.* Gringotts: fast and accurate internal Denial-of-Wallet detection for serverless computing. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22*. New York, NY: Association for Computing Machinery, 2022, 2627–41.
4. Dooley D. Serverless, shadow APIs and Denial of Wallet attacks. 2019. https://www.helpnetsecurity.com/2019/03/29/serverless-challenges/. (14 March 2024, date last accessed).
5. Sachenko R. Severe truth about serverless security and ways to mitigate major risks. 2020. https://hackernoon.com/severe-truth-about-serverless-security-and-ways-to-mitigate-major-risks-cd3i3x6f. (14 March 2024, date last accessed).
6. Rahic A. Fantastic Serverless security risks, and where to find them. 2018. https://www.serverless.com/blog/fantastic-serverless-security-risks-and-where-to-find-them/. (14 March 2024, date last accessed).
7. Pursec. The ten most critical security risks in serverless architecture. 2018. https://www.puresec.io/hubfs/SAS-Top10-2018/PureSec – SAS Top 10-2018.pdf. (14 March 2024, date last accessed).
8. OWASP. OWASP Top 10 (2017) Interpretation for Serverless. https://owasp.org/www-pdf-archive/OWASP-Top-10-Serverless-Interpretation-en.pdf. (14 March 2024, date last accessed).
9. AWS. Security Overview of AWS Lambda. 2021. https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-overview-aws-lambda/security-overview-aws-lambda.pdf. (14 March 2024, date last accessed).
10. AWS. Serverless Applications Lens AWS Well-Architected. Framework. 2019. https://docs.aws.amazon.com/pdfs/wellarchitected/latest/serverless-applications-lens/wellarchitected-serverless-applications-lens.pdf#document-revisions. (14 March 2024, date last accessed).
11. Kelly D, Glavin FG, Barrett E. DoWTS–Denial-of-Wallet test simulator: synthetic data generation for preemptive defence. *J Intell Inf Syst* 2022;**60** 2, 1–24.
12. Claburn T. Not saying you should but we're told it's possible to land serverless app a '$40K/month bill using a 1,000-node botnet'. The Register, 2021. https://www.theregister.com/2021/04/21/denial_of_wallet/. (14 March 2024, date last accessed).
13. Mileski D, Mihajloska H. Distributed Denial of Wallet attack on serverless pay-as-you-go model. In: *2022 30th Telecommunications Forum (TELFOR)*. Belgrade, Serbia IEEE, 2022, 1–4.
14. Shen J, Zhang H, Geng Y. *et al.* Gringotts: fast and accurate internal denial-of-wallet detection for serverless computing. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Se-*

curity, CCS '22, New York, NY: Association for Computing Machinery, 2022, 2627–41.

15. Datta P, Kumar P, Morris T. *et al.* Valve: securing function workflows on serverless computing platforms. In: *Proceedings of The Web Conference 2020*. Taipei, Taiwan: ACM. 2020, 939–50.
16. OWASP. OWASP API Security Project. 2019. https://owasp.org/API-Security/editions/2019/en/0x00-header/. (14 March 2024, date last accessed).
17. Su JM, Cheng MH, Wang XJ. *et al.* 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)*. Bali, Indonesia IEEE, 2019, 306–9.
18. Maki N, Nakata R, Toyoda S. *et al.* An effective cybersecurity exercises platform CyExec and its training contents. *Int J Inform Educ Technol* 2020;**10**:215–21. https://doi.org/10.18178/ijiet.2020.10.3.1366.
19. Idris M, Syarif I, Winarno I. Development of vulnerable web application based on OWASP API security risks. In: *2021 International Electronics Symposium (IES)*. Surabaya, Indonesia IEEE, 2021, 190–4.
20. Wang S, Su Z. Image classification. 2019. https://paperswithcode.com/task/image-classification. (24 November 2022, date last accessed).
21. ImageNet. Imagenet Large Scale Visual Recognition Challenge (ILSVRC). https://image-net.org/challenges/LSVRC/index.php. (14 March 2024, date last accessed).
22. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv, 4 September 2014, preprint: not peer reviewed. https://doi.org/10.48550/arXiv.1409.1556.
23. He K, Zhang X, Ren S. *et al.* Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, IEEE: Boston, MA, USA 770–8.
24. Iandola FN, Han S, Moskewicz MW. *et al.* SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size, arXiv, 24 February 2016, preprint: not peer reviewed. https://doi.org/10.48550/arXiv.1602.07360.
25. Krizhevsky A. Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Commun ACM* 2017;**60**:84–90. https://doi.org/10.1145/3065386.
26. Chollet F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 1251–8. IEEE: Las Vegas, NV, USA.
27. Szegedy C, Vanhoucke V, Ioffe S. *et al.* Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, 2818–26. IEEE: Las Vegas, NV, USA.
28. Sandler M, Howard A, Zhu M. *et al.* Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 4510–20. IEEE: Salt Lake City, UT, USA.
29. Niyaz Q, Sun W, Javaid AY. A deep learning based DDoS detection system in software-defined networking (SDN), arXiv, 22 November 2016, preprint: not peer reviewed. https://doi.org/10.48550/arXiv.1611.07400.
30. He Z, Zhang T, Lee RB. Machine learning based DDoS attack detection from source side in cloud. In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. New York, NY, USA: IEEE, 2017, 114–20.
31. Priya SS, Sivaram M, Yuvaraj D. *et al.* Machine learning based DDoS detection. In: *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*. Pune, India: IEEE, 2020, 234–7.
32. Ko I, Chambers D, Barrett E. Feature dynamic deep learning approach for DDoS mitigation within the ISP domain. *Int J Inf Secur* 2020;**19**:53–70. https://doi.org/10.1007/s10207-019-00453-y.
33. Markus-Go. BoNeSi. https://github.com/Markus-Go/bonesi. (14 March 2024, date last accessed).
34. Nataraj L, Karthikeyan S, Jacob G. *et al.* Malware images: visualization and automatic classification. In: *Proceedings of the 8th International Symposium on Visualization for Cyber Security. VizSec '11*, New York, NY: Association for Computing Machinery, 2011.
35. Luo JS, Lo DCT. Binary malware image classification using machine learning with local binary pattern. In: *2017 IEEE International Conference on Big Data (Big Data)*. Boston, MA, USA: IEEE, 2017, 4664–7.
36. Freitas S, Duggal R, Chau DH. Malnet: a large-scale cybersecurity image database of malicious software, arXiv, 31 January 2021, preprint: not peer reviewed. https://doi.org/10.48550/arXiv.2102.01072.
37. Azab A, Khasawneh M. Msic: malware spectrogram image classification. *IEEE Access* 2020;**8**:102007–21. https://doi.org/10.1109/ACCESS.2020.2999320.
38. Wang W, Zhu M, Zeng X. *et al.* Malware traffic classification using convolutional neural network for representation learning. In: *2017 International Conference on Information Networking (ICOIN)*. Da Nang, Vietnam: IEEE, 2017, 712–7.
39. Taheri S, Salem M, Yuan JS. Leveraging image representation of network traffic data and transfer learning in botnet detection. *Big Data Cogn Comput* 2018;**2**, 7. https://doi.org/10.3390/bdcc2040037.
40. Garcia S, Grill M, Stiborek J. *et al.* An empirical comparison of botnet detection methods. *Comput Sec* 2014;**45**:100–23. https://doi.org/10.1016/j.cose.2014.05.011.
41. McCullough E, Iqbal R, Katangur A. Analysis of machine learning techniques for lightweight DDoS attack detection on IoT Networks. In: *International Conference on Forthcoming Networks and Sustainability in the IoT Era*. Online Springer, 2020, 96–110.
42. Kelly D, Glavin F, Barrett E. Serverless computing: behind the scenes of major platforms. In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. Online IEEE, 2020, 304–12.
43. Keras documentation: Keras applications. https://keras.io/api/applications/. (24 November 2022, date last accessed).
44. AWS. Wild Rydes. https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/. (14 March 2024, date last accessed).
45. Beswick J. Building a location-based, scalable, serverless web app. 2020. https://aws.amazon.com/blogs/compute/building-a-location-based-scalable-serverless-web-app-part-1/. (14 March 2024, date last accessed).
46. PYTS. Single gramian angular field. https://pyts.readthedocs.io/en/stable/auto_examples/image/plot_single_gaf.html. (14 March 2024, date last accessed).