Generative Deep Learning for Internet of Things Network Traffic Generation

Conference Paper · December 2020

DOI: 10.1109/PRDC50213.2020.00018

5 authors, including:

Mustafizur Rahman Shahid
Télécom SudParis - Institut Polytechnique de Paris
7 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Houda Jmila
Institut Mines-Télécom
15 PUBLICATIONS **68** CITATIONS
SEE PROFILE

Hervé Debar
Télécom SudParis - Institut Mines-Télécom
160 PUBLICATIONS **5,866** CITATIONS
SEE PROFILE

Generative Deep Learning for Internet of Things Network Traffic Generation

Mustafizur R. Shahid*, Gregory Blanc*, Houda Jmila*, Zonghua Zhang†, Hervé Debar*
*Télécom SudParis, Institut Polytechnique de Paris, France
{mustafizur.shahid, gregory.blanc, houda.jmila, herve.debar}@telecom-sudparis.eu
†IMT Lille-Douai, Institut Mines-Télécom, France
zonghua.zhang@imt-lille-douai.fr

***Abstract*—The rapid development of the Internet of Things (IoT) has prompted a recent interest into realistic IoT network traffic generation. Security practitioners need IoT network traffic data to develop and assess network-based intrusion detection systems (NIDS). Emulating realistic network traffic will avoid the costly physical deployment of thousands of smart devices. From an attacker's perspective, generating network traffic that mimics the legitimate behavior of a device can be useful to evade NIDS. As network traffic data consist of sequences of packets, the problem is similar to the generation of sequences of categorical data, like word by word text generation. Many solutions in the field of natural language processing have been proposed to adapt a Generative Adversarial Network (GAN) to generate sequences of categorical data. In this paper, we propose to combine an autoencoder with a GAN to generate sequences of packet sizes that correspond to bidirectional flows. First, the autoencoder is trained to learn a latent representation of the real sequences of packet sizes. A GAN is then trained on the latent space, to learn to generate latent vectors that can be decoded into realistic sequences. For experimental purposes, bidirectional flows produced by a Google Home Mini are used, and the autoencoder is combined with a Wassertein GAN. Comparison of different network characteristics shows that our proposed approach is able to generate sequences of packet sizes that behave closely to real bidirectional flows. We also show that the synthetic bidirectional flows are close enough to the real ones that they can fool anomaly detectors into labeling them as legitimate.**

***Index Terms*—Deep Learning, Generative Adversarial Network, Autoencoder, Network Security, Internet of Things**

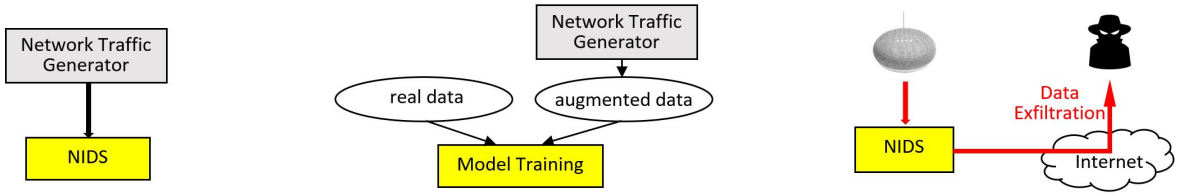
I. Introduction

The rapid adoption of the Internet of Things (IoT) raises new security concerns. Most smart devices are vulnerable because of a lack of security awareness from both the manufacturer and the end users. Most users never change the default password or use weak credentials making their device vulnerable to brute force attacks. When a vulnerability is discovered, it is seldom properly patched because of a poor update policy from the manufacturer. Since Mirai in 2016 [1], IoT malware have evolved and are getting more and more sophisticated.

To protect IoT networks, Network Intrusion Detection Systems (NIDS) specifically designed for IoT are being developed [2]. To evaluate the ability of an NIDS to correctly detect intrusions, both legitimate and malicious network traffics are required (see Figure 1.a). While malicious network traffic is used to evaluate attack detection rate, legitimate network traffic is necessary to assess the false positive rate. However, very few IoT network traffic datasets are publicly available. One solution is to physically deploy real IoT devices to produce network traffic data. Yet, this can become very costly if one needs a network with thousands of smart devices. Moreover, due to privacy concerns, it might not be possible to share real network traffic data. An alternative is to synthetically generate IoT network traffic. Synthetic network traffic generation can also be useful for data augmentation purposes (see Figure 1.b). Using data augmentation, a machine learning model can be trained on both real and synthetic network traffic allowing a faster convergence of the model and a better achieved performance.

Such synthetic traffic can also be used for malicious purposes. Rigaki and Garcia [3] have explored the special case of data exfiltration. In this context, an attacker, who has the ability to sniff the network, may collect legitimate traffic. The attacker then proceeds to train generative models from the collected traffic, so as to learn how to generate network traffic resembling the real, legitimate one. In order to deceive an NIDS, the trained model may be used to initiate network communications that mimic the legitimate behavior of the infected device (also known as *mimicry attack*). For example, as illustrated in Figure 1.c, the microphone of a compromised Google Home Mini can be used to spy on and listen to conversations. However, the network being protected by an NIDS, the attacker needs to find a way to generate network traffic that looks legitimate in order to circumvent detection, and exfiltrate sensitive data. Indeed, contrary to general-purpose computers, IoT devices perform very specific tasks, hence their networking behavior is very stable and follows specific patterns. Any network communication that does not follow the legitimate behavioral pattern can be easily pinpointed as being anomalous.

Network traffic consists of flows identified by source and destination IP addresses and ports, and the transport layer protocol. We represent bidirectional flows as a sequence of packets sent and received between an internal source and an external destination. Despite the fact that a flow and the packets composing it are closely related, existing works either focus on the generation of flow-level features or packet-level



(a) NIDS performance assessment

(b) Model training

(c) Data exfiltration

Fig. 1: Motivation for IoT network traffic generation: Scenario (a): realistic network traffic generation for NIDS performance evaluation; Scenario (b): Data augmentation to train machine learning based NIDS; Scenario (c): mimicry attack generation for data exfiltration purpose.

features, but not both at the same time.

In this paper, we aim at generating synthetic sequences of packet sizes that correspond to bidirectional flows that look like they were generated by a real IoT device. In addition to generating packet-level features which are the sizes of individual packets, our developed generator implicitly learns to comply with flow-level characteristics, such as the total number of packets and bytes in a bidirectional flow. Similarly to text data, sequences of packet sizes are sequences of categorical data. Hence, our problem is analogous to word by word text generation, a domain in which many solutions have been proposed to adapt a Generative Adversarial Network (GAN) to generate sequences of categorical data [4]–[6]. Inspired by the work proposed in [6], we propose to generate sequences of packet sizes by combining an autoencoder with a Wasserstein GAN (WGAN). First, the autoencoder is trained to learn a latent representation of the real sequences of packet sizes. A WGAN is then trained on the latent space, to learn to generate latent vectors that can be decoded into realistic sequences. For the experimental part of our study, we use network traffic data produced by a Google Home Mini. Comparing the distributions of different network characteristics, such as the number of packets and bytes per flow, shows that the generated bidirectional flows exhibit behavioral patterns that are very similar to the real traffic.

The rest of the paper is organized as follows: Section II presents the related work; in Section III, we formally define a sequence of packet sizes that correspond to a bidirectional flow; then, in Section IV we present our proposed generative model; Section V describes the experimental settings and Section VI presents the obtained results; finally, Section VII discusses some of the limitations of the proposed approach and Section VIII concludes and presents possible future works.

II. Related Work

Network traffic generation has long been addressed through the use of very simple statistical models. Hence, the very first work focusing on the generation of legitimate network traffic is from J. Sommers et al. [7], in which, flow characteristics are collected from standard NetFlow [8] or packet traces. Then, the empirical distributions of each parameter are derived and used to generate new flows. Interdependencies between different parameters are not taken into account. The generation of network traffic using generative deep learning models is an emerging topic, thus very few related work exist. Some of them attempt to generate *network flows* while others propose to generate *individual network packets*. A network flow is identified by its i) source IP address ii) destination IP address iii) source port iv) destination port, and v) the transport layer protocol. The features that characterize *a flow* are usually the duration, the total number of packets sent and the total number of bytes sent. The features used to describe *a packet* are the different fields of the network layer (e.g., *IP*) or the transport layer (e.g., *TCP* or *UDP*) headers. Approaches addressing flow generation only use flow-level features and do not attempt to characterize the individual packets that constitute the flow. On the other hand, work focusing on the generation of individual packets ignore the flow-level features.

Recently, Ring et al. [9] use GANs to generate flow-based network traffic with all typical attributes. Different from most existing works on flow generation, they consider IP addresses and ports as features to be generated. Other works focus on the generation of flows for malicious purpose. For example, Rigaki et al. [3] propose to evade malware detection by generating flows that mimic the behavior of a legitimate Facebook chat application using a GAN. The generated features include the total number of bytes in the flow, the duration of the flow and the time between two successive flows. Yan et al. [10] leverage the capabilities of GANs to generate ambivalent traffic: DDoS attacks that mimic legitimate behavior. Features are categorized between mutable and immutable, the latter ones are mostly related to the DoS function and are not to be generated by the GAN, for fear it will lose its traffic function and hence its attack ability. Similarly, Charlier et al. [11] focus on the use of GANs to generate DDoS attack flows. In [12], Wu et al. describe a deep reinforcement learning (DRL) based framework to generate adversarial flows to deceive an intrusion detection model. The DRL agent acts by adding perturbations to malicious flows. The limitation of existing work on flow generation is that they do not attempt to characterize individual packets that compose the flow. They are limited to the generation of flow-level features, such as, the total number of packets and bytes contained in the flow. They do not consider more fine-grained features, such as the size of the individual packets composing a flow. Traffic generation

based only on flow-level features is not realistic enough and will fail to fool network monitoring tools that perform packet-level analysis.

Other studies work at the packet level. They intend to generate valid network packets using GANs. In [13], Z. Lin et al. describe preliminary works on how to use GANs for oblivious network analysis, that is, to learn the internal structures and protocol formats of black-box devices. To this purpose, they develop synthetic protocols to show that a GAN is able to learn intra-field and inter-field dependencies, ultimately generating compliant packets for a black-box protocol. They also discuss how GANs can be used to generate malicious packets. A. Cheng proposes to generate individual IP packets using convolutional neural networks (CNN) and GANs [14]. The aim of the GAN is to generate IP packets with compliant header values from scratch. To this purpose, the raw byte-wise representation of an IP packet is mapped to a pixel-wise image-like representation. Then, a CNN-based GAN is trained to learn to generate valid IP packets. One different CNN-GAN is trained for each specific packet type (e.g., ICMP ping, DNS query or HTTP GET request packets). H. Nguyen-An et al. [15] propose a very basic packet-level network traffic generator for the IoT. They do not use any algorithm to learn patterns from the training set. Instead they only compute the average packet size and the average time periodicity between the packets. Then, they simply generate a stream of packets of constant size (equal to the average packet size) and with a constant time periodicity (equal to the average period). Works focusing on packets generation treat packets individually and independently from each other, failing to capture the sequential nature of network communication.

To the best of our knowledge, all existing works either focus either on flow-level or packet-level traffic generation, but not a combination of both. However, we believe that the resulting traffic is incomplete since a flow and the packets composing it are closely linked. For example, the number of bytes exchanged for the duration of a flow usually amounts to the sum of the sizes of each packet that composes the flow.

III. Sequence of Packet Sizes

The packet size is a widely used feature in studies that focus on the development of IoT device network monitoring systems [16]–[21]. Hence, it is an important network traffic feature for both security product developers and malware authors that want to mimic the legitimate behavior of a device. However, note that the solution proposed in this paper can be used to generate any feature that can be represented as a sequence of categorical elements, like TCP flags for example. We propose to generate sequences that represent the size of the individual packets composing a bidirectional flow. Hence, besides learning to generate packet-level features which are the sizes of individual packets, our developed generator implicitly learn to comply with flow-level characteristics, such as the ordering of the packets or the total number of packets and bytes in a bidirectional flow. To the best of our knowledge, this is the first attempt at bridging the gap between flow generation and packet generation problems. Moreover, none of the existing works on artificial network traffic generation have focused on the specific context of IoT as we do.

We aim to generate sequences of packet sizes, representing bidirectional flows, that look like they were generated by a real IoT device. A bidirectional flow is identified by a 5-tuple, the source and destination IP addresses and ports, and the transport layer protocol. A sequence contains the sizes of packets sent as well as the sizes of packets received during a single communication. A communication is a complete TCP session (from SYN to FIN). Note that a timeout is used to split long communications into multiple bidirectional flows. Let N be the maximum number of packets that can be sent (or received) in a single bidirectional flow. That is, a bidirectional flow can contain a maximum of $2 \times N$ packets (N packets sent and N packets received). It is important to note that N is device- or application-specific. We have witnessed that some devices never generate flows with more than a certain number of packets. For experimental purposes, one might set N and truncate sequences that are too long. This might be useful to control the cost of the training process, in terms of resources, which increases as N grows. Let P be a sequence of packet sizes corresponding to a bidirectional flow, let s_i be the size of the i^{th} packet sent and r_i the size of the i^{th} packet received. P can therefore be defined as follows:

$$P = \{s_1, r_1, s_2, r_2, ..., s_N, r_N\}$$

If a bidirectional flow contains less than N packets sent then the remaining elements of the sequence are filled with zeros (the same is true for the packets received). Hence, zero acts as an end of sequence marker. It is important to notice that we consider P as a sequence of categorical data rather than numerical data. For example, let D be a device which generates sequences of packet sizes of the following form:

```
60 60 52 52 123 135 52 52 52
```

Considering the dataset of collected sequences, we may notice that the device D never generates packets of size 61, 53 or 50. If, for once, D generates the following (approximate) sequence:

```
60 61 52 53 122 135 52 50 52
```

we may decide that it cannot have been generated by D as it contains packet sizes that D never generated in its history, namely 61, 53, and 50. The approximate sequence is still very similar to the original one (erroneous packet sizes are close to legitimate ones, if considered as numerical values). To avoid ending up generating such close but unrealistic sequences, one needs to consider the packet size as a categorical variable. Hence, we use one-hot-encoding over all the possible packet size values to represent each element of P.

Note that our problem is similar to word by word text data generation in that a bidirectional flow is equivalent to a sentence (with $2 \times N$ being the maximum length of the sentence), while packet sizes, as categorical data, are equivalent to the words that compose the sentence.

IV. Generative Models for Sequence Generation

In this Section, we firstly introduce GANs and explain their limitations when dealing with sequences of categorical data. Then, we describe our proposed model that combine an autoencoder with a GAN.

A. On the Difficulty of Generating Sequences of Categorical Data

GANs were introduced by I. Goodfellow et al. [22] and have been successfully applied in computer vision to generate realistic images [23]. A GAN consists of *a generator* and *a discriminator*. The role of the *generator* is to generate observations as similar as possible to the samples present in a given dataset. To this purpose, the generator transforms random noise into samples that look as if they have been drawn from the original dataset. The role of the *discriminator* is to predict whether a given sample comes from the original dataset or has been generated by the generator. Both, the discriminator and generator are neural networks. At the beginning of the training process, their weights are randomly initialized. The GAN is trained by alternatively training the generator and the discriminator. As the generator begins to fool the discriminator, the discriminator must learn new patterns to differentiate real samples from generated ones. In turn, the generator needs to find new ways to fool an ever improving discriminator. This cycle continues up to the point the generator starts generating samples that the discriminator cannot discriminate from real samples anymore. However, GANs are very hard to train and are prone to *mode collapse*. Mode collapse occurs when the generator starts generating one or a small set of possible observations that always fool the discriminator [24]. In that case, the generator stops learning anything useful. It raises the issue of how representative the generated samples are of the diversity of the original dataset. The Wasserstein GAN (WGAN) [25] improves traditional GANs. It provides more stable training process and gets rid of mode collapse issues. This motivates us to use WGAN for our experiments. The loss function of a WGAN is the Wasserstein loss, given by:

$$-\frac{1}{m}\sum_{i=1}^{m} y_i p_i$$

where m is the total number of training instances, y_i and p_i are respectively the label, and the prediction of the *critic* (the discriminator of a WGAN is called the critic), corresponding to the i^{th} training instance. The label y_i is either equal to 1 (real) or -1 (generated), and the prediction p_i is in the range $[-\infty, +\infty]$. Hence, by minimizing the loss function, the critic of a WGAN tries to maximize the difference between its predictions for real samples and generated samples. Without any additional constraint, the Wassestein loss can be very large and become intractable. This is why the critic of a WGAN must be a 1-Lipschitz continuous function. In the original paper, the 1-Lipschitz constraint is enforced by clipping the weight of the critic. In [26], the authors proposed to enforce the 1-Lipschitz constraint by penalizing the norm of the gradient of the critic with respect to its input, which is a more natural way to achieve the 1-Lipschitz constraint. In this paper, we will explore both WGAN with weight clipping (WGAN-C) and WGAN with gradient penalty (WGAN-GP).

Although successful for image generation [27], GANs have known little success with sequence of categorical data generation until recently. Indeed, when generating the next element of a sequence of categorical values, the generator actually provides a probability distribution over all possibilities (e.g., the vocabulary for text data). The actual sequence is constructed by picking the next element from this probability distribution. This picking operation is hard to back-propagate as it is not differentiable [4], [5]. To overcome this issue, many solutions have been proposed in the context of text generation. Kusner et al. [4] propose to use the Gumbel-softmax distribution as the output of the generator. Yu et al. [5] describe how Reinforcement Learning can be used to bypass the issue. The model we use to generate sequences of packet sizes is highly inspired from the work of D. Donahue et al. [6] who propose to combine a vanilla autoencoder with a GAN to generate text data. First, the autoencoder is trained to learn to convert sequences of categorical data (sentences composed of words) into a latent vector in a continuous space. Then a GAN is trained in the continuous latent space to learn to generate latent vectors that can be decoded into sequences of categorical data.

B. Combining Autoencoder with GAN

To generate sequences of packet sizes, we propose to combine an autoencoder with a GAN as described in Figure 2.

An autoencoder is composed of an encoder and a decoder. The encoder compresses the input to obtain a latent representation of it. The role of the decoder is to reconstruct the original input from its latent representation. Hence, the training process aims at minimizing the reconstruction error between the input and the output. For our sequence of one-hot encoded packet size data, this corresponds to minimizing the cross-entropy loss between the input and the output of the autoencoder. Let $L = 2 \times N$ (N defined in Section III) be the total length of a sequence of packet sizes, and V be the vocabulary size, that is, the number of possible values that a packet size can take. Hence, the one-hot encoded representation of a single packet size is a vector of length V. A sequence of one-hot encoded packet sizes can be represented with a matrix X:

$$X = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1L} \\ x_{21} & x_{22} & ... & x_{2L} \\ ... & ... & & ... \\ x_{V1} & x_{V2} & ... & x_{VL} \end{bmatrix}$$

where the i^{th} column of X corresponds to the one-hot encoded representation of the i^{th} packet size in the sequence. Let X be the matrix representation of the input of the autoencoder, and $\hat{X}$ be the equivalent matrix representation of the output of the autoencoder. Then, the cross-entropy loss L_{CE} between X and $\hat{X}$ is given by summing the binary cross-entropies between every single element of X and $\hat{X}$:

$$L_{CE} = \sum_{j=1}^{L}\sum_{i=1}^{V}(x_{ij} log(\hat{x}_{ij}) + (1 - x_{ij}) log(1 - \hat{x}_{ij}))$$

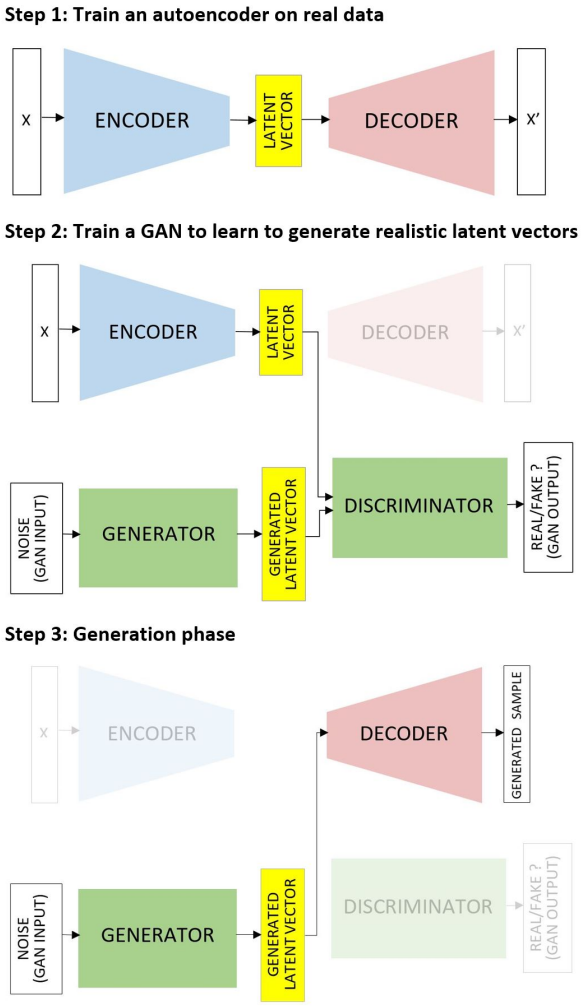


Fig. 2: Combining an autoencoder with a GAN to generate sequences of categorical values

Once the autoencoder has been trained, a GAN is trained on the latent space. For our model, we will use a WGAN (both WGAN-C and WGAN-GP will be tested) as it provides more stability in the training stage and is more resilient to mode collapse [25]. Figure 2 shows the different steps from the training phase to the generation phase. First, an autoencoder is trained to learn to compress and reconstruct real sequences of packet sizes. The encoder part of the autoencoder is then used to obtain the continuous latent representation of the real sequences. Next, a WGAN is trained on the continuous latent space to learn to generate realistic latent vectors. Once trained, the generator of the WGAN is used to generate real-looking latent vectors. The generated latent vectors are then fed to the decoder of the autoencoder to generate realistic sequences of packet sizes that correspond to bidirectional flows.

V. Experiments

In this Section we first define the different methods used to assess the quality of the generated bidirectional flows. Then, we describe the experimental settings, that is, the dataset used and the architecture of the autoencoder and WGAN.

A. Evaluation Methodology

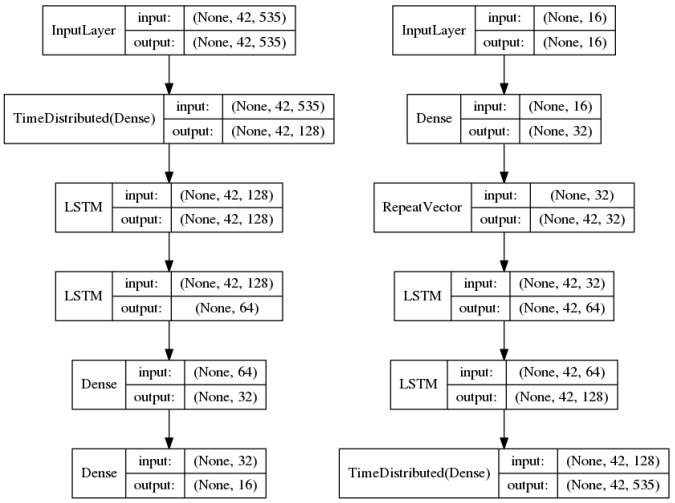
The quality of the generated bidirectional flows is assessed using different methods. First, we measure the percentage of generated sequences that are valid. Then, we plot histograms to compare the empirical distributions of different network characteristics of the synthetic bidirectional flows with the real ones. The characteristics that are compared are i) the distribution of the packets sizes, ii) the distribution of the number of packets per bidirectional flow, and iii) the distribution of the number of bytes per bidirectional flow. Those network characteristics are widely used to describe network flows [3], [9], [10], [12], [28]. The purpose of comparing the distributions of different network characteristics is to determine if the generated bidirectional flows are diverse enough (no mode collapse) and if they behave like the real ones. Finally, we also assess the proportion of synthetic bidirectional flows that can evade a potential anomaly detection based NIDS. To this purpose, we train different anomaly detection algorithms to learn the legitimate networking behavior profile of a Google Home Mini. Synthetic bidirectional flows are then fed to the trained anomaly detectors to determine the proportion of synthetic flows that is able to fool the trained detectors into labeling them as legitimate.

B. Experimental Setup

A Google Home Mini was used to produce real network traffic data. The device was actively used for 7 days. We set the value of N, described in Section III, to 21. That is, we only keep bidirectional flows that contain at most 21 packets sent and 21 packets received, which correspond to sequences of length $L = 42$. As explained in Section III, if a bidirectional flow contains less than 21 sent packets, the remaining elements are filled with zeros – the same is applied to received packets. The following is an example of a sequence of packet sizes (after zero padding), where 6 packets have been sent and 4 packets have been received:

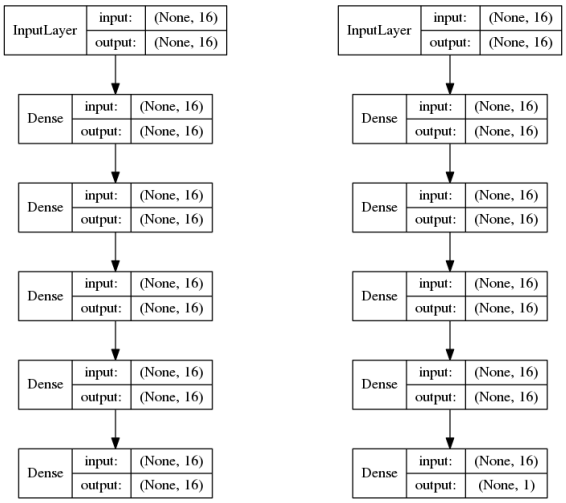
```
60 60 52 52 123 135 52 52 52 0 52 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
```

The reason we only keep sequences of maximum length 42 is because they correspond to 90% of the bidirectional flows produced by our hosted Google Home Mini. The remaining 10% of the bidirectional flows contain a number of packets exchanged ranging from 43 to 7674 packets. Meaning that if we were to represent all the bidirectional flows of the Google Home Mini, we would need sequences of length 7674, which can be computationally expensive. For this reason, limiting sequences to 42 packet sizes is reasonable. Moreover, in the case of a malware that want to mimic legitimate behavior, it makes more sense to focus on the 90% most common flows rather than on the 10% rarest flows. The final dataset contains 12,198 sequences of packet sizes. The total number of possible



(a) Encoder (b) Decoder

Fig. 3: Architecture of the autoencoder used for the experiment



(a) Generator (b) Critic

Fig. 4: Architecture of the WGAN used for the experiment

packet size values is $V = 535$ (the vocabulary size). Hence, each element of the sequence is one-hot encoded in a vector of size 535. Hence, the shape of the corresponding matrix X (defined in Section IV-B) describing one sequence of packet sizes is $(V, L) = (535, 42)$.

The architecture used for the autoencoder is given in Figure 3. The sequences of shape (535, 42) are compressed to 16-dimensional latent vectors by the encoder. The encoder is composed of a time distributed dense layer, followed by two LSTM layers and two densely connected layers. An LSTM network is a type of recurrent neural network that can capture temporal dependencies in sequential data. Hence, LSTM layers are well suited to capture the ordering of the packet sizes in a sequence. The activation function used in all layers of the encoder is the hyperbolic tangent (`tanh`). The decoder consists of a dense layer, followed by two LSTM layers and one time distributed dense layer. The activation function used for all layers is also `tanh`, except for the output layer for which a `sigmoid` activation is used. Note that for dense layers to avoid vanishing gradient issues that occur when using `tanh` and `sigmoid` activation functions, Batch Normalization [29] is performed before applying the activation function. The loss function used is the cross-entropy loss, as described in Section IV-B. The autoencoder is trained for 300 epochs using `Adam` optimizer. The learning rate is set to 0.001 for the first 100 epochs, then to 0.0005 for another 100 epochs, finally to 0.0001 for the last 100 epochs.

As for the WGAN used to learn to generate latent vectors that decode into realistic features vector, both the generator and the critic are densely connected neural networks as shown in Figure 4. The generator takes as input a noise of dimension 16 drawn from a standard Gaussian distribution. It consists of 4 hidden layers, each composed of 16 units with `tanh` activation. As with the autoencoder, to avoid vanishing gradient issues, Batch Normalization is performed before applying `tanh` activation function. The output layer of the generator is a densely connected layer with 16 units and no activation function. The critic consists of 4 hidden layers. Each composed of 16 units with `LeakyReLU` (alpha=0.01) activation. The output of the critic is a single neuron with no activation. As proposed in [25], `RMSprop` optimizer, with learning rate 0.00005, is used for training. The critic is trained 5 times between each generator updates. Both, WGAN-C and WGAN-GP are tested. As explained in Section IV-A they only differ in the way the 1-Lipschitz constraint of the critic is enforced.

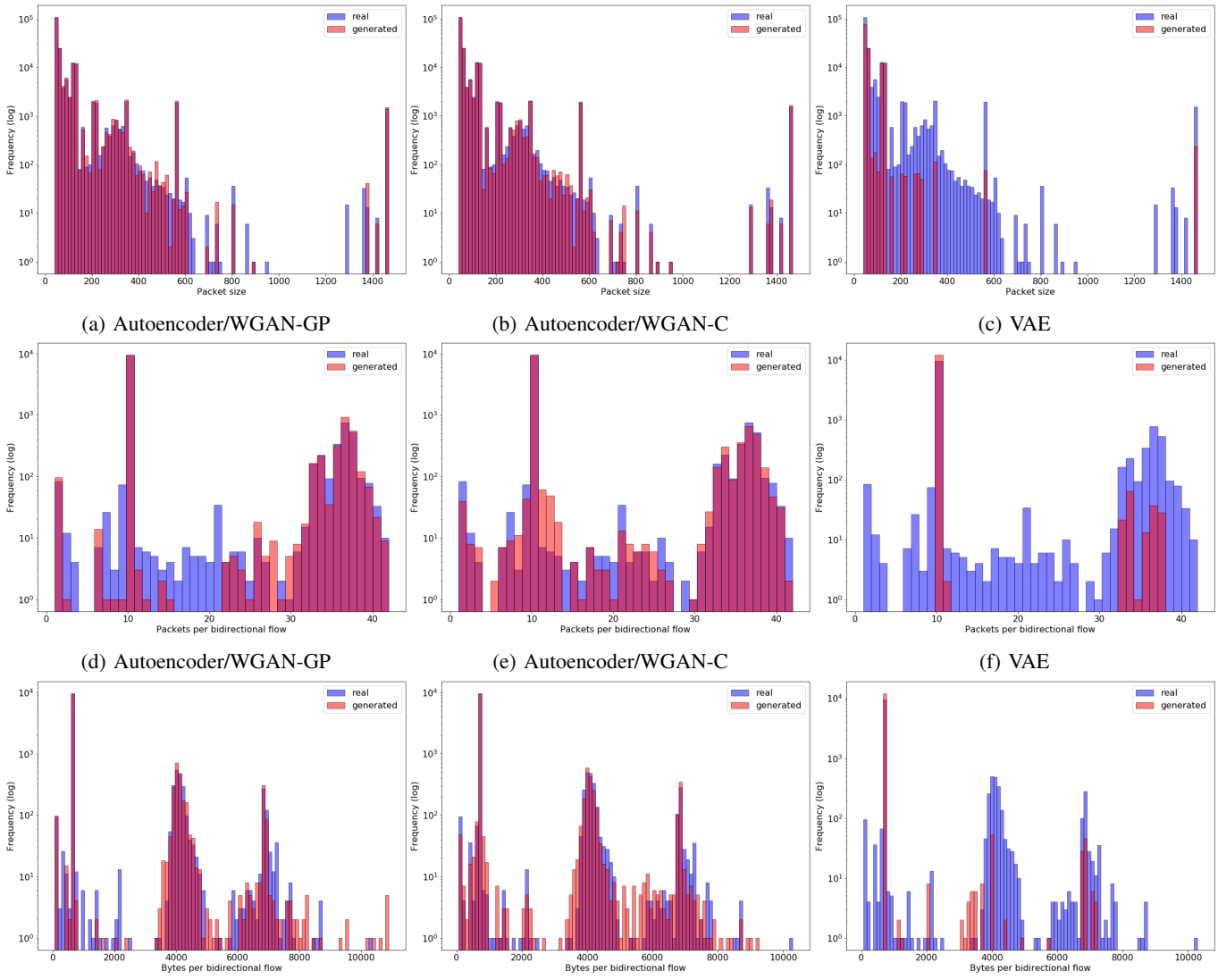
VI. Results

A. Percentage of Valid Sequences

The generated sequences are of length 42 and consist of the size of the packets sent and received. As described in Section III, if a sequence has less than 21 packets sent thenthe remaining elements must be zeros. As a consequence, if the size of a sent packet in the sequence is equal to zero then all the subsequent elements of the sequence that correspond to sent packets must also be equal to zero. The generated sequences that do not comply with this basic rule are considered invalid. The same reasoning holds for the elements of the sequence that corresponds to the size of packets received. Our trained autoencoder/WGAN-GP and autoencoder/WGAN-C models generate 99.5% and 99.1% of valid sequences respectively. For the evaluation performed in the next subsections, invalid sequences are discarded.

B. Statistical Characteristics Comparison

The distributions of different network characteristics of the generated and real bidirectional flows are compared for both autoencoder/WGAN-GP and autoencoder/WGAN-C models. The obtained results are also compared to the ones obtained by a baseline variational autoencoder (VAE) model. A VAE is a special type of autoencoder that can be used as a generative



(a) Autoencoder/WGAN-GP (b) Autoencoder/WGAN-C (c) VAE

(d) Autoencoder/WGAN-GP (e) Autoencoder/WGAN-C (f) VAE

(g) Autoencoder/WGAN-GP (h) Autoencoder/WGAN-C (i) VAE

Fig. 5: Comparison of the distribution of packet sizes (a, b, c), the number of packets per bidirectional flow (d, e, f), the number of bytes per bidirectional flows (g, h, i) for different models (autoencoder/WGAN-GP, autoencoder/WGAN-C, VAE)

model [30]. The architecture of the VAE used is very similar to the architecture of the autoencoder described in Figure 3. The only difference is that, for the VAE the encoder outputs two parameters: a mean vector and a variance vector. Those two parameters are used to sample the latent vector. Once trained, to generate new instances, latent vectors are sampled from the standard normal distribution and fed to the decoder of the VAE. The aim is to determine if combining an autoencoder with a WGAN is necessary, or if a VAE model is enough to generate realistic data.

The 12,198 real bidirectional flows produced by the Google Home Mini along with 12,198 generated bidirectional flows are used to plot each histogram. Figure 5 shows the obtained distributions of packets sizes, number of packets per bidirectional flow and the number of bytes per bidirectional flow. The frequency is represented in a logarithmic scale, as the number of occurrences of rare events is very small compared to the occurrences of common events. The distribution plots indicate that the bidirectional flows generated by both autoencoder/WGAN-GP and autoencoder/WGAN-C models share very close characteristics with the real ones. The similarity is even more emphasized for occurrences that are very common in the real traffic (more than 10 times). For example, packet sizes between 40 bytes and 600 bytes, bidirectional flows containing between 32 and 40 packets, or bidirectional flows containing around 4000 bytes, are very common in the real traffic and are also very common in the generated traffic. Both the WGAN-GP and the WGAN-C based models outperform the VAE which suffers from severe mode collapse. Indeed, the flows generated by the VAE lack diversity and do not cover all the possible flow types that

	WGAN-GP	WGAN-C	VAE
Packet sizes	1.694	**1.539**	27.439
Packets per bidirectional flows	10.228	**9.799**	97.362
Bytes per bidirectional flows	4.673	**3.837**	40.597

TABLE I: Earth mover's distance (10^{-4}) between the real and generated traffic histograms of Figure 5. WGAN-C based model achieves the smallest distance.

the Google Home Mini produces. In terms of the diversity of the generated bidirectional flows, the WGAN-C based model seems to perform better than the WGAN-GP based model. For example, the WGAN-C based model generates bidirectional flows containing between 16 and 21 packets, which is not the case for the WGAN-GP based model (see Figures 5.d and 5.e).

We use the Earth Mover's Distance (EMD, also referred as the first Wasserstein distance) to compare the histograms [31]: informally, if two distributions are seen as two masses of earth, the EMD between those two distributions is proportional to the minimum amount of work required to transform one distribution into the other (one unit of work is the amount of work necessary to move one unit of weight by one unit of distance). The EMD is a statistical distance that provides a measure to quantify the dissimilarity between the histograms of the generated traffic and the histograms of the real traffic. The smaller the EMD between the generated and real traffic histograms the closer the generated traffic is to the real one. As the EMD is used to compare probability distributions, the histograms are normalized to have a total area equal to 1. Table I shows the EMD for the histograms in Figure 5. The WGAN-C based model achieves the smallest EMD for every compared network characteristics. Hence, it is performing slightly better than the WGAN-GP based model in generating sequences of packet sizes that behaves closely to the real bidirectional flows.

C. Evading an Anomaly Detection Based NIDS

In this subsection, the aim is to assess how our proposed generative model can be used by a malware to mimic legitimate behavior and evade an anomaly detection based NIDS. In anomaly detection, during the training phase, the model learns the profile of the legitimate networking behavior of a device. Then during the testing phase, the model is applied to new data to detect any deviation from the learnt legitimate behavior profile. We train different anomaly detection algorithms on real Google Home Mini network data to learn the legitimate behavior profile. The trained anomaly detectors are tested against legitimate traffic to assess their False Positive Rate (FPR) and against malicious traffic to assess their True Positive Rate (TPR) (also referred as the recall or the attack detection rate). Malicious network traffic is obtained from IoTPOT [32], an IoT honeypot designed to be infected by IoT malware. The trained anomaly detectors are also tested against synthetically generated bidirectional flows to evaluate the proportion of synthetic flows that can evade them.

	OCSVM	IForest	EE
TPR	.9504	.8947	.9234
FPR	.0238	.0217	.1246

TABLE II: TPR and FPR on the test set achieved by the trained anomaly detectors

Three anomaly detection algorithms are tested: One-Class SVM (OCSVM), Isolation Forest (IForest), and Elliptic Envelope (EE). The features used as input for the anomaly detectors are the normalized packet sizes. As for the datasets, 80% of the 12,198 real Google Home Mini bidirectional flows are used for training and 20% for testing. 2440 malicious bidirectional flows from IoTPOT are used during the testing phase to assess the TPR of the trained detectors.

Table II presents the performance on the test set achieved by the different anomaly detectors. In terms of the attack detection rate OCSVM seems to perform the best with an TPR of 95.04%. While in terms of the FPR, IForest performs the best with an FPR of 2.17%. EE yields the worst FPR (12.46%).

Table III shows the False Negative Rate (FNR) on the synthetic bidirectional flows, denoted $FNR_{synthetic}$, which corresponds to the proportion of synthetic bidirectional flows that are predicted as being legitimate despite the fact that those flows are not coming from the Google Home Mini but potentially from a malware. It is compared to the FNR and the True Negative Rate (TNR) on the test set containing real bidirectional flows. The FNR on the test set (FNR_{test}) is the proportion of malicious flows that are incorrectly predicted as being legitimate. While the TNR on the test set corresponds to the proportion of bidirectional flows actually coming from the Google Home Mini that are correctly labeled by the anomaly detector as legitimate.

The $FNR_{synthetic}$ is very high compared to the FNR_{test} meaning that if a malware was to use our trained generative model to mimic the legitimate networking behavior, it would considerably improve its evasion capability and the malware would be able to evade the anomaly detectors most of the time. In fact, the $FNR_{synthetic}$ indicates that from 89.48% to 98.86% of the synthetic bidirectional flows (depending on the type of anomaly detector and the WGAN type used for the generator training) are able to fool the anomaly detectors into labeling them as legitimate. While without the use of a generative model the FNR_{test} indicates that only 4.96% to 10.53% of the malicious flows are incorrectly predicted as being legitimate. The $FNR_{synthetic}$ is also slightly higher than the TNR in most of the cases indicating that a synthetic bidirectional flow is more likely to be labeled as legitimate by the anomaly detectors than a real flow actually coming from a Google Home Mini. This can be explained by the presence of bidirectional flows coming from the Google Home Mini that are very rare and end up being wrongly labeled as being

	OCSVM	IForest	EE
$FNR_{synthetic}$ (WGAN-GP)	.9817	.9833	.9047
$FNR_{synthetic}$ (WGAN-C)	.9798	.9886	.8948
FNR_{test}	.0496	.1053	.0766
TNR	.9762	.9783	.8754

TABLE III: FNR when the anomaly detectors are fed with synthetic flows ($FNR_{synthetic}$) compared to the FNR and TNR on the test set

malicious (rare instances that appear in the test set but were not seen during anomaly detector training). While the generative model tends to generate the rarest bidirectional flow types less often and give priority to the frequent ones.

VII. Discussion

Mimicking the legitimate networking behavior of a device is interesting for data exfiltration purposes. For example, one can imagine a malware intended for Google Home Minis that uses the microphone of the compromised device to listen to conversations and exfiltrate the data. However for other types of malware, such as botnets used to perform large-scale DDoS attacks, complying with the legitimate networking behavior might be too much of a constraint. In an attempt to comply with the legitimate behavior, the malware might end up losing its malicious capability altogether. Further studies need to be carried out on how to find a balance between complying with legitimate behavior and not loosing malicious capabilities.

The types of sequences of packet sizes generated by the trained generator are very dependent on the data used during training. The generator will only be able to generate sequences similar to the sequences it has seen during the training phase. For our experimental setup we interacted with the Google Home Mini primarily to ask questions like "what's the weather today?" or "what's the news today?". Hence, our trained generator will generate sequences of packets that are representative of our interactions with the Google Home Mini like asking questions. Now, If someone makes a very different use of the Google Home Mini, like asking it to play music via a Spotify account, then the Google Home Mini might produce very different types of packet sequences. For the generator to be able to generate these new types of sequences, it will need to be retrained on the new data.

The generated bidirectional flows can be seen as two unidirectional flows: one unidirectional flow for the packets sent and the other for the packets received. The order of the packet sizes for each unidirectional flow is correct. However, the ordering of packet sizes for the bidirectional flow might not be correct. Indeed, in Section III a sequence of packet sizes P is defined by:

$$P = \{s_1, r_1, s_2, r_2, ..., s_N, r_N\}$$

where s_i is the size of the i^{th} packet sent and r_i the size of the i^{th} packet received. Hence, we assume that one packet sent is followed by one packet received, which is not necessarily the case. For example, one packet sent might trigger the reception of multiple packets and vice versa. Further studies are required to properly model the packet sizes ordering in a bidirectional flow.

VIII. Conclusion and Future Works

IoT network traffic generation is of importance for both the network defenders and the attackers. While the defenders would like to be able to develop and test NIDS without the costly physical deployment of real smart devices, the aim of the attackers is to be able to generate traffic that mimic the legitimate behavior of a device in order to evade NIDS. In this paper, we presented a method to generate sequences of packet sizes representing bidirectional flows that look as if they were generated by a real smart device. To overcome the issue with the use of GANs for sequence of categorical data generation, we decided to combine an autoencoder with a WGAN. First, the autoencoder is trained to learn to convert sequences of packet sizes (sequences of categorical data) into a latent vector in a continuous space. Then a WGAN is trained on the latent space to learn to generate latent vectors that can be decoded into realistic sequences, through the decoder of the autoencoder. Experimental results using a Google Home Mini show that our method allows us to generate high quality and realistic looking sequences of packet sizes representing bidirectional flows.

For future works, we should train and test our model on more data and with a variety of smart devices including security cameras. We are also planning to model packets ordering to better take into account cases when multiple packets are received for one packet sent or vice versa. We are also considering to include other packet features such as the status of the TCP flags or the TTL value of each packet in the sequence. Moreover, the inter-arrival times between packets in the sequence, as well as the total duration of the generated bidirectional flows are also important characteristics that need to be considered.

IX. Acknowledgement

The authors are partially funded under the VARIoT project with TENtec n. 28263632, co-financed by the Connecting Europe Facility of the European Union, and under the SPARTA project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 830892. The dataset used in this project has been generated by a platform built for VARIoT.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017.

[2] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2671–2701, 2019.

[3] M. Rigaki and S. Garcia, "Bringing a gan to a knife-fight: Adapting malware communication to avoid detection," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018.

[4] M. J. Kusner and J. M. Hernández-Lobato, "Gans for sequences of discrete elements with the gumbel-softmax distribution," *arXiv preprint arXiv:1611.04051*, 2016.

[5] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[6] D. Donahue and A. Rumshisky, "Adversarial text generation without reinforcement learning," *arXiv preprint arXiv:1810.06640*, 2018.

[7] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. Association for Computing Machinery, 2004.

[8] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "Rfc 3954: Cisco systems netflow services export version 9," *IETF http://www. ietf. org/rfc/rfc3954. txt*, 2004.

[9] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, 2019.

[10] Q. Yan, M. Wang, W. Huang, X. Luo, and F. R. Yu, "Automatically synthesizing dos attack traces using generative adversarial networks," *International Journal of Machine Learning and Cybernetics*, 2019.

[11] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "Syngan: Towards generating synthetic network attacks using gans," *arXiv preprint arXiv:1908.09899*, 2019.

[12] D. Wu, B. Fang, J. Wang, Q. Liu, and X. Cui, "Evading machine learning botnet detection models via deep reinforcement learning," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019.

[13] Z. Lin, S.-J. Moon, C. M. Zarate, R. Mulagalapalli, S. Kulandaivel, G. Fanti, and V. Sekar, "Towards oblivious network analysis using generative adversarial networks," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. Association for Computing Machinery, 2019.

[14] A. Cheng, "Pac-gan: Packet generation of network traffic using generative adversarial networks," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2019.

[15] H. Nguyen-An, T. Silverston, T. Yamazaki, and T. Miyoshi, "Generating iot traffic: A case study on anomaly detection," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2020.

[16] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[17] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Anomalous communications detection in iot networks using sparse autoencoders," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019.

[18] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[19] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, 2018.

[20] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[21] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014.

[23] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[24] D. Foster, *Generative deep learning: teaching machines to paint, write, compose, and play*. O'Reilly Media, 2019.

[25] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[26] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30*, 2017.

[27] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[28] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, 2016.

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[31] *The Earth Mover's Distance (EMD)*, 2020 (accessed June 24, 2020). [Online]. Available: http://infolab.stanford.edu/pub/cstr/reports/cs/tr/99/1620/CS-TR-99-1620.ch4.pdf

[32] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.