Flow-based Network Traffic Generation using Generative Adversarial Networks

Markus Ring[a,*], Daniel Schlör[b], Dieter Landes[a], Andreas Hotho[b]

[a] *Faculty of Electrical Engineering and Informatics, Coburg University of Applied Sciences, 96450 Coburg, Germany*
[b] *Data Mining and Information Retrieval Group, University of Würzburg, 97074 Würzburg, Germany*

Abstract

Flow-based data sets are necessary for evaluating network-based intrusion detection systems (NIDS). In this work, we propose a novel methodology for generating realistic flow-based network traffic. Our approach is based on Generative Adversarial Networks (GANs) which achieve good results for image generation. A major challenge lies in the fact that GANs can only process continuous attributes. However, flow-based data inevitably contain categorical attributes such as IP addresses or port numbers. Therefore, we propose three different preprocessing approaches for flow-based data in order to transform them into continuous values. Further, we present a new method for evaluating the generated flow-based network traffic which uses domain knowledge to define quality tests. We use the three approaches for generating flow-based network traffic based on the CIDDS-001 data set. Experiments indicate that two of the three approaches are able to generate high quality data.

Keywords: GANs, TTUR WGAN-GP, NetFlow, Generation, IDS

*Corresponding author
Email addresses: markus.ring@hs-coburg.de (Markus Ring), daniel.schloer@informatik.uni-wuerzburg.de (Daniel Schlör), dieter.landes@hs-coburg.de (Dieter Landes), hotho@informatik.uni-wuerzburg.de (Andreas Hotho)

1. Introduction

Detecting attacks within network-based traffic has been of great interest in the data mining community over decades. Recently, Buczak and Guven [1] presented an overview of the community effort with regard to this issue. However, there are still open challenges (e.g. the high cost of false-positives or the lack of labeled data sets which are publicly available) for the successful use of data mining algorithms for anomaly-based intrusion detection [2, 3]. In this work, we focus on a specific challenge within that setting.

Problem Statement. For network-based intrusion detection (NIDS), few labeled data sets are publicly available which contain realistic user behavior and up-to-date attack scenarios. Available data sets are often outdated or suffer from other shortcomings. Using real network traffic is also problematic due to the missing ground truth. Since flow-based data sets contain millions up to billions of flows, manual labeling of real network traffic is difficult even for security experts and extremely time-consuming. As another disadvantage, real network traffic often cannot be shared within the community due to privacy concerns. However, labeled data sets are necessary for training supervised data mining methods (e.g. classification algorithms) and provide the basis for evaluating the performance of supervised as well as unsupervised anomaly-based intrusion detection methods.

Objective. Large training data sets with high variance can increase the robustness of anomaly-based intrusion detection methods. Therefore, we intend to build a generative model which allows to generate realistic flow-based network traffic. The generated data can be used to improve the training of anomaly-based intrusion detection methods as well as for their evaluation. To that end, we propose an approach that is able to learn the characteristics of collected network traffic and generates new flow-based network traffic with the same underlying characteristics.

Approach and Contributions. Generative Adversarial Networks (GANs) [4] are a popular method to generate synthetic data by learning from a given

set of input data. GANs consist of two neural networks, a generator network G and a discriminator network D. The generator network G is trained to generate synthetic data from noise. The discriminator network D is trained to distinguish generated synthetic data from real world data. The generator network G is trained by the output signal gradient of the discriminator network D. G and D are trained iteratively until the generator network G is able to fool the discriminator network D. GANs achieve remarkably good results in image generation [5, 6, 7, 8]. Furthermore, GANs have also been used for generating text [9] or molecules [10].

This work uses GANs to generate complete flow-based network traffic with all typical attributes. To the best of our knowledge, this is the first work that uses GANs for this purpose. GANs can only process continuous input attributes. This poses a major challenge since flow-based network data consist of continuous and categorical attributes (e.g. *IP addresses* or *port numbers*). Therefore, we analyze different preprocessing strategies to transform categorical attributes of flow-based network data into continuous attributes. The first method simply treats attributes like *IP addresses* and *ports* as numerical values. The second method creates binary attributes from categorical attributes. The third method uses IP2Vec [11] to learn meaningful vector representations of categorical attributes. After preprocessing, we use Improved Wasserstein GANs (WGAN-GP) [12] with the two time-scale update rule (TTUR) proposed by Heusel et al. [13] to generate new flow-based network data based on the public CIDDS-001 [14] data set. Then, we evaluate the quality of the generated data with several evaluation measures. The proposed approach is able to generate realistic flows but does not consider the temporal dependencies of flow sequences. As a consequence, this approach can be used to generate additional training data for intrusion detection methods which process flows individually like [15], [16] or [17] and for all approaches which operate on data sets with no timestamps like KDD CUP 99 (e.g. [18]) or NSL-KDD (e.g. [19]). However, it can not be used to generate additional training data for approaches which operate on time windows or sequences of flows like [20] or [21].

The paper has several contributions. The main contribution is the generation of flow-based network data using GANs. We propose three different preprocessing approaches and a new evaluation method which uses domain knowledge to evaluate the quality of generated data. In addition to that, we extend IP2Vec [11] such that IP2Vec is able to learn similarities between the flow attributes: *bytes*, *packets* and *duration*.

Structure. The next section of the paper describes flow-based network traffic, GANs, and IP2Vec in more detail. In section 3, we present three different approaches for transforming flow-based network data. An experimental evaluation of these approaches is given in section 4 and the results are discussed in section 5. Section 6 analyzes related work on network traffic generators and GANs applied to the domain IT security. A summary and outlook on future work concludes the paper.

2. Foundations

This section starts with analyzing the underlying flow-based network traffic. Then, GANs are explained in more detail. Finally, we explain IP2Vec [11] which is the basis of our third data transformation approach.

2.1. Flow-based Network Traffic

We focus on flow-based network traffic in unidirectional *NetFlow* format [22]. Flows contain header information about network connections between two endpoint devices like servers, workstation computers or mobile phones. Each flow is an aggregation of transmitted network packets which share some properties [23]. Typically, all transmitted network packets with the same *source IP address*, *source port*, *destination IP address*, *destination port* and *transport protocol* within a time window are aggregated into one flow. *NetFlow* [22] aggregates all network packets which share these five properties into one flow until an active or inactive timeout is reached. In order to consolidate contiguous streams the aggregation of network packets stops if no further packet is received within

Table 1: Overview of typical attributes in flow-based data like *NetFlow* [22] or *IPFIX* [23]. The third column provides the type of the attributes and the last column shows exemplary values for these attributes.

#	Attribute	Type	Example
1	date first seen	timestamp	2018-03-13 12:32:30.383
2	duration	continuous	0.212
3	transport protocol	categorical	TCP
4	source IP address	categorical	192.168.100.5
5	source port	categorical	52128
6	destination IP address	categorical	8.8.8.8
7	destination port	categorical	80
8	bytes	numeric	2391
9	packets	numeric	12
10	TCP flags	binar/categorical	.A..S.

a time window of α second (inactive timeout). The active timeout stops the aggregation of network packets after β seconds, even if further network packets are observed to avoid unlikely long entries.

Table 1 shows the typical attributes of unidirectional *NetFlow* [22] data. *NetFlow* are heterogeneous data which consists of continuous, numeric, categorical and binary attributes. Most attributes like *IP addresses* and *ports* are categorical. Further, there is a timestamp attribute (*date first seen*), a continuous attribute (*duration*) and numeric attributes like *bytes* or *packets*. We define the type of *TCP flags* as binary/categorical. *TCP flags* can be either interpreted as six binary attributes (e.g. *isSYN* flag, *isACK* flag, etc.) or as one categorical value.

2.2. GANs

Discriminative models classify objects into predefined classes [24] and are often used for intrusion detection (e.g. in [18], [25], or [26]). In contrast to discriminative models, generative models are used to generate data like flow-

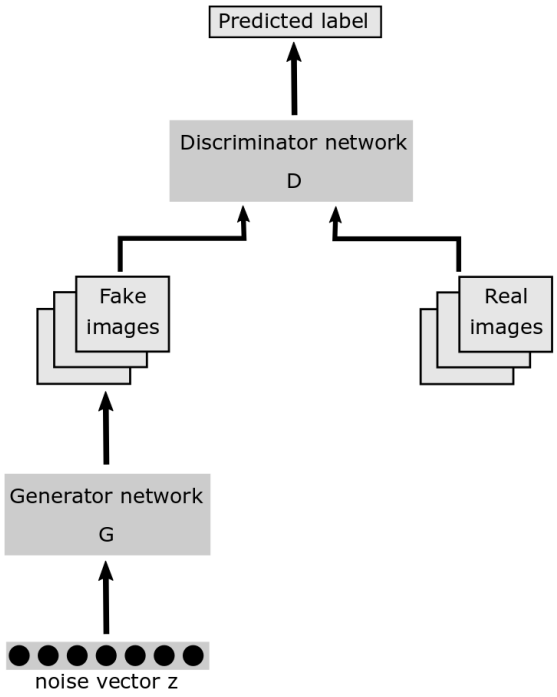

Figure 1: Architecture of GANs.

based network traffic. Many generative models build on likelihood maximization for a parametric probability distribution. As the likelihood function is often unknown or the likelihood gradient is computationally intractable, some models like Deep Boltzmann Machines [27] use approximations to solve this problem. Other models avoid this problem by not explicitly representing likelihood. Generative Stochastic Networks for example learn the transition operation of a Markov chain whose stationary distribution estimates the data distribution. GANs avoid Markov chains estimating the data distribution by a game-theoretic approach: The generator network G tries to mimic samples from the data distribution, while the discriminator network D has to differentiate real and generated samples. Both networks are trained iteratively until the discriminator D can't distinguish real samples from generated samples any more. Beside computational advantages, the generator G is never updated with real samples. Instead, the generator network G is fed with an input vector of noise z. The genera-

tor is trained using only the discriminator's gradients through backpropagation. Therefore, it is less likely to overfit the generator G by memorization and reproduction of real samples. Figure 1 illustrates the generation process.

Goodfellow et al. say that: *"another advantage of adversarial networks is that they can represent very sharp, even degenerate distributions"* [4] which is the case for some *NetFlow* attributes. However, (the original) vanilla GANs [4] require the visible units to be differentiable, which is not the case for categorical attributes like *IP addresses* in *NetFlow* data. Further, vanilla GANs are sensitive to parameter tuning and their loss function often does not correlate with the quality of the generated data. Gulrajani et al. [12] show that Wasserstein GANs (WGANs), besides other advantages, are capable of modeling discrete distributions over a continuous latent space and that their loss function correlates with the quality of generated data. In contrast to vanilla GANs, WGANs [8] use the Earth Mover (EM) distance as a value function replacing the classifying discriminator network with a *critic* network estimating the EM distance. While the original WGAN approach uses weight clipping to guarantee differentiability almost everywhere, Gulrajani et al. [12] improve training of WGANs by using gradient penalty as soft constrain to enforce the Lipschitz constraint. One research frontier in the area of GANs is to solve the issue of non-convergence [28]. Heusel et al. [13] propose a two time-scale update rule (TTUR) for training GANs with arbitrary loss functions. The authors prove that TTUR converges under mild assumptions to a stationary local Nash equilibrium.

For those reasons, we use Improved Wasserstein Generative Adversarial Networks (WGAN-GP) [12] with the two time-scale update rule (TTUR) from [13] in this work.

2.3. IP2Vec

2.3.1. Motivation of IP2Vec

IP2Vec [11] is inspired by Word2Vec [29, 30] and aims at transforming *IP addresses* into a continuous feature space $\mathbb{R}^m$ such that standard similarity measures can be applied. Ring et al. [11] show that IP2Vec is able to transform

IP addresses to semantic vector representations $\mathbb{R}^m$ based on their network behavior. The vector representations in [11] are used to distinguish clients from servers and infected hosts from non-infected hosts. Since the results from [11] are promising, we want to investigate the suitability of IP2Vec for our work.

IP2Vec transforms *IP addresses* to vector representations $\mathbb{R}^m$ using available context information from flow-based network traffic. *IP addresses* which appear frequently in similar contexts will be close to each other in the feature space $\mathbb{R}^m$. More precisely, similar contexts imply to IP2Vec that the devices associated to these *IP addresses* establish similar network connections. Figure 2 illustrates the basic idea.

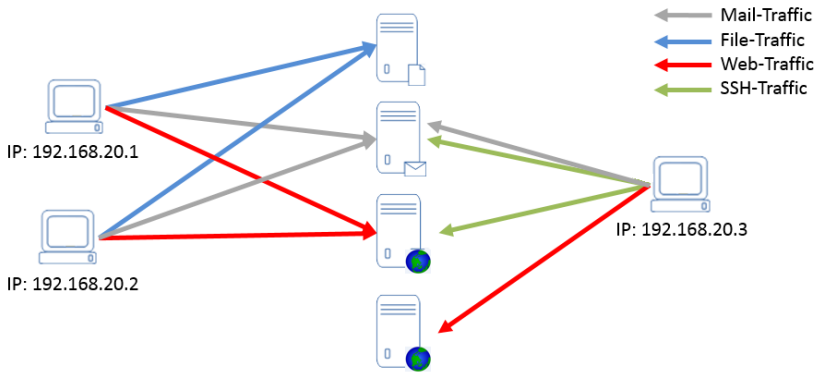

Figure 2: Idea of IP2Vec.

Arrows in Figure 2 denote network connections from three *IP addresses*, namely *192.168.20.1*, *192.168.20.2*, and *192.168.20.3*. Colors indicate different services. Consequently, IP2Vec leads to the following result:

$$sim(192.168.20.1, 192.168.20.2) > sim(192.168.20.1, 192.168.20.3), \quad (1)$$

where $sim(X, Y)$ is an arbitrary similarity function (e.g. cosine similarity) between the *IP addresses* X and Y. IPVec considers *IP addresses 192.168.20.1* and *192.168.20.2* as more similar than *192.168.20.1* and *192.168.20.3* because the *IP addresses 192.168.20.1* and *192.168.20.2* refer to the same targets and use the same services. In contrast to that, the *IP address 192.168.20.3* targets

different servers and uses different services (SSH-Connections).

2.3.2. Model

IP2Vec is based upon a fully connected neural network with a single hidden layer (see Figure 3).

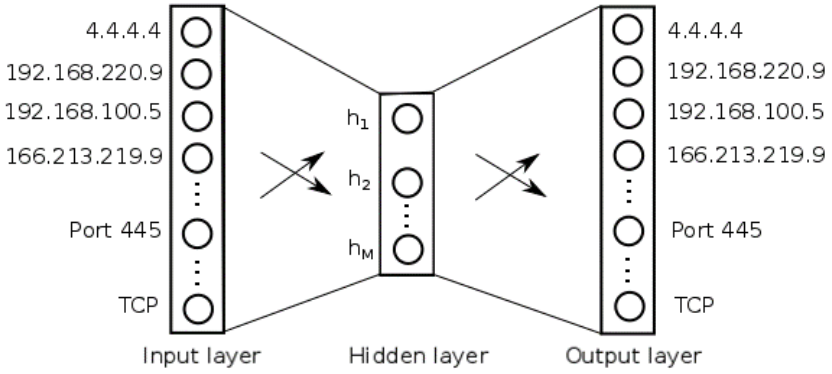

Figure 3: Architecture of the neural network used by IP2Vec.

The features extracted from flow-based network traffic constitute the neural network's input. These features (*IP addresses*, *destination ports* and *transport protocols*) define the input *vocabulary* which contains all *IP addresses*, *destination ports* and *transport protocols* that appear in the flow-based data set. Since neural networks cannot be fed with categorical attributes, each value of our input *vocabulary* is represented as an one-hot vector. The length of the one-hot-vector is equal to the size of the *vocabulary*. Each neuron in the input and output layer is assigned a specific value of the *vocabulary* (see Figure 3).

Let us assume the training data set contains 100,000 different *IP addresses*, 20,000 different *destination ports* and 3 different *transport protocols*. Then, the size of the one-hot vector is 120,003 and only one component is 1, while all others are 0. Input and output layers comprise exactly the same number of neurons which is equal to the size of the *vocabulary*. The output layer uses a softmax classifier which indicates the probabilities for each value of the *vocabulary* that it appears in the same flow (context) as the input value to the neural network. The softmax classifier [31] normalizes the output of all output neurons such that

Table 2: Generation of training samples in IP2Vec [11]. Input values are highlighted with cyan background and expected output values are highlighted with gray background. The following abbrevations are used: src IP addr. (source IP address), dst IP addr. (destination IP address), dst port (destintion port), proto (transport protocol).

					input value	output value
src IP addr.	dst IP addr.	dst port	proto	→	src IP addr.	dst IP addr.
					src IP addr.	dst port
					src IP addr.	proto
src IP addr.	dst IP addr.	dst port	proto	→	dst port	dst IP addr.
src IP addr.	dst IP addr.	dst port	proto	→	proto	dst IP addr.

the sum of the outputs is 1. The number of neurons in the hidden layer is much smaller than the number of neurons in the input layer.

2.3.3. Training

The neural network is trained using captured flow-based network traffic. IP2Vec uses only the *source IP address*, *destination IP address*, *destination port* and *transport protocol* of flows. Table 2 outlines the generation of training samples.

IP2Vec generates five training samples from each flow. Each training sample consists of an input value and an expected output value. In the first step, IP2Vec selects an input value for the training sample. The selected input value is highlighted with cyan background in Table 2. The expected output values for the corresponding input value are highlighted with gray background. In Table 2 can be seen that IP2Vec generates three training samples where the *source IP address* is the input value, one training sample where the *destination port* is the input value and one training sample where the *transport protocol* is the input value.

In the training process, the neural network is fed with the input value and tries to predict the probabilities of the other values from the vocabulary. For

training samples, the probability of the concrete output value is 1 and 0 for all other values. In general, the output layer indicates the probabilities for each value of the input *vocabulary* that it appears in the same flow as the given input value.

The network uses back-propagation for learning. This kind of training, however, could take a lot of time. Let us assume that the hidden layer comprises 32 neurons and the training data set encompasses one million different *IP addresses* and *ports*. This results in 32 million weights in each layer of the network. Consequently, training such a large neural network is going to be slow. To make things worse, a huge amount of training flows is required for adjusting that many weights and for avoiding overfitting. Consequently, we have to update millions of weights for millions of training samples. Therefore, IP2Vec attempts to reduce the training time by using Negative Sampling in a similar way as Word2Vec does [29]. In Negative Sampling, each training sample modifies only a small percentage of the weights, rather than all of them. More details on Negative Sampling may be found in [11] and [30].

2.3.4. Continuous Representation of IP addresses

After the training phase, IP2Vec uses the weights of the hidden layer as m-dimensional vector representations of *IP addresses*. That means, a 32 dimensional continuous representation of each *IP address*, *transport protocol* and *port* is obtained if the hidden layer comprises 32 neurons.

Intuition. Why does this approach work? If two *IP addresses* refer to similar *destination IP addresses*, *destination ports*, and *transport protocols*, then the neural network needs to output similar results for these *IP addresses*. One way for the neural network to learn similar output values for different input values is to learn similar weights in the hidden layer of the network. Consequently, if two *IP addresses* exhibit similar network behavior, IP2Vec attempts to learn similar weights (which are the vectors of the target feature space $\mathbb{R}^m$) in the hidden layer.

3. Transformation Approaches

This section describes three different methods to transform the heterogeneous *NetFlow* data such that they may be processed by Improved Wasserstein Generative Adversarial Networks (WGAN-GP).

3.1. Preliminaries

In general, we use in all three methods the same preprocessing steps for the attributes *date first seen*, *transport protocol*, and *TCP flags* (see Table 1).

Usually, the concrete timestamp is marginal for generating realistic flow-based network data. Instead, many intrusion detection systems derive additional information from the timestamp like *"is today a working day or weekend day"* or *"does the event occur during typical working hours or at night"*. Therefore, we do not generate timestamps. Instead, we create two attributes *weekday* and *daytime*. To be precise, we extract the weekday information of flows and generate seven binary attributes *isMonday*, *isTuesday* and so on. Then, we interpret the daytime as seconds $[0, 86400)$ and normalize them to the interval $[0, 1]$. We transform the *transport protocol* (see #3 in Table 1) to three binary attributes, namely *isTCP*, *isUDP*, and *isICMP*. The same procedure is followed for *TCP flags* (see #10 in Table 1) which are transformed to six binary attributes *isURG*, *isACK*, *isPUS*, *isSYN*, *isRES*, and *isFIN*.

3.2. Method 1 - Numeric Transformation

Although *IP addresses* and *ports* look like real numbers, they are actually categorical. Yet, the simplest approach is to interpret them as numbers after all and treat them as continuous attributes. We refer to this method as *Numeric-based Improved Wasserstein Generative Adversarial Networks (short: N-WGAN-GP)*. This method transforms each octet of an *IP address* to the interval [0,1], e.g. 192.168.220.14 is transformed to four continuous attributes: (ip_1) $192/255 = 0.7529$, (ip_2) $168/255 = 0.6588$, (ip_3) $220/255 = 0.8627$ and (ip_4) $14/255 = 0.0549$. We do a similar procedure for *ports* by dividing them

through the highest port number, e.g. the *source port = 80* will be transformed to one continuous attribute $80/65535 = 0.00122$.

The attributes *duration*, *bytes* and *packets* (see attributes #2, #8 and #9 in Table 1) are normalized to the interval $[0, 1]$. Table 3 provides examples and compares the three transformation methods.

3.3. Method 2 - Binary Transformation

The second method creates several binary attributes for *IP addresses*, *ports*, *bytes*, and *packets*. We refer to this method as *Binary-based Improved Wasserstein Generative Adversarial Networks (short: B-WGAN-GP)*. Each octet of an *IP address* is mapped to it's 8-bit binary representation. Consequently, *IP addresses* are transformed into 32 binary attributes, e.g. 192.168.220.14 is transformed to 11000000 10101000 11011100 00001110. *Ports* are converted to their 16-bit binary representation, e.g. the *source port 80* is transformed to 00000000 01010000. For representing *bytes* and *packets*, we transform them to a binary representation as well and limit their length to 32 bit. The attribute *duration* is normalized to the interval $[0, 1]$. Table 3 shows an example for this transformation procedure.

3.4. Method 3 - Embedding Transformation

The third method transforms *IP addresses*, *ports*, *duration*, *bytes*, and *packets* into so-called *embeddings* in a m-dimensional continuous feature space $\mathbb{R}^m$ following the ideas in Section 2.3. We refer to this method as *Embedding-based Improved Wasserstein Generative Adversarial Networks (short: E-WGAN-GP)*.

E-WGAN-GP extends IP2Vec (see Section 2.3) for learning embeddings not only for *IP addresses*, *ports*, and *transport protocols*, but also for the attributes *duration*, *bytes*, and *packets*. To that end, the *input vocabulary* of IP2Vec is extended by the values of the latter three attributes and additional training pairs are extracted from each flow. Table 4 presents the extended training sample generation.

Table 3: Preprocessing of flow-based data. The first column provides the original flow attributes and examplarly values, the other columns show the extracted features (column *Attr.*) and the corresponding values (column *Value*)) for each of preprocessing method.

	N-WGAN-GP		B-WGAN-GP		E-WGAN-GP	
Attribute / Value	Attr.	Value	Attr.	Value	Attr.	Value
date first seen	isMonday	1	isMonday	1	isMonday	1
2018-05-28 11:39:23	isTuesday	0	isTuesday	0	isTuesday	0
	isWednesday	0	isWednesday	0	isWednesday	0
	isThursday	0	isThursday	0	isThursday	0
	isFriday	0	isFriday	0	isFriday	0
	isSaturday	0	isSaturday	0	isSaturday	0
	isSunday	0	isSunday	0	isSunday	0
	daytime	$\frac{41963}{86400} = 0.485$	daytime	$\frac{41963}{86400} = 0.485$	daytime	$\frac{41963}{86400} = 0.485$
duration	norm_dur	$\frac{1.503 - dur_{min}}{dur_{max} - dur_{min}}$	norm_dur	$\frac{1.503 - dur_{min}}{dur_{max} - dur_{min}}$	dur_1	$\begin{pmatrix} e_1 \\ ... \\ e_m \end{pmatrix}$
1.503					...	
					dur_m	
transport protocol	isTCP	1	isTCP	1	isTCP	1
TCP	isUDP	0	isUDP	0	isUDP	0
	isICMP	0	isICMP	0	isICMP	0
IP address	ip_1	$\frac{192}{255} = 0.7529$	ip_1 to ip_8	1,1,0,0,0,0,0,0	ip_1	$\begin{pmatrix} e_1 \\ ... \\ e_m \end{pmatrix}$
192.168.210.5	ip_2	$\frac{168}{255} = 0.6588$	ip_9 to ip_16	1,0,1,0,1,0,0,0	...	
	ip_3	$\frac{210}{255} = 0.8627$	ip_17 to ip_24	1,1,0,1,0,0,1,0	ip_m	
	ip_4	$\frac{5}{255} = 0.0196$	ip_25 to ip_32	0,0,0,0,0,1,0,1		
port	pt	$\frac{53872}{65535} = 0.8220$	pt_1 to pt_8	1,1,0,1,0,0,1,0	pt_1	$\begin{pmatrix} e_1 \\ ... \\ e_m \end{pmatrix}$
53872			pt_9 to pt_16	0,1,1,1,0,0,0,0	...	
					pt_m	
bytes	norm_byt	$\frac{1.503 - byt_{min}}{byt_{max} - byt_{min}}$	byt_1 to byt_8	0,0,0,0,0,0,0,0	byt_1	$\begin{pmatrix} e_1 \\ ... \\ e_m \end{pmatrix}$
144			byt_9 to byt_16	0,0,0,0,0,0,0,0	...	
			byt_17 to byt_24	0,0,0,0,0,0,0,0	byt_m	
			byt_25 to byt_32	1,0,0,1,0,0,0,0		
packets	norm_pck	$\frac{1.503 - pck_{min}}{pck_{max} - pck_{min}}$	pck_1 to pck_8	0,0,0,0,0,0,0,0	pck_1	$\begin{pmatrix} e_1 \\ ... \\ e_m \end{pmatrix}$
1			pck_9 to pck_16	0,0,0,0,0,0,0,0	...	
			pck_17 to pck_24	0,0,0,0,0,0,0,0	pck_m	
			pck_25 to pck_32	0,0,0,0,0,0,0,1		
TCP flags	isURG	0	isURG	0	isURG	0
.A..S.	isACK	1	isACK	1	isACK	1
	isPSH	0	isPSH	0	isPSH	0
	isRES	0	isRES	0	isRES	0
	isSYN	1	isSYN	1	isSYN	1
	isFIN	0	isFIN	0	isFIN	0

Table 4: Extended generation of training samples in IP2Vec. Input values are highlighted with cyan background and expected output values are highlighted with gray background. The following abbrevations are used: src IP addr. (source IP address), dst IP addr. (destination IP address), dst port (destintion port), proto (transport protocol).

									input value	output value
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	src IP addr.	dst IP addr.
								→	src IP addr.	src port
								→	src IP addr.	proto
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	dst IP addr.	src IP addr.
								→	dst IP addr.	dst port
								→	dst IP addr.	proto
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	src port	src IP addr.
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	dst port	dst IP addr.
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	bytes	packets
								→	bytes	duration
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	packets	bytes
								→	packets	duration
src IP addr.	src port	dst IP addr.	dst port	proto	bytes	packets	duration	→	duration	packets

Each flow produces 13 training samples each of which consists of an input and an expected output value. The input values are highlighted with cyan background in Table 4. The expected output values for the corresponding input value are highlighted with gray background. Our adapted training sample generation extracts further training samples for the attributes *bytes*, *packets* and *duration*. Further, we also create training pairs with the *destination IP address* as input. Ring et al. [11] argue that it is not necessary to extract training samples with *destination IP addresses* as input when working on unidirectional flows. Yet, in this case, IP2Vec does not learn meaningful representation for multi- and broadcast IP addresses which only appear as *destination IP addresses* in flow-based network traffic. Table 3 shows the result of an exemplary transformation.

E-WGAN-GP maps flows to embeddings which need to be re-transformed to the original space after generation. To that end, values are replaced by the closest embeddings generated by IP2Vec. For instance, we calculate the cosine similarity between the generated output for the *source IP address* and all existing *IP address* embeddings generated by IP2Vec. Then, we replace the output with the *IP address* which has the highest similarity.

4. Experiments

This section provides an experimental evaluation of our three approaches *N-WGAN-GP*, *B-WGAN-GP* and *E-WGAN-GP* for synthetic flow-based network traffic generation.

4.1. Data Set

We use the publicly available *CIDDS-001* data set [14] which contains unidirectional flow-based network traffic as well as detailed information about the networks and *IP addresses* within the data set. Figure 4 shows an overview of the emulated business environment of the *CIDDS-001* data set. In essence, the *CIDDS-001* data set contains four internal subnets which can be identified by their IP address ranges: a developer subnet (*dev*) with exclusively Linux clients, an office subnet (*off*) with exclusively Windows clients, a management subnet (*mgt*) with mixed clients, and a server subnet (*srv*). Additional knowledge facilitates the evaluation of the generated data (see Section 4.3).

The *CIDDS-001* data set contains four weeks of network traffic. We consider only the network traffic which was captured at the network device within the *OpenStack* environment (see Figure 4) and divide the network traffic in two parts: *week1* and *week2-4*. The first two weeks contain normal user behavior and attacks, whereas *week3* and *week4* contain only normal user behavior and no attacks. We use this kind of splitting in order to obtain a large training data set *week2-4* for our generative models and simultaneously provide a reference data set *week1* which contains normal and malicious network behavior. Overall, *week2-4* contains around 22 million flows and *week1* contains around 8.5 million flows. We consider only the TCP, UDP and ICMP flows and remove the 895 IGMP flows from the data set.

4.2. Definition of a Baseline

As baseline for our experiments, we build a generative model which creates new flows based on the empirical probability distribution of the input data. The baseline estimates the probability distribution for each attribute by counting

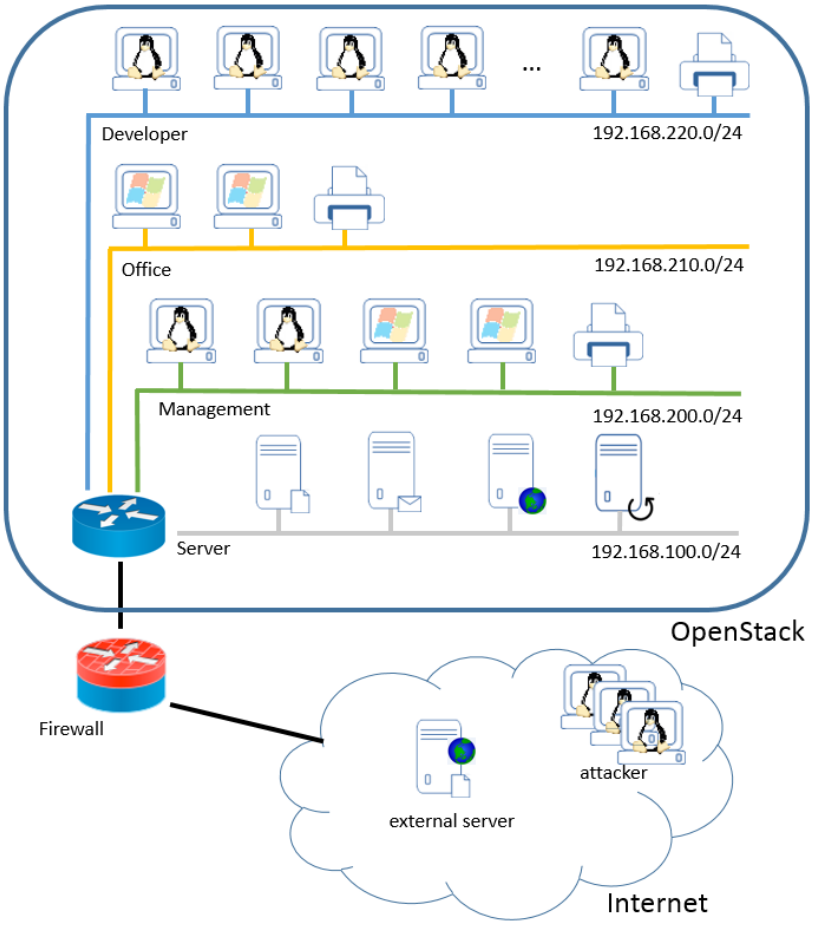

Figure 4: Overview of the simulated network environment from the CIDDS-001 data set [14].

from the input data. New flows are generated by drawing from the empirical probability distributions. Each attribute is drawn independently from other attributes.

4.3. Evaluation Methodology

Evaluation of generative models is challenging and an open research topic: Borji [32] analyzed different evaluation measures for GANs. Images generated with GANs are often presented to human judges and evaluated by visual comparison. Another well-known evaluation measure for images is the Inception Score (IS) [33]. IS classifies generated images in 1000 different classes using the Inception Net v3 [34]. IS, however, is not applicable in our scenario since the Inception Net v3 can only classify images, but no flow-based network traffic.

In the IT security domain, there is neither consensus on how to evaluate network traffic generators, nor a standardized methodology [35]. Glasser and Lindauer [36] discuss about the problem of evaluating synthetic data. The authors conclude that synthetic data will only be realistic in some limited and measurable dimensions in the absence of a clear definition of realism. Therefore, Glasser and Lindauer use human feedback from domain experts to evaluate the quality of generated data for anomaly detection. Stiborek et al. [37] use an anomaly score to evaluate their generated data. Siska et al. [38] and Iannucci et al. [39] build graphs and evaluate the diversity of the generated traffic by comparing the number of nodes and edges between generated and real network traffic. Other flow-based network traffic generators often focus on specific aspects in their evaluation, e.g. distributions of *bytes* or *packets* are compared with real *NetFlow* data in [40] and [41].

Since there is no single widely accepted evaluation methodology, we use several evaluation approaches to assess the quality of the generated data from different views. To evaluate the diversity and distribution of the generated data, we visualize attributes (see Section 4.4.2) and compute the Euclidean distances between generated and real flow-based network data (see Section 4.4.3). To evaluate the quality of the content and relationships between attributes within

a flow, we introduce *domain knowledge checks* (see Section 4.4.4) as a new evaluation method. This method is developed on the basic idea of Glasser and Lindauer [36]. While Glasser and Lindauer [36] use feedback from human experts, the *domain knowledge checks* are automated test procedures on the basis of domain knowledge.

4.4. Generation of Flow-based Network Data

Now, we evaluate the quality of the generated data by the *baseline* (see Section 4.2), *N-WGAN-GP*, *B-WGAN-GP*, and *E-WGAN-GP* (see Section 3).

4.4.1. Parameter Configuration

For all four approaches, we use *week2-4* of the *CIDDS-001* data set as training input and generate 8.5 million flows for each approach.

We configured *N-WGAN-GP*, *B-WGAN-GP* and *E-WGAN-GP* to use a feed-forward neural network as generator and discriminator. Furthermore, we used the default parameter configuration of [13] and trained the networks for 5 epochs. An epoch is one training iteration over the complete training data set [31]. Consequently, we use each flow of the training data set five times for training the neuronal networks. We observed that a higher number of epochs neither leads to increasing quality nor reduces the loss values of the GANs. For identifying the number of neurons in each hidden layer, we set up a small parameter study in which we varied the number of neurons from 8 to 192. We found that 80 neurons in each hidden layer were sufficient for *B-WGAN-GP* and *E-WGAN-GP*. Similar numbers of neurons (e.g. 64 or 96) in each hidden layer lead to no significant changes in the quality of the generated data. For *N-WGAN-GP*, we set the number of neurons in the hidden layer to 24 since the numerical representation of flows is much smaller than for *B-WGAN-GP* or *E-WGAN-GP*.

Additionally, we have to learn embeddings for *E-WGAN-GP* in a previous step. Therefore, we configured IP2Vec to use 20 neurons in the hidden layer and trained the network like Ring et al. [11] for 10 epochs.

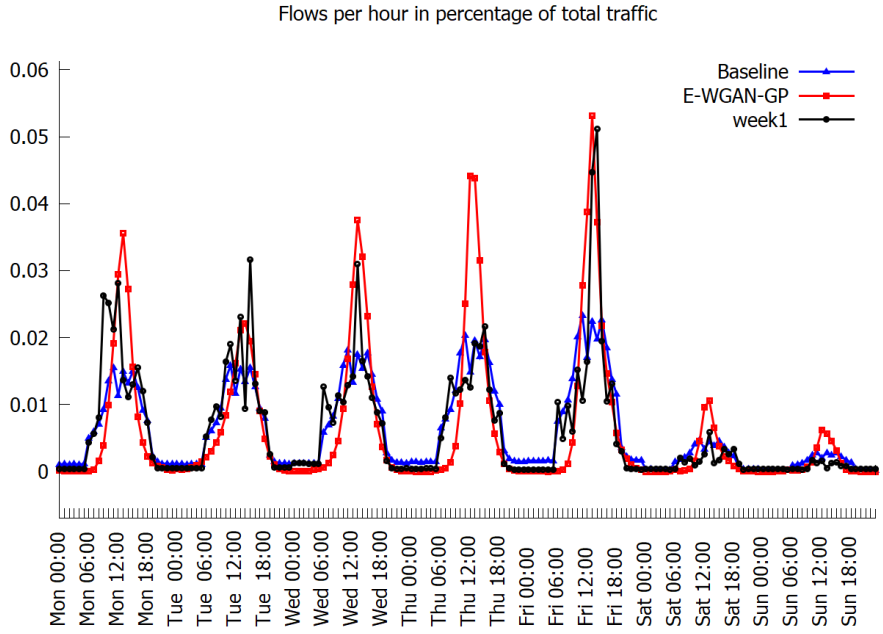

Figure 5: Temporal distribution of flows per hour.

4.4.2. Visualization

Figure 5 shows the temporal distribution of the generated flows and reference week *week1*. The y-axis shows the flows per hour as a percentage of total traffic and the three lines represent the reference week (*week1*), the generated data of the baseline (*baseline*), and the generated data of the E-WGAN-GP approach (*E-WGAN-GP*). Since all three transformation approaches process the attribute *date first seen* in the same way, only (*E-WGAN-GP*) is included for the sake of brevity. *E-WGAN-GP* reflects the essential temporal distribution of flows. In the *CIDDS-001* data set, the emulated users exhibit common behavior including lunch breaks and offline work which results in temporal limited network activities and a jagged curve (e.g. around 12:00 on working days). In contrast to that, the curve of *E-WGAN-GP* is smoother than the curve of the original traffic *week1*.

In the following, we use different visualization plots in order to get a deeper understanding of the generated data.

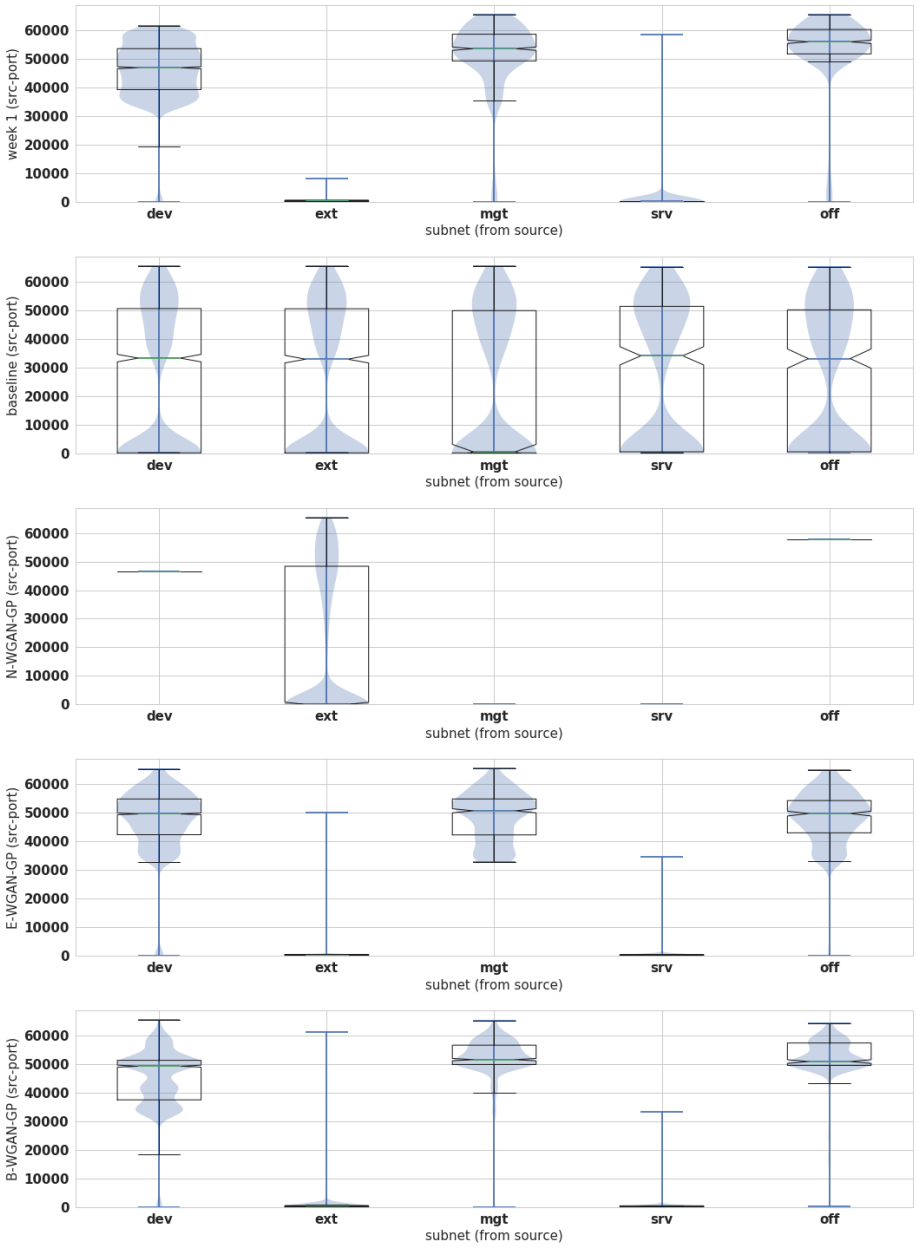

Figure 6: Distribution of the attribute *source port* for the subnets. The rows show in order: (1) data sampled from real data (week 1) and data generated by (2) baseline, (3) N-WGAN-GP, (4) E-WGAN-GP and (5) B-WGAN-GP.

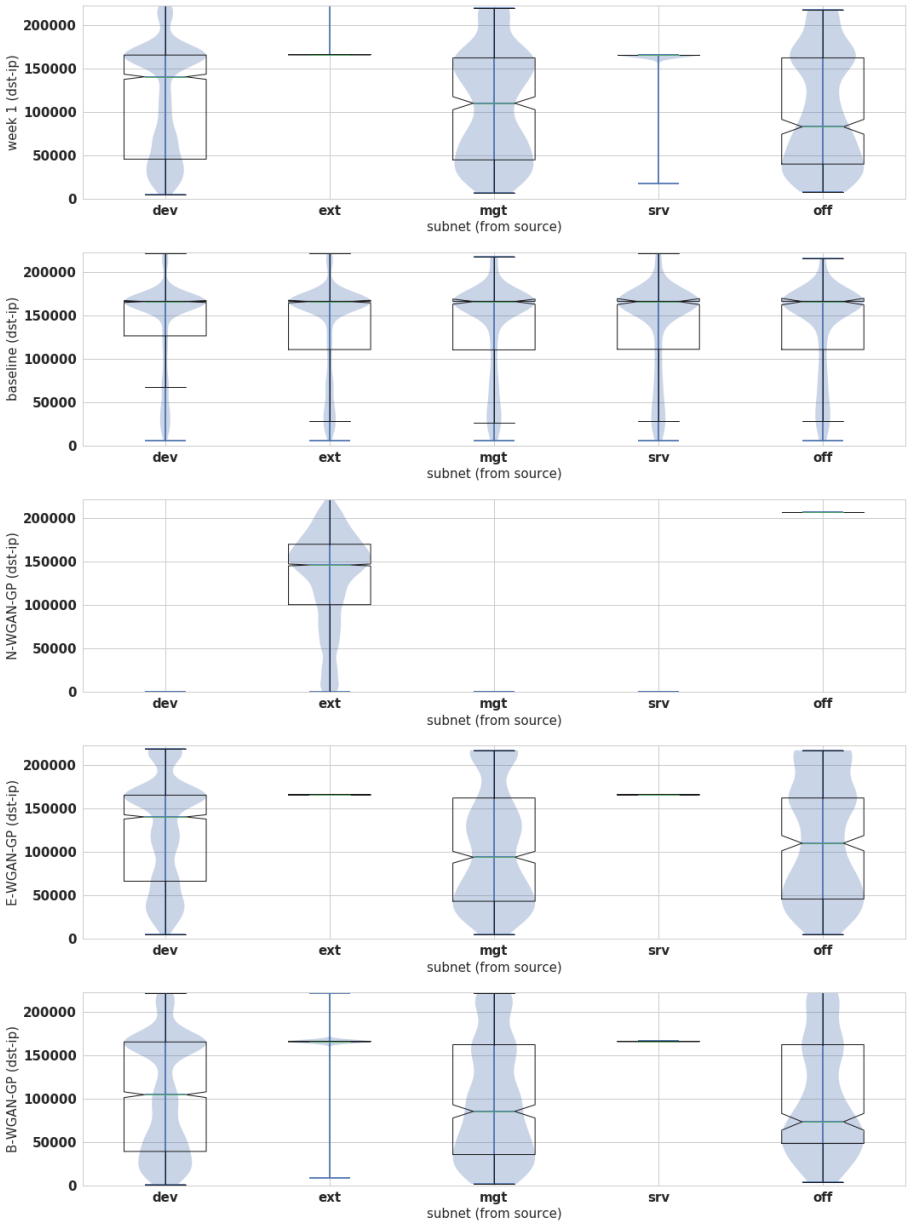

Figure 7: Distribution of the attribute *destination ip* for the subnets. The rows show in order: (1) data sampled from real data (week 1) and data generated by (2) baseline, (3) N-WGAN-GP, (4) E-WGAN-GP and (5) B-WGAN-GP.

Figures 6 and 7 show the real distributions (first row) sampled from *week1* respectively generated distributions by our maximum likelihood estimator *baseline* (second row) and generated distributions by our *WGAN-GP* models using different data representations for each row (third to fifth row). Each violin plot shows the data distribution of the attribute *source port* (Figure 6) respectively the attribute *destination IP address* (Figure 7) for the different *source IP addresses* grouped by their subnet (see Section 4.1). *IP addresses* from different subnets come along with different network behavior. For instance, *IP addresses* from the *mgt* subnet are typically clients which use services while *IP addresses* from the *srv* subnet are servers which offer services. This knowledge was not explicitly modeled during data generation.

We will now briefly discuss the conditional distribution of *source ports* (Figure 6). In the first row, we can clearly distinguish typical client-port (*dev*, *mgt*, *off*) and server-port (*ext*, *srv*) distributions. As expected, the maximum likelihood *baseline* is not able to capture the differences of the distributions depending on the subnet of the *source IP address* and models a distribution which is a combination of all five subnets from the input data. In contrast, the *B-WGAN-GP* and *E-WGAN-GP* capture the conditional probability distributions for the *source port* given the subnet of the *source IP address* very well.

N-WGAN-GP is incapable of representing the distributions properly. Note that almost exclusively flows with external *source IP addresses* are generated in the selected samples. In-depth analysis of the generated data suggests that numeric representations fail to match the designated subnets exactly. As nearly all generated data is assigned to the *ext* subnet, it comes as no surprise that the distribution represents a combination of all five subnets from the input data for both *source ports* (Figure 6) and *destination IP addresses* (Figure 7).

For the attribute *destination IP address*, the distribution is a mixture of external and internal *IP addresses* for *dev*, *mgt* and *off* subnets (see reference week *week1*). This matches the user roles, surfing on the internet (external) as well as accessing internal services (e.g. printers). For external subnets, the

destination IP address has to be within the internal IP address range. Traffic from external sources to external targets does not run through the simulated network environment of the *CIDDS-001* data set. Consequently, there is no flow within the *CIDDS-001* data set which has a *source IP address* and a *destination IP address* from the *ext* subnet. This fact can be seen for *week1* in Figure 7 where flows which have their origin in the *ext* subnet only address a small range of *destination IP addresses* which reflect the range of internal *IP addresses*. *E-WGAN-GP* and *B-WGAN-GP* capture this property very well while the *baseline* and *N-WGAN-GP* fail to capture this property.

4.4.3. Euclidean Distances

The second evaluation compares the distribution of the generated and real flow-based network data in each attribute independently. Therefore, we calculate Euclidean distances between the probability distributions of the generated data and the input flow-based network data (*week2-4*) in each attribute. We choose the Euclidean distance over the Kullback-Leibler divergence in order to avoid calculation problems where the probability of generated data is zero. Table 5 highlights the results. We refrain from calculating the Euclidean distance for the attribute *date first seen* since exact matches of timestamps (considering seconds and milliseconds) do not make sense. At this point, we refer to Figure 5 which analyzes the temporal distribution of the generated timestamps.

Network traffic is subject to concept drift and exact reproduction of probability distributions is not desirable. This fact can be seen in Table 5 where the Euclidean distances between the probability distributions from *week1* and *week2-4* of the *CIDDS-001* data set are between 0.02 and 0.14. Consequently, generated network traffic should have similar Euclidean distances to the training data like the reference week *week1*. However, it should be mentioned that there is no perfect distance value x which indicates the correct amount of concept drift. The generated data of *E-WGAN-GP* tends to have similar distances to the training data (*week2-4*) like the reference data set *week1*. Table 5 shows that the *baseline* has the lowest distance to the training data in each attribute. The

Table 5: Euclidian distances between the training data (*week2-4*) and the generated flow-based network traffic in each attribute.

Attribute	Baseline	N-WGAN-GP	B-WGAN-GP	E-WGAN-GP	week1
duration	0.0002	0.4764	0.4764	0.0525	0.0347
transport protocol	0.0001	0.0014	0.0042	0.0015	0.0223
source IP address	0.0003	0.1679	0.0773	0.0988	0.1409
source port	0.0003	0.5658	0.0453	0.0352	0.0436
destination IP address	0.0003	0.1655	0.0632	0.1272	0.1357
destination port	0.0003	0.5682	0.0421	0.0327	0.0437
bytes	0.0002	0.5858	0.0391	0.0278	0.0452
packets	0.0004	1.0416	0.0578	0.0251	0.0437
TCP flags	0.0003	0.0217	0.0618	0.0082	0.0687

generated data of *N-WGAN-GP* differs considerably from the training data set in some attributes. This is because *N-WGAN-GP* often does not generate the exact values but a large number of new values. The binary approach *B-WGAN-GP* has small distances in most attributes (except for attribute *duration*). This may be caused by the distribution of *duration* in the training data as most flows in the training data set have very small values in this attribute. Further, the normalization of the duration to interval $[0, 1]$ entails that almost all flows have very low values in this attribute. *N-WGAN-GP* and *B-WGAN-GP* tend to generate the smallest possible duration (0.000 seconds) for all flows.

4.4.4. Domain Knowledge Checks

We use *domain knowledge checks* to evaluate the intrinsic quality of the generated data. To that end, we derive several properties that generated flow-based network data need to fulfill in order to be realistic. We use the following seven heuristics as sanity checks:

- Test 1: If the *transport protocol* is UDP, then the flow must not have any

TCP flags.

- Test 2: The *CIDDS-001* data set is captured within an emulated company network. Therefore, at least one IP address (*source IP address* or *destination IP address*) of each flow must be internal (starting with 192.168.XXX.XXX).

- Test 3: If the flow describes normal user behavior and the *source port* or *destination port* is 80 (HTTP) or 443 (HTTPS), the *transport protocol* must be *TCP*.

- Test 4: If the flow describes normal user behavior and the *source port* or *destination port* is 53 (DNS), the *transport protocol* must be *UDP*.

- Test 5: If a multi- or broadcast IP address appears in the flow, it must be the *destination IP address*.

- Test 6: If the flow represents a *netbios* message (*destination port* is 137 or 138), the *source IP addresses* must be internal (192.168.XXX.XXX) and the *destination IP address* must be an internal broadcast (192.168.XXX.255).

- Test 7: *TCP*, *UDP* and *ICMP* packets have a minimum and maximum packet size. Therefore, we check the relationship between *bytes* and *packets* in each flow according to the following rule:

  $42 * packets \leq bytes \leq 65.535 * packets$

Table 6 shows the results of checking the generated data against these rules.

The reference data set *week1* achieves 100 percent in each test which is not surprising since the data is real flow-based network traffic which is captured in the same environment as the training data set. The *baseline* approach does not capture dependencies between flow attributes and achieves worse results. This can be especially observed in Tests 1, 4, and 6. Since multi- and broadcast *IP addresses* appear only in the attribute *destination IP address*, the *baseline* cannot fail Test 5 and achieves 100 percent.

Table 6: Results of the *domain knowledge checks* in percentage. Higher values indicate better results.

	Baseline	N-WGAN-GP	B-WGAN-GP	E-WGAN-GP	week1
Test 1	14.08	96.46	97.88	**99.77**	100.0
Test 2	81.26	0.61	98.90	**99.98**	100.0
Test 3	86.90	95.45	**99.97**	**99.97**	100.0
Test 4	15.08	7.14	**99.90**	99.84	100.0
Test 5	**100.0**	25.79	47.13	99.80	100.0
Test 6	0.07	0.00	40.19	**92.57**	100.0
Test 7	71.26	**100.0**	85.32	99.49	100.0

For our generative models, *E-WGAN-GP* achieves the best results on average. The usage of embeddings leads to more meaningful similarities within categorical attributes and facilitates the learning of interrelationships. Embeddings, however, also reduce the possible resulting space since no new values can be generated. *B-WGAN-GP* generates flows which achieve high accuracy in Tests 1 to 4. However, this approach shows weaknesses in Tests 5 and 6 where several internal relationships must be considered. The numerical approach *N-WGAN-GP* has the lowest accuracy in the tests. In particular, Test 4 shows that normalization of *source port* or *destination port* to a single continuous attribute is inappropriate. Straightforward mapping of 2^{16} different port values to one continuous attribute leads to too many values for a good reconstruction. In contrast to that, the binary representation of *B-WGAN-GP* leads to better results in that test.

5. Discussion

Flow-based network traffic consists of heterogeneous data and GANs can only process continuous input values. To solve this problem, we analyze three methods to transform categorical to continuous attributes. The advantages and

disadvantages of these approaches are discussed in the following.

N-WGAN-GP is a straightforward numeric method but leads to unwanted similarities between categorical values which are not similar considering real data. For instance, this transformation approach assesses the *IP addresses* 192.168.220.10 and 191.168.220.10 as highly similar although the first *IP address* 192.168.220.10 is private and the second *IP address* 191.168.220.10 is public. Hence, the two addresses should be ranked as fairly dissimilar. Obviously, even small errors in the generation process can cause significant errors. This effect can be observed in Test 2 (see Table 6) where *N-WGAN-GP* has problems with the generation of private *IP addresses*. Instead, this approach often generates non-private *IP addresses* such as 191.168.X.X or 192.167.X.X. In image generation, the original application domain of GANs, small errors do not have serious consequences. A brightness 191 instead of 192 in a generated pixel has nearly no effect on the image and the error is (normally) not visible for human eyes. Further, *N-WGAN-GP* normalizes the numeric attributes *bytes* and *packets* to the interval $[0, 1]$. The generated data are then de-normalized using the original training data. Here, we can observe that real flows often have typical byte sizes like 66 bytes which are also not exactly matched. This results in higher Euclidean distances in these attributes (see Table 5). Overall, the first method *N-GAN-WP* does not seem to be suitable for generating realistic flow-based network traffic.

B-WGAN-GP extracts binary attributes from categorical attributes and converts numerical attributes to their binary representation. Using this transformation, additional structural information (e.g. subnet information) of *IP addresses* can be maintained. Further, *B-WGAN-GP* assigns larger value ranges to categorical values in the transformed space than *N-WGAN-GP*. While *N-WGAN-GP* uses a single continuous attribute to represent a *source port*, *B-WGAN-GP* uses 16 binary attributes for representation. These two aspects support *B-WGAN-GP* in generating better categorical values of a flow as can be observed in the results of the *domain knowledge checks* (see e.g. Test 2 and Test 4 in Table 6). Further, Figures 6 and 7 indicate that *B-WGAN-GP*

captures the internal structure of the traffic very well even though it is less restricted than *E-WGAN-GP* with respect to the treatment of previously unseen values.

E-WGAN-GP learns embeddings for *IP addresses*, *ports*, *bytes*, *packets*, and *duration*. These embeddings are continuous vector representations and take contextual information into account. As a consequence, the generation of flows is less error-prone as small variations in the embedding space generally do not change the outcome in input space much. For instance, if a GAN introduces a small error in *IP address* generation, it could find the embedding of the *IP address* 192.168.220.5 as nearest neighbor instead of the embedding of the expected *IP address* 192.168.220.13. Since both *IP addresses* are internal clients, the error has nearly no effect. As a consequence, *E-WGAN-GP* achieves the best results of the generative models in the evaluation. Yet, this approach (in contrast to *N-WGAN-GP* and *B-WGAN-GP*) cannot generate previously unseen values due to the embedding translation. This is not a problem for the attributes *bytes*, *packets* and *duration*. Given enough training data, embeddings for all (important) values of *bytes*, *duration* and *packets* are available. For example, consider the attribute *bytes*. We assume that the available embedding values $b_1, b_2, b_3, ..., b_{k-1}, b_k$ sufficiently cover the possible value range of the attribute *bytes*. As specific byte-values have no particular meaning, we are only interested in the magnitude of the attribute. Therefore, non existing values b_x can be replaced with available embedding values without adversely affecting the meaning.

The situation may be different for *IP addresses* and *ports*. *IP addresses* represent hosts with a distinct complex network behavior, for instance as a web server, printer, or Linux client. Generating new *IP addresses* goes along with the invention of a new host with new network behavior. To answer the question whether the generation of new *IP addresses* is necessary, the purpose needs to be considered in which the generated data shall be used later. If the training set comprises more than 10,000 or 100,000 different *IP addresses*, there is probably no need to generate new *IP addresses* for an IDS evaluation data set. However,

this does not hold generally. Instead, one should ask the following two questions: (1) are there enough different *IP addresses* in the training data set and (2) is there a need to generate previously unseen *IP addresses*? If previously unseen *IP addresses* are required, *E-WGAN-GP* is not suitable as transformation method, otherwise *E-WGAN-GP* will generate better flows than all other approaches.

The situation for *ports* is similar to *IP addresses*. Generally, there are 65536 different *ports* and most of these *ports* should appear in the training data set. Generating new *port* values is also associated with generating new behavior. If the training data set comprises SSH connections (port 22) and HTTP connections (port 80), but no FTP connections (port 20 and 21), generators are not able to produce realistic FTP connections if they have never seen such connections. Since the network behavior of FTP differs greatly from SSH and HTTP, it does not make much sense to generate unseen service *ports*. However, the situation is different for typical client *ports*.

Generally, GANs capture the implicit conditional probability distributions very well, given that a proper data representation is chosen which is the case for *E-WGAN-GP* and *B-WGAN-GP* (see Figures 6 and 7). While the visual differences between binary and embedded data representations are subtle, the *domain knowledge checks* show larger quality differences. Overall, this analysis suggests that *E-WGAN-GP* and *B-WGAN-GP* are able to generate good flow-based network traffic. While *E-WGAN-GP* achieves better evaluation results, *B-WGAN-GP* is not limited in the value range and is able to generate previously unseen values.

6. Related Work

This work targets the generation of flow-based network traffic using GANs. Therefore, we provide an overview of flow-based traffic generators before we review the general use of GANs in the application domain IT security.

6.1. Traffic Generators

Molnár et al. [35] give a comprehensive overview of network traffic generators, categorize them with respect to their purposes, and analyze the used validation measures. The authors conclude that there is no consensus on how to validate network traffic generators. Since the proposed approach aims to generate flow-based network traffic, the following overview considers primarily flow-based network traffic generators. Neither approaches which emulate computer networks and capture their network traffic like [14] or [42], nor static intrusion detection data sets like DARPA 98 or KDD CUP 99 will be considered in the following. We categorize flow-based network traffic generators into (I) *Replay Engines*, (II) *Maximum Throughput Generators*, (III) *Attack Generators*, and (IV) *High-Level Generators*.

Category (I). As the name suggests, *Replay Engines* use previously captured network traffic and replay the packets from it. Often, the aim of *Replay Engines* is to consider the original inter packet time (IPT) behavior between the network packets. TCPReplay [43] and TCPivo [44] are well-known representatives of this category. Since network traffic is subject to concept drift, replaying already known network traffic only makes limited sense for generating IDS evaluation data sets. Instead, a good network traffic generator for our purpose should be able to generate new synthetic flow-based network traffic.

Category (II). Maximum Throughput Generators usually aim to test end-to-end network performance [35]. Iperf [45] is such a generator and can be used for testing bandwidth, delay jitter, and loss ratio characteristics. Consequently, methods from this category primarily aim at evaluating network bandwidth performance.

Category (III). Attack Generators use real network traffic as input and combine it with synthetically created attacks. FLAME [46] is a generator for malicious network traffic. The authors use rule-based approaches to inject e.g. *port scan* attacks or *denial of service* attacks. Vasilomanolakis et al. [47] present ID2T, a

similar approach which combines real network traffic with synthetically created malicious network traffic. For creating malicious network traffic, the authors use rule-based scripts or manipulate parameters of the input network traffic. Sperotto et al. [48] analyze *ssh brute force* attacks on flow level and use a Hidden Markov Model to model the characteristics of them. However, their model generates only the number of *bytes*, *packets* and *flows* during a typical attack scenario and does not generate complete flow-based data.

Category (IV). High-Level Generators aim to generate new synthetic network traffic which contains realistic network connections. Stiborek et al. [37] propose three statistical methods to model host-based behavior on flow-based network level. The authors use real network traffic as input and extract typical inter- and intra-flow relations of host behavior. New flow-based data is generated based on a time variant joint probability model which considers the extracted user behavior. Siska et al. [38] propose a graph-based method to generate flow-based network traffic. The authors use real network traffic as input and extract traffic templates. Traffic templates are extracted for each service port (e.g. port 80 (HTTP) or 53 (DNS)) and contain structural properties as well as the value distributions of other flow attributes (e.g. log-normal distribution of transmitted *bytes*). These traffic templates can be combined with user-defined traffic templates. A flow generator selects flow attributes from the traffic templates and generates new network traffic. Iannucci et al. [39] propose PGPBA and PGSK, two synthetic flow-based network traffic generators. Their generators are based on the graph generation algorithms *Barabasi-Albert* (PGPBA) and *Kronecker* (PGSK). The authors initialize their graph-based approaches with network traffic in packet-based format. When generating new traffic, the authors first compute the probability of the attribute *bytes*. All other attributes of flow-based data are calculated based on the conditional probability of the attribute *bytes*. To evaluate the quality of their generated traffic, Iannucci et al. [39] analyze the degree and pagerank distribution of their graphs to show the veracity of the generated data.

The approach presented here does not simply replay existing network traffic like category (I). In fact, traffic generators from the first two categories have a different objective. Our approach belongs to category (IV) and generates new synthetic network traffic and is not limited to generating only malicious network traffic like category (III). While Siska et al. [38] and Iannucci et al. [39] use domain knowledge to generate flows by defining conditional dependencies between flow attributes, we use GAN-based approaches which learn all dependencies between the flow attributes inherently.

6.2. GANs

This section analyses how GANs were recently introduced in the domain IT security. A more general discussion about attacks and defenes for deep learning against adversarial examples may be found in Yuan et al. [49]. Rigaki and Garcia [50] use a GAN based approach to modify malware communication in order to avoid detection. The authors evaluate their method using an Intrusion Prevention System (IPS) which is based on a Markov model. The IPS considers the *bytes*, *duration* and *time-delta* of flows for determining malicious network traffic. Therefore, Rigaki and Garcia use a GAN which learns to imitate Facebook chat traffic characteristics based on these flow attributes. For capturing the *time-delta* of the flows, the generator and discriminator of the GAN are Recurrent Neuronal Networks (RNN). After the training phase, the authors use the GAN to generate legitimate Facebook traffic characteristics and adapt the malware to match these traffic patterns. Following this approach, the malware is able to successfully bypass the IPS. Hu and Tan [51] present a GAN based approach named MalGAN in order to generate synthetic malware examples which are able to bypass anomaly-based detection methods. Malware examples are represented as 160-dimensional binary attributes.

Anderson et al. [52] developed a character-based GAN to mimic domain generation algorithms (DGA) as used by malware to contact command and control servers. The authors train an auto-encoder to generate domain names and reassembled encoder and decoder to an adversarial setting fooling DGA-

detection classifiers. DeepDGA generates domain-names, which are categorical data, however as the domain-name is the only attribute generated, their setting is hardly comparable to our flow-based network data generation task.

Yin et al. [53] propose Bot-GAN, a framework which generates synthetic network data in order to improve botnet detection methods. However, their framework does not consider the generation of categorical attributes like *IP addresses* and *ports* which is one of the key contributions of our work.

Zheng et al. [54] use a generative adversarial network based approach for fraud detection in bank transfers. To be precise, the authors use a deep denoising autoencoder and two Gaussian Mixture Models (GMM). The encoder and one GMM act as discriminator and the decoder act as generator. The second GMM classifies the bank transfers in combination with a threshold into the classes *normal* or *fraud*. Zheng et al. achieve good results with this approach and are able to beat non GAN-based approaches. However, their input data differ significantly from flow-based data and consist primarily of continuous attributes like *amount of transferred money*, *balance of the account* or *frequency of transfers*.

For analysis of mobile traffic Zhang et al. [55] propose *ZipNet-GAN*, a GAN-based approach for fine-grained pattern extraction from coarse-grained network data, similar to super-resolution in image processing. Despite they are combining generative adversarial networks with the generation of fine-grained network traffic, their approach is very different to ours since they only work with traffic as aggregated continuous attribute, not with network data at flow-level and rely on coarse-grained information as input.

As can be seen in this section, GANs are already used in the domain IT security and prove their general suitability. However, existing works are only applied to specific application scenarios and consider only continuous attributes. In contrast to that, the proposed approach aims to generate network data in standard flow-based *NetFlow* format and considers all typical categorical attributes like *IP addresses* or *port numbers*.

7. Summary

Labeled flow-based data sets are necessary for evaluating and comparing anomaly-based intrusion detection methods. Evaluation data sets like DARPA 98 and KDD Cup 99 cover several attack scenarios as well as normal user behavior. These data sets, however, were captured at some point in time such that concept drift of network traffic causes static data sets to become obsolete sooner or later.

In this paper, we proposed three synthetic flow-based network traffic generators which are based on Improved Wasserstein GANs (WGAN-GP) [12] using the two time scale update rule from [13]. Our generators are initialized with flow-based network traffic and then generate new synthetic flow-based network traffic. In contrast to previous high-level generators, our GAN-based approaches learn all internal dependencies between attributes inherently and no additional knowledge has to be modeled. Flow-based network traffic consists of heterogeneous data, but GANs can only process continuous input data. To overcome this challenge, we proposed three different methods to handle flow-based network data. In the first approach *N-WGAN-GP*, we interpreted *IP addresses* and *ports* as continuous input values and normalized numeric attributes like *bytes* and *packets* to the interval $[0, 1]$. In the second approach *B-WGAN-GP*, we created binary attributes from categorical and numerical attributes. For instance, we converted *ports* to their 16-bit binary representation and extracted 16 binary attributes. *B-WGAN-GP* is able to maintain more information (e.g. subnet information of *IP addresses*) from the categorical input data. The third approach *E-WGAN-GP* learns meaningful continuous representations of categorical attributes like *IP addresses* using IP2Vec [11]. The preprocessing of *E-WGAN-GP* is inspired from the text mining domain which also has to deal with non-continuous input values. Then, we generated new flow-based network traffic based on the *CIDDS-001* data set [14] in an experimental evaluation. Our experiments indicate that especially *E-WGAN-GP* is able to generate realistic data which achieves good evaluation results. *B-WGAN-GP* achieves similarly

good results and is able to create new (unseen) values in contrast to *E-WGAN-GP*. The quality of network data generated by *N-WGAN-GP* is less convincing, which indicates that straight forward numeric transformation is not appropriate.

Our research indicates that GANs are well suited for generating flow-based network traffic. We plan to extend our approach in order to generate sequences of flows instead of individual flows. Therefore, we want to evaluate further network structures (e.g. LSTMs or CNN) which are able to learn temporal relationships of flow sequences. In addition, we want to work on the development of further evaluation methods.

Acknowledgments

M.R. was supported by the BayWISS Consortium Digitization. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] A. L. Buczak, E. Guven, A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, IEEE Communications Surveys & Tutorials 18 (2) (2016) 1153–1176.

[2] R. Sommer, V. Paxson, Outside the Closed World: On Using Machine Learning For Network Intrusion Detection, in: IEEE Symposium on Security and Privacy, IEEE, 2010, pp. 305–316.

[3] C. A. Catania, C. G. Garino, Automatic network intrusion detection: Current techniques and open issues, Computers & Electrical Engineering 38 (5) (2012) 1062–1072.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Advances in Neural Information Processing Systems (NIPS), 2014, pp. 2672–2680.

[5] A. Radford, L. Metz, S. Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, in: International Conference on Learning Representations (ICLR), 2016.

[6] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, W. Shi, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 105–114.

[7] P. Isola, J.-Y. Zhu, T. Zhou, A. A. Efros, Image-to-Image Translation with Conditional Adversarial Networks, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 5967–5976.

[8] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein Generative Adversarial Networks, in: International Conference on Machine Learning (ICML), 2017, pp. 214–223.

[9] L. Yu, W. Zhang, J. Wang, Y. Yu, SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient, in: Conference on Artificial Intelligence (AAAI), AAAI Press, 2017, pp. 2852–2858.

[10] K. Preuer, P. Renz, T. Unterthiner, S. Hochreiter, G. Klambauer, Fréchet ChemblNet Distance: A metric for generative models for molecules, CoRR abs/1803.09518.
URL `http://arxiv.org/abs/1803.09518`

[11] M. Ring, D. Landes, A. Dallmann, A. Hotho, IP2Vec: Learning Similarities between IP Adresses, in: Workshop on Data Mining for Cyber Security (DMCS), International Conference on Data Mining, IEEE, 2017, pp. 657–666.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. C. Courville, Improved Training of Wasserstein GAN, in: Advances in Neural Information Processing Systems (NIPS), 2017, pp. 5769–5779.

[13] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, in: Advances in Neural Information Processing Systems (NIPS), 2017, pp. 6629–6640.

[14] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho, Flow-based benchmark data sets for intrusion detection, in: European Conference on Cyber Warfare and Security (ECCWS), ACPI, 2017, pp. 361–369.

[15] M. M. Najafabadi, T. M. Khoshgoftaar, C. Kemp, N. Seliya, R. Zuech, Machine Learning for Detecting Brute Force Attacks at the Network Level, in: IEEE International Conference on Bioinformatics and Bioengineering (BIBE), IEEE, 2014, pp. 379–385.

[16] M. M. Najafabadi, T. M. Khoshgoftaar, A. Napolitano, C. Wheelus, RUDY Attack: Detection at the Network Level and Its Important Features, in: International Florida Artificial Intelligence Research Society Conference, 2016, pp. 288–293.

[17] Q. A. Tran, F. Jiang, J. Hu, A real-time netflow-based intrusion detection system with improved BBNN and high-frequency field programmable gate arrays, in: International Conference on Trust, Security and Privacy in Computing and Communications, IEEE, 2012, pp. 201–208.

[18] C. Wagner, J. François, T. Engel, et al., Machine Learning Approach for IP-Flow Record Anomaly Detection, in: International Conference on Research in Networking, Springer, 2011, pp. 28–39.

[19] V. L. Cao, M. Nicolau, J. McDermott, A Hybrid Autoencoder and Density Estimation Model for Anomaly Detection, in: International Conference on Parallel Problem Solving from Nature, Springer, 2016, pp. 717–726.

[20] S. Garcia, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, Computers & Security 45 (2014) 100–123.

[21] M. Ring, D. Landes, A. Hotho, Detection of slow port scans in flow-based network traffic, PLOS ONE 13 (9) (2018) 1–18. doi:10.1371/journal.pone.0204507.

[22] B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954 (2004).

[23] B. Claise, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, RFC 5101 (2008).

[24] J. Han, J. Pei, M. Kamber, Data Mining: Concepts and Techniques, 3rd Edition, Elsevier, 2011.

[25] E. B. Beigi, H. H. Jazi, N. Stakhanova, A. A. Ghorbani, Towards Effective Feature Selection in Machine Learning-Based Botnet Detection Approaches, in: IEEE Conference on Communications and Network Security, IEEE, 2014, pp. 247–255.

[26] M. Stevanovic, J. M. Pedersen, An analysis of network traffic classification for botnet detection, in: IEEE International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), IEEE, 2015, pp. 1–8.

[27] R. Salakhutdinov, H. Larochelle, Efficient learning of deep Boltzmann machines, in: International Conference on Artificial Intelligence and Statistics, 2010, pp. 693–700.

[28] I. Goodfellow, NIPS 2016 tutorial: Generative Adversarial Networks, arXiv preprint arXiv:1701.00160.

[29] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, arXiv preprint arXiv:1301.3781.

[30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed Representations of Words and Phrases and their Compositionality, in: Advances in Neural Information Processing Systems (NIPS), 2013, pp. 3111–3119.

[31] N. Buduma, N. Locascio, Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms, O'Reilly Media, 2017.

[32] A. Borji, Pros and Cons of GAN Evaluation Measures, arXiv preprint arXiv:1802.03446.

[33] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved Techniques for Training GANs, in: Advances in Neural Information Processing Systems (NIPS), 2016, pp. 2234–2242.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the Inception Architecture for Computer Vision, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

[35] S. Molnár, P. Megyesi, G. Szabo, How to Validate Traffic Generators?, in: IEEE International Conference on Communications Workshops (ICC), IEEE, 2013, pp. 1340–1344.

[36] J. Glasser, B. Lindauer, Bridging the gap: A pragmatic approach to generating insider threat data, in: Security and Privacy Workshops (SPW), IEEE, 2013, pp. 98–104.

[37] J. Stiborek, M. Rehák, T. Pevnỳ, Towards scalable network host simulation, in: International Workshop on Agents and Cybersecurity, 2015, pp. 27–35.

[38] P. Siska, M. P. Stoecklin, A. Kind, T. Braun, A Flow Trace Generator using Graph-based Traffic Classification Techniques, in: International Wireless Communications and Mobile Computing Conference (IWCMC), ACM, 2010, pp. 457–462. doi:10.1145/1815396.1815503.

[39] S. Iannucci, H. A. Kholidy, A. D. Ghimire, R. Jia, S. Abdelwahed, I. Banicescu, A Comparison of Graph-Based Synthetic Data Generators for Benchmarking Next-Generation Intrusion Detection Systems, in: IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2017, pp. 278–289.

[40] J. Sommers, P. Barford, Self-Configuring Network Traffic Generation, in: ACM Internet Measurement Conference (ACM IMC), ACM, 2004, pp. 68–81.

[41] A. Botta, A. Dainotti, A. Pescapé, A tool for the generation of realistic network workload for emerging networking scenarios, Computer Networks 56 (15) (2012) 3531–3547.

[42] A. Shiravi, H. Shiravi, M. Tavallaee, A. A. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Computers & Security 31 (3) (2012) 357–374.

[43] A. Turner, Tcpreplay, (Date last accessed 14-June-2018).
URL `http://tcpreplay.synfin.net/`

[44] W.-c. Feng, A. Goel, A. Bezzaz, W.-c. Feng, J. Walpole, TCPivo: A High-Performance Packet Replay Engine, in: ACM Workshop on Models, Methods and Tools for Reproducible Network Research, ACM, 2003, pp. 57–64.

[45] D. Jon, E. Seth, M. A. Bruce, P. Jeff, P. Kaustubh, Iperf: The TCP/UDP bandwidth measurement tool, (Date last accessed 14-June-2018).
URL `https://iperf.fr/`

[46] D. Brauckhoff, A. Wagner, M. May, FLAME: A Flow-Level Anomaly Modeling Engine, in: Workshop on Cyber Security Experimentation and Test (CSET), USENIX Association, 2008, pp. 1:1–1:6.

[47] E. Vasilomanolakis, C. G. Cordero, N. Milanov, M. Mühlhäuser, Towards the creation of synthetic, yet realistic, intrusion detection datasets, in: IEEE Network Operations and Management Symposium (NOMS), IEEE, 2016, pp. 1209–1214.

[48] A. Sperotto, R. Sadre, P.-T. de Boer, A. Pras, Hidden Markov Model modeling of SSH brute-force attacks, in: International Workshop on Distributed Systems: Operations and Management, Springer, 2009, pp. 164–176.

[49] X. Yuan, P. He, Q. Zhu, R. R. Bhat, X. Li, Adversarial Examples: Attacks and Defenses for Deep Learning, arXiv preprint arXiv:1712.07107.

[50] M. Rigaki, S. Garcia, Bringing a GAN to a Knife-fight: Adapting Malware Communication to Avoid Detection, in: 1st Deep Learning and Security Workshop, San Francisco, USA, 2016.

[51] W. Hu, Y. Tan, Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN, arXiv preprint arXiv:1702.05983.

[52] H. S. Anderson, J. Woodbridge, B. Filar, Deepdga: Adversarially-tuned domain generation and detection, in: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, ACM, 2016, pp. 13–21.

[53] C. Yin, Y. Zhu, S. Liu, J. Fei, H. Zhang, An Enhancing Framework for Botnet Detection Using Generative Adversarial Networks, in: International Conference on Artificial Intelligence and Big Data (ICAIBD), 2018, pp. 228–234. doi:10.1109/ICAIBD.2018.8396200.

[54] Y.-J. Zheng, X.-H. Zhou, W.-G. Sheng, Y. Xue, S.-Y. Chen, Generative adversarial network based telecom fraud detection at the receiving bank, Neural Networks 102 (2018) 78–86.

[55] C. Zhang, X. Ouyang, P. Patras, Zipnet-gan: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network, in: Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, ACM, 2017, pp. 363–375.