POLYTECHINC OF TURIN

Master’s Degree in Computer Engineering

Master’s Degree Thesis

GAN-based black box evasion attack against a machine learning botnet detection system

Supervisors

Prof. Andrea BOTTINO

Prof. Alberto MOZO VELASCO

Candidate

Dario FERRANDINO

October 2022

Summary

Botnets have become an important issue in the domain of computer and network security as they serve as platform of many threats such as spam, denial of service attacks, phishing, data thefts, and online frauds. Most of the popular botnet detection methods consist of monitoring the network, capturing packets, processing them in a format suitable for botnet detection, inspecting the network flows and detecting malicious traffic. State of the art Botnet Detectors exploit machine learning techniques to detect malicious traffic flows. Usually, the structure and the internal parameters of these models are not observable from the outside and, therefore, they are considered black-box models. Recently, several Generative Adversarial Network (GAN) based frameworks, which can successfully generate adversarial malicious traffic flows examples to fool Intrusion Detection Systems (IDSs), have been proposed. The purpose of this work is to successfully train a Generative Adversarial Network to significantly reduce the probability that the generated adversarial malicious traffic flow examples are detected as effective botnet attacks from any of the proposed black-box botnet detector using an architecture which can handle the generation of both numerical and categorical variables. The aim is to evaluate the improvement that this approach can make over previous proposed frameworks. The proposed attack frameworks are based on Wasserstein GAN (WGAN) and Wasserstein GAN with Gradient Penalty (WGAN-GP), due to their improved stability during training over the original GAN designed by Goodfellow et al. In general, the architecture consists of a Generator which transforms the original malicious traffic flow examples into adversarial malicious ones, and a Critic which tries to learn the black-box botnet detector. In another version of the attack, the Critic behaves as a standard WGAN-GP critic to analyze the behaviour of such an attack. In order to ensure the validity of the generated malicious examples, only the non-functional features of the attack examples are modified. Using the CTU-13 dataset, successful attacks have been performed on several types of black-box botnet classifiers, achieving excellent results.

Acknowledgements

Before any work, I have the honor to express my gratitude to all those who contributed to the realization of my work.
This document has been drawn up during a 10 months Erasmus program at the Polytechnic University of Madrid.
It was a new and exciting experience made possible thanks to the support of my family which has always sustained my academic demands and deeply believe in the value of education.
I would like to thank the Polytechnic University of Madrid for having accepted me as a student within their school.
I want to express my heartfelt thanks to "Mr. Alberto Mozo", who trained and accompanied me throughout this work.
I also thank "Mr. Andrea Bottino", teacher at the Polytechinic University of Turin, who remotely followed the progresses of the work.
I would like to thank my university's colleagues for their friendship and the constant support which has made the academic path easier and funnier.
Finally a special thanks to the Polytechinc of Turin which has always made available all the resources I needed.

Table of Contents

1 Introduction 2

1.1 Preface . . . 2

2 Goals and Requirements 4

2.1 Objectives and methodology . . . 4

2.2 Document Structure . . . 4

2.2.1 Theoretical foundations . . . 4

2.2.2 Design principles and Implementation . . . 5

2.2.3 Practical attempts . . . 5

3 State of The Art 6

3.1 Botnets . . . 6

3.1.1 Botnet Detection Techniques . . . 6

3.2 Adversarial Machine Learning . . . 7

3.2.1 Evasion Attacks . . . 8

3.3 Generative Adversarial Networks . . . 8

3.3.1 Original GAN . . . 9

3.3.2 Wasserstein GAN . . . 10

3.4 Wasserstein GAN with gradient penalty . . . 11

3.5 Gumbel-Softmax . . . 12

3.5.1 Reparametrization Trick . . . 13

3.5.2 Gumbel-Softmax distribution . . . 13

3.5.3 Straight-Through Gumbel-Softmax Estimator . . . 15

3.6 Related Works . . . 15

4 Design 16

4.1 Assumptions . . . 17

4.2 Models with substitute detectors . . . 17

4.3 Model without substitute detector . . . 17

4.4 Critic and Generator networks description . . . 20

4.4.1 Critic . . . 21

4.4.2 Generator . . . 21

5 Development **24**
5.1 Dataset . . . 24
5.1.1 Black-box and GAN datasets . . . 24
5.2 Models implementation . . . 28
5.2.1 Attacks with Substitute Detector . . . 28
5.2.2 Attacks without Substitute Detector . . . 28
5.2.3 Generator . . . 29
5.2.4 Critic . . . 30

6 Experiments **32**
6.1 Hardware . . . 32
6.2 Tools . . . 32
6.3 Black-box Botnet Detectors . . . 32
6.4 Attacks Evaluation Metrics . . . 34
6.5 Results . . . 35
6.5.1 Evasion Rate . . . 35
6.5.2 Transferability Analysis . . . 36
6.5.3 Minkowski Distance of the non-functional features . . . 37
6.5.4 Evaluation of the improvement in modifying also categorical features . . . 39

7 Conclusions **43**
7.1 Success of the attacks . . . 43
7.1.1 Models with substitute detector . . . 43
7.1.2 Model without substitute detector . . . 44
7.2 Improvement of perturbing categorical features . . . 44
7.3 Efficacy of the Straight-Trough Gumbel-Softmax Estimator . . . 44
7.4 Reproducibility of the adversarial examples . . . 45

8 Future Works **46**

A Tensorflow main components implementation **47**
A.1 Gumbel-Softmax Layer . . . 47
A.2 Generator . . . 49
A.3 GAN model class with substitute detector . . . 50

Bibliography **54**

List of Tables

5.1 CTU-13 feature description 25
5.2 Number of examples from each scenario for Black-box and GAN datasets 26
5.3 GAN dataset Train Test split 27
5.4 Generator Model - GAN with substitute detector and softmax activation for categorical feature generation 29
5.5 Generator Model - GAN with substitute detector and Gumbel-Softmax activation for categorical feature generation 30
5.6 Generator Model - GAN without substitute detector and Gumbel-Softmax activation for categorical feature generation 30
5.7 Critic Model 31

6.1 Evaluation Metrics scores for Black-box classifiers on Black-Box test set 33
6.2 Evasion Rate scores on GAN test set 35
6.3 Evasion Rate Scores Transferability Analysis 37
6.4 Evasion rate comparison between perturbing both numerical and categorical features and only numerical features 39

List of Figures

3.1 Botnet scheme 7
3.2 Plot of categorical and continuous distributions 12
3.3 Reparametrization Trick 14
3.4 Temperature, τ, effects on the Gumbel-Softmax distribution (image from [18]) 14

4.1 GAN with substitute detector - Training Scheme 20
4.2 GAN without substitute detector - Training Scheme 20
4.3 Generator scheme: the figure shows the architecture of generators version 1 and 2. Version 3 takes as input only the noise vector. . . . 23

6.1 Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 1 40
6.2 Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 2 41
6.3 Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 3 42

List of Equations

Eq. 3.1 Original GAN objective function 9
Eq. 3.2 Original GAN Generator Loss under discriminator optimality 9
Eq. 3.3 Alternative original GAN Generator Loss 9
Eq. 3.4 Earth Mover Distance 10
Eq. 3.5 Kantorovic-Rubenstein transformation of Wasserstein Distance 10
Eq. 3.6 K-Lipschitz continuity constraint 10
Eq. 3.7 Wasserstein GAN objective function 11
Eq. 3.8 WGAN generator loss function 11
Eq. 3.10 WGAN-GP critic loss function 12
Eq. 3.11 Reparametrization Trick example 13
Eq. 3.13 Gumbel-Softmax 13
Eq. 4.1 GAN with substitute detector - Critic Loss 21
Eq. 4.2 GAN without substitute detector - Critic Loss 21
Eq. 4.3 Solution's Generator Loss 22
Eq. 5.1 Black box dataset numerical variables standardization formula 26
Eq. 5.2 GAN dataset numerical variables standardization formula . 27
Eq. 6.1 Black-box classifiers evaluation metrics 33

Eq. 6.2 Evasion Rate formula 34
Eq. 6.3 Minkowski Distance 34
Eq. 6.4 Minkowski distance between a point and a Set 35

Chapter 1

Introduction

1.1 Preface

Machine learning and artificial intelligence technologies have been growing exponentially in the last decades, changing substantially the way humans interacts with machines and overcoming the performances of the previous state-of-the-art methods in many tasks and fields, but carrying with them new vulnerabilities and threats that were previously unknown.

Adversarial machine learning is the study of the attacks on machine learning algorithms, and of the defenses against such attacks[1].

As much as new learning algorithms have been explored and proved to be extremely effective and reliable for several application, many researchers have demonstrated different ways in which it is possible to fool them or change their behaviour, indeed, such algorithms manifest vulnerabilities and little robustness against adversarial examples[2][3][4][5].

Machine learning is quickly taking on a key role in organizations value, and as a result, companies which need to safeguard them is expanding quickly. Adversarial Machine Learning is thus growing significantly within the software sector. Machine learning systems are now being secured thanks to investments made by Google, Microsoft, and IBM.

The most common and most studied forms of adversarial attacks are evasion attacks. During deployment, the attacker modifies the data to trick classifiers that have already been trained. They are the most common types of assaults employed in intrusion and malware scenarios since they are carried out during the deployment phase. Samples are modified as much as needed to avoid the detection, without directly affecting the training data.

The tools that have recently attracted many researchers for their astonishing results, especially in the image generation field, are the **Generative Adversarial**

Networks. A GAN is a framework in which two neural network, namely a generator and a discriminator, compete to improve their ability to generate fake data which looks real, and to discriminate among real and fake data, respectively. Among the many applications and fields in which these models can be employed, they result an optimal tool to craft adversarial examples for black-box evasion attacks.

Chapter 2

Goals and Requirements

2.1 Objectives and methodology

The first objective of this work was to describe and produce a Generative Adversarial Network based model which was able to generate adversarial malicious network flow examples to fool and evade different types of black-box machine learning botnet detection systems.
In order to do this, different types of machine learning classifiers were trained and tested and, then, three types of GAN based frameworks were designed to fool the black-box classifiers. The three types of developed frameworks slightly change one from another but all of them is specifically designed to handle the generation of both numerical and categorical tabular data. The main objective of each attack framework is to fool as much as possible, the target black-box classifier but the transferability of each attack is also assessed in order to observe how much the evasion attack on a specific black-box model can be generalized to other, unknown, models that have been trained on the same data distribution. Finally the similarity between the adversarial malicious examples and the original benign and malicious sets is evaluated to assess how much the considered frameworks are able to retain the appearance of the malicious examples on the crafted ones. The generated examples are crafted from the malicious examples present in the CTU-13 dataset, which will be further described and analyzed in chapter 5.1.

2.2 Document Structure

2.2.1 Theoretical foundations

This document starts by explaining what botnets are and by touching the very basic principles of adversarial machine learning. Moreover, it provides some basic information about the principles of some of the most known types of GANs. In the end of this macro-chapter, a brief definition of the Gumbel-Softmax distribution, for which will be reserved particular attention for the generation of categorical features, will be provided.

2.2.2 Design principles and Implementation

Following the theoretical part, a general overview of the design of the attack frameworks will be illustrated to understand the behaviour and the choices made in this phase; then will follow the description of the dataset used to train both the attack frameworks and the black-box classifier together with the pre-processing operations made before starting the training of the models. After this, the implementation details of the attack frameworks will be shown together with the description of the dataset used and the training specifics of the black-box classifiers.

2.2.3 Practical attempts

Finally, after having trained the three generators the evasion rate, the transferability and the similarity to the original examples of the adversarial examples will be assessed and discussed to extract some useful conclusion. Moreover, we will also evaluate the improvement that the perturbation of categorical features can bring to black-box evasion attacks with respect to the perturbation of only numerical features.

Chapter 3

State of The Art

The following chapter will illustrate and explain the main concepts related to the work as it is meant to provide a baseline to better understand the development and the context of the research.

3.1 Botnets

Any program which performs any type of activity which aims to damage the system of a user in an automated way without the user's knowledge is identified as a bot. The network made up by all the bots controlled by a botmaster is referred as a botnet. Botnets are often made up of a large number of hosts that are controlled by a remote attacker. The attacker controls the hosts by communicating with the botnets via a Command & Control (C&C) communication channel. This distinguishes the botnet from other types of malware. For C&C communication, the attacker may employ different kind of communication protocols such as HTTP, IRC, etc. The main challenge in defending against such attacks comes precisely from the huge number of different machines that are involved in the attack which makes difficult the task to block malicious traffic without accidentally blocking genuine requests.

3.1.1 Botnet Detection Techniques

Several botnet detection methods have been proposed with the aim to detect infected hosts and defend against network attacks. One of the techniques to detect botnet attacks is the signature-based Botnet detection technique. This approach tries to detects Botnets by analyzing their signatures. Signature-based detection has several advantages, including a low false alarm rate, fast detection, ease of implementation, and more information about the type of detected attack, but the

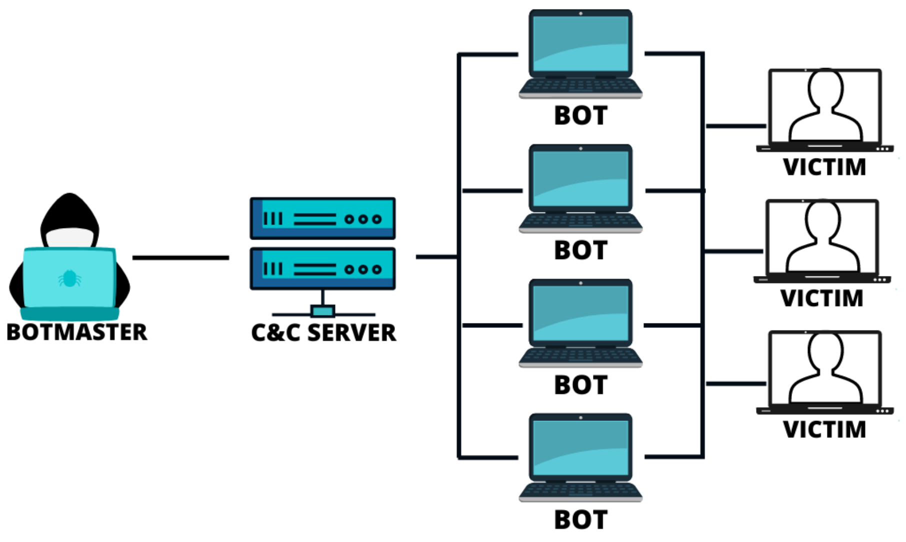


Figure 3.1: Botnet scheme

main drawback of such a method is that it can only detect well-known botnets and fail to detect new type of threats.

Another method is based on inspecting features from the packets, specifically, the payload of targeted packets in order to detect anomalies and malicious behaviour. However, as botnet can encrypt the payload, this approach has obvious limitation and, moreover, inspecting the payload of certain packets can be computationally expensive and can cause delays in high-speed networks[6][7].

An effective approach to detect botnets is the flow-based detection method which relies only on the information of packets headers which are aggregated into flows to compute useful statistics. Since the nature of the problem falls into the category of classification tasks, many machine learning algorithms such as K-NN, decision trees, SVM, Deep Neural Networks and others have been applied together with the flow-based detection method to recognize botnet activities and demonstrated very important results[8][9].

3.2 Adversarial Machine Learning

The study of attacks on machine learning algorithms, as well as the defenses against such attacks, is known as adversarial machine learning.

3.2.1 Evasion Attacks

In adversarial machine learning, an evasion attack focuses on finding the vulnerabilities of a machine learning classifier, exploiting them to bypass the model detection. Evasion attacks do not operate on training data but they rather consist in modifying or crafting a given example to let it be misclassified by the detector. For instance, as pointed out by Dalvi et al. in [10], a linear classifier for spam detection can be easily fooled by adding specific words in the text.

Adversarial Example
An adversarial example is a crafted example that have been purposefully design by an attacker to force a machine model to misclassify it. Often adversarial examples are either modelled by a noise distribution or by adding small perturbations into the original example. The perturbation can be imperceptible and the modified input can still clearly look as malign to human eyes while the classifier cannot recognize it.

White Box Attacks
White-box attacks are a type of evasion attacks in which the attacker is supposed to have access to the machine learning model's internals, which means that the architecture, the number and the values of the parameters are known to the attacker. Moreover, the intruder is supposed to be able to get the model's predictions for any provided input example.

Black Box Attacks
In adversarial machine learning, black box attacks are another type of evasion attacks that apply more constraint to the abilities of the attacker. They presume that an intruder who wants to evade the detection of a machine learning classifier can only get the target model's predictions for given inputs and has no knowledge about the model structure and its internal parameters. The adversarial example is constructed, in this case, either with a model developed from scratch or without any model at all. In any case, the goal of this approach is to generate adversarial examples that can be transferred to the targeted black box model.

3.3 Generative Adversarial Networks

Generative Adversarial Networks are a class of machine learning models made up by a generator and a discriminator (or critic in Wasserstein GANs) which compete one against the other in order to eventually generate new synthetic data which has the same statistics of real data. The essential concept of a GAN is the indirect training through the discriminator, which is another neural network that can determine how realistic an input is and whose weights are updated throughout the training phase as well. This essentially implies that the generator is taught to deceive the discriminator rather try to reduce the distance between the generated output and a given example. This allows the model to learn in an unsupervised manner.

3.3.1 Original GAN

The original GAN designed by Godfellow et al.[11] is based on a minimax game in which the generator and the discriminator are trained to optimize the following objective function:

$$\begin{aligned}\min_G \max_D L(G, D) &= \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[logD(x)] + \mathop{\mathbb{E}}_{z\sim\mathbb{P}_z}[log(1 - D(G(z)))] \\ &= \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[logD(x)] + \mathop{\mathbb{E}}_{x\sim\mathbb{P}_g}[log(1 - D(x))]\end{aligned} \tag{3.1}$$

where $\mathbb{P}_r$ and $\mathbb{P}_g$ are respectively the data distribution of real examples and of the generator, $\mathbb{P}_z$ is the noise distribution and g_θ is the output of the generator. The discriminator is a normal classification model which outputs the probability in the range [0,1] of an input to come from the real distribution $\mathbb{P}_r$. The paper shows the proof that, when the discriminator is optimal, the generator optimizes:

$$\min_G L(G, D) = -\log(4) + 2JSD(p_{data}||\mathbb{P}_g) \tag{3.2}$$

which can be considered as optimizing the Jensen-Shannon divergence between the generative data distribution and the real sample distribution. While the quality of the generated samples of the original GAN are often regarded as the best, compared to other generative models, many researches have shown mathematical and empirical proofs that training a GAN by optimizing the objective function 3.1 is not an easy task and can often lead to training failures due to problems such as vanishing gradient [12]. In fact, as pointed out by Arjovsky and Bottou [13], as the discriminator gets better, the gradient with respect to the generator vanishes, meaning that, since the the approximation of the generator's loss to eq. (3.2) depends on the discriminator's optimality, this can lead to either inaccurate or vanishing gradients.

Due to vanishing gradients related with the loss function of the generator, the authors of the original GAN suggest to try to train the generator to optimize:

$$\min_G L(G, D) = -log(D(G(z)) \tag{3.3}$$

which can avoid the vanishing gradient problem, especially in the early stages of the learning process of the generator. However, Arjovsky and Bottou show that also this loss function is not optimal since it produces gradients with high norm and high variance as discriminator is trained to optimatily, which can lead to unstable behaviour and slower convergence.

3.3.2 Wasserstein GAN

Wasserstein GAN[14] was proposed by Arjovsky, et al. to improve the training of Generative Adversarial Networks by means of a new cost function based on a different distance metric, the Earth Mover Distance (EMD). Considering two data distributions, p and q, as two amounts of dirt, the EMD, also named Wasserstein distance, measures the minimum amount of work nedeed to transport the mass of the data distribution p to the data distirbution q.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathop{\mathbb{E}}_{(x,y)\sim\gamma}[||x - y||] \tag{3.4}$$

"*where* $\prod(\mathbb{P}_r, \mathbb{P}_g)$ *denotes the set of all joint distributions (x, y) whose marginals are respectively Pr and Pg. Intuitively,*$\gamma(x, y)$ *indicates how much mass must be transported from x to y in order to transform the distributions* $\mathbb{P}_r$ *into the distribution* $\mathbb{P}_g$"[14].

So, the Wasserstein distance between two data distribution $\mathbb{P}_r$ (real) and $\mathbb{P}_g$ (generated) is the infimum among all the ways. $\prod(\mathbb{P}_r, \mathbb{P}_g)$, to transport the data.

Since computing $\prod(\mathbb{P}_r, \mathbb{P}_g)$ is highly intractable, the authors proposed a transformation based on the Kantorovich-Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{||f||_L \leq 1} \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[f(x)] - \mathop{\mathbb{E}}_{x\sim\mathbb{P}_g}[f(x)] \tag{3.5}$$

For the transformation to work, f is demanded to be the supremum over all the f that are 1-Lipschitz continuous.
In general, a function is K-Lipschitz continuous if and only if exists a real constant, $K \geq 0$, such that, for all $x_1, x_2 \in \mathbb{R}$:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \tag{3.6}$$

Which means that the derivative of the function, f, is less than or equal to K everywhere.
If the function f is not 1-Lipschitz continuous but K-Lipschitz continuous, the supremum becomes $K * W(\mathbb{P}_r, \mathbb{P}_g)$. So, if f is truly the supremum, the process will, up to a multiplicative factor, compute the Wasserstein distance. However, there's

no need to know what is the value of K since it will be absorbed in the process of hyperparameter tuning. Finding the supremum over 1-Lipschitz functions is still intractable, but, in the new form expressed in 3.5, it is easier to approximate.

In WGAN the discriminator is renamed to "critic" since it doesn't outputs probabilities anymore, but scores in the range $[-inf, +inf]$. The critic is trained to learn the function, f_w, which is parametrized with respect to its weights and demanded to be K-Lipschitz continuous.

The critic is trained to solve the problem

$$\max_{w \in W} \mathop{\mathbb{E}}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathop{\mathbb{E}}_{z \sim Z}[f_w(g_\theta(z))] \tag{3.7}$$

to measure the Wasserstein distance between $\mathbb{P}_r$ and $\mathbb{P}_g$.

After training the critic with a fixed set of generator weights θ to well approximate $W(\mathbb{P}_r, \mathbb{P}_\theta)$, the weights of the generator are updated to minimize the distance between the two distributions:

$$\min_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = \nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = - \mathop{\mathbb{E}}_{z \sim Z}[\nabla_\theta f_w(g_\theta(z)] \tag{3.8}$$

K-Lipschitz continuity is enforced by clipping the value of the weights between a certain range $[-c, +c]$ and, as stated by the authors of the paper, it is an easy but terrible way to enforce the constraint. Even if the Wasserstein GAN converges when the critic is trained until Compared to the Jensen-Shannon distance, Wasserstein distance is almost differentiable everywhere which allows the discriminator to be trained to optimality avoiding vanishing gradients and can mitigate the problem of mode collapse when used as objective function.

3.4 Wasserstein GAN with gradient penalty

An alternative to weight clipping to enforce the constraint of K-Lipschitz continuity was proposed by Gulrajani et al. [15] based on adding a gradient penalty regularization term in the cost function of the discriminator. The intuition behind this method is that, given two distributions $\mathbb{P}_r$ $\mathbb{P}_g$ in a compact metric space X, then there exists a 1-Lipschitz function f^* which is the optimal solution of (3.5) and f^* has gradient norm 1 almost everywhere. Specifically, they prove that points interpolated between the real and generated data should have gradient norm of 1 for f [16].

if f^* is differentiable, $\pi(x = y) = 0$, and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$, then

$$\mathbb{P}_{(x,y)\sim\pi}[\nabla f^*(x_t) = \frac{y - x_t}{||y - x_t||}] = 1 \tag{3.9}$$

The new constraint is enforced in a soft way: since enforcing gradient norm at most 1 everywhere is intractable, the authors propose to enforce it only on the straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$.

$$L = \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)] + \lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{3.10}$$

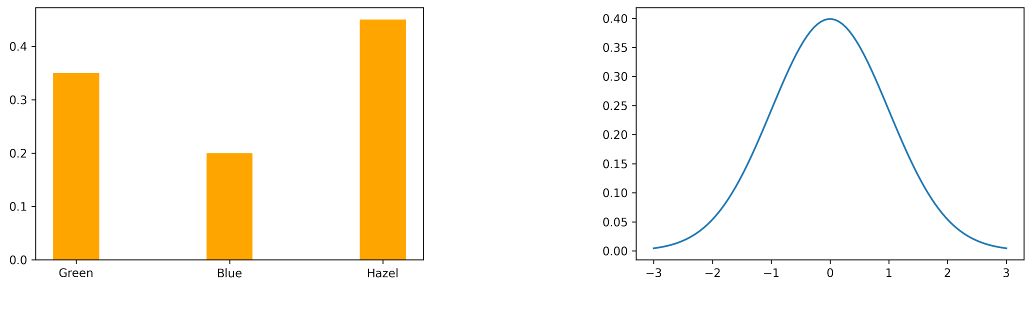
where λ is the penalty coefficient and is an hyperparameter and $\hat{x}$ is defined as

$$\hat{x} = t\tilde{x} + (1-t)x \qquad \text{with } t \text{ sampled from a uniform distribution } U[0,1]$$

It has been demonstrated that clipping weights to enforce a K-Lipschitz constraint biases the critic limiting its capacity of learning complex functions, while, despite the simplification, enforcing the constraint through gradient penalty experimentally results in better performance.

3.5 Gumbel-Softmax

The process of generating samples of categorical variables in stochastic neural networks comes with the main issue of sampling from a categorical distribution in a differentiable way, in order to be able to back-propagate the gradient through the previous layers. The problem arise from the fact that generative networks model noise from a continuous distribution to produce an example, but the possible values of a categorical distribution are discrete as shown in Fig. 3.2 and, therefore, there is the need to introduce a smooth function that enables to approximate the sampling process from a categorical distribution in such a way that the back-propagation algorithm can still work and it is possible to compute a gradient.



(a) Categorical distribution **(b)** Continuous distribution

Figure 3.2: Plot of categorical and continuous distributions

3.5.1 Reparametrization Trick

Drawing a sample from a continuous distribution is an issue when it comes to deal with the stochasticity of the sampling process. The gradient cannot be computed directly on a stochastic node which samples from a distribution. Instead, the Reparametrization trick allows to cast the stochastic process to a linear combination of deterministic and stochastic elements and back-propagate through the deterministic ones[17].

For instance, let z be a point sampled from a normal distribution, $N(\mu, \sigma)$, it is not possible to differentiate a random variable because the parameter we want to differentiate by parametrize also the density of the distribution. What is better, is to extrapolate the parameters that describe the random variable, and differentiate with respect to them:

$$\begin{gathered}\text{instead of considering z} \sim N(\mu, \sigma) \\ \text{let } \epsilon \sim N(0{,}1) \\ z = \mu + \sigma * \epsilon\end{gathered} \tag{3.11}$$

The example in (3.11) shows a specific application of the reparametrization trick, but what is important is that, moving out of the stochastic node the parameters which we want to differentiate by, the behaviour of the model does not change and the gradient can flow through the previous layers. Figure 3.3 visually shows what happens when applying the reparametrization trick in a network.

3.5.2 Gumbel-Softmax distribution

The Gumbel-Softmax distribution was introduced independently, in 2016, by two research teams [18][19]. It is a continuous distribution which enables to approximate samples from a categorical distribution. Let Z be a categorical variable with class probabilities $\pi_1, \pi_1, ..., \pi_k$. Gumbel-Max trick allows to draw samples from Z:

$$z = one\_hot(\arg\max_i [g_i + \log \pi_i]) \tag{3.12}$$

where $g_1, g_2, ..., g_k$ are i.i.d. samples drawn from Gumbel(0,1). Instead of using the argmax function which is not differentiable, we can approximate it with the softmax function which is continuous and differentiable [18].

$$y_i = \frac{e^{\frac{\log(\pi_i)+g_i}{\tau}}}{\sum_{j=1}^{k} e^{\frac{\log(\pi_j)+g_j}{\tau}}} \tag{3.13}$$

Eq (3.13) represents a k-dimensional sample vector from the Gumbel-Softmax distribution. In other words, it only approximates the argmax with a differentiable

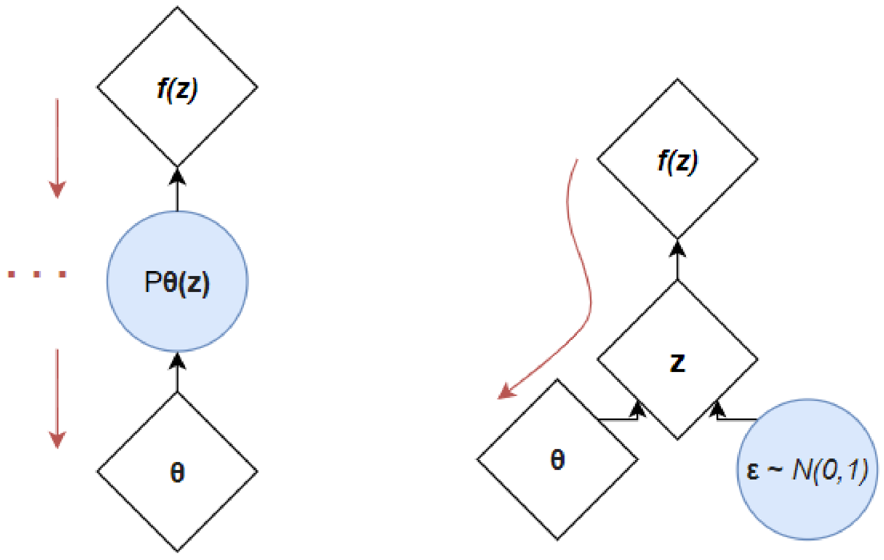


Figure 3.3: Reparametrization Trick

Inspiered by [18]. On the left a network in which the sample is build directly through a stochatstic node, the back-prop algorithm cannot work since the node is not differentiable. On the right the reparametrization trick is applied and the gradient can flow through the set of parameter θ

function which is the softmax one[20]. As $\tau \to 0$ samples from this distribution become one-hot, whereas as $\tau \to \infty$ the distribution of probabilities becomes more uniformly distributed over all the classes.

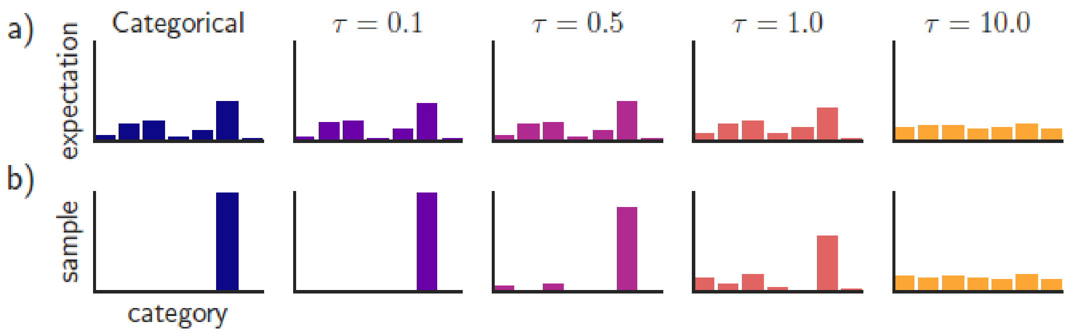


Figure 3.4: Temperature, τ, effects on the Gumbel-Softmax distribution (image from [18])

3.5.3 Straight-Through Gumbel-Softmax Estimator

Since using continuous approximations during the training of a network and using one-hot vectors during evaluation can bias the training process and can lead to a significant drop in performance evaluation, in the forward step we can hard-encode the soft-encoded vectors, produced by the gumbel softmax activation function, and back-propagate through the soft-encoded ones which are differentiable, to approximate the gradient. This technique is known as Straight Through Gumbel Estimator. [21]

3.6 Related Works

Researchers have shown great interest in the Adversarial Black-Box attack topic and in Generative Adversarial Networks since they were first designed by Godfellow et al. and several applications have been explored.

In 2017, Papernot et al., introduced a practical way to perform a black-box attack to fool a DNN image classifier using a substitute DNN classifier to learn the black-box model boundaries and craf the adversarial examples through the Fast Gradient Sign algorithm[5].

As one of the best tools to generate adversarial examples, GANs have been also used to attack black-box classification systems such as Malware detectors or Intrusion Detection Systems. Research by Hu and Tan (2017) shown the first GAN based framework to generate adversarial malware examples named MalGAN [2]. Their approach was to use the GAN detector to learn the black-box model decisions, using the labels and the generator to craft adversarial examples adding to the original malware feature list, irrelevant features, that is, features which do not affect the validity of the malware. The results proven by the authors were further confirmed and improved by Kawai, Ota, & Dong [3].

Recently a Generative Adversarial Network-based attack against Intrusion Detection Systems was described by Lin, Shi, & Xue, named IDSGAN [4]. It is based on the Wasserstein GAN model and the results have been obtained by training the network using the popular NSL-KDD dataset. The assumption that the generated adversarial examples are still valid comes from the intuition to modify only the non-functional features of a given example. Each type of network attack flow has its own list of functional features, so the features to be modified depend on the type of network attack.

Chapter 4

Design

Three different approaches were designed to perform the black-box evasion attack, they can be distinguished by the training of the **Critic**, which can act as:

- as substitute detector that tries to learn the black-box model decision boundaries
- as a standard WGAN critic

and for the architecture of the **Generator**, which can have:

- Simple Softmax activation layer for the generation of the categorical features
- Gumbel-Softmax activation layer for the generation of the categorical features

In the following we assign a number to each version:

Version 1. WGAN with substitute detector and Softmax layer for the generation of categorical features

Version 2. WGAN with substitute detector and Gumbel-Softmax layer for the generation of categorical features

Version 3. WGAN-GP **without** substitute detector and Gumbel-Softmax layer for the generation of categorical features

In this chapter, the design of the project will be illustrated, comprehensive of the assumptions made, the overall training schemes, the description of the architectures of the proposed solutions.
Note that in the last case the model critic does not act as a substitute detector, meaning that the black-box detector is queried only before starting the training phase, whereas the first two models require to query the black-box model, for the labels of the generated adversarial examples, at each iteration.

4.1 Assumptions

In order to develop the project, the following assumptions were made.

1. The targeted botnet detector is implemented as a machine learning classifier.
2. The attacker has no knowledge about the structure and the internal parameters of the model, thus, as already discussed in the previous chapters, the type of the attack is black-box.
3. The attacker can collect a consistent number of malicious and benign network flow examples to train a GAN, in the order of thousands.
4. The attacker can query the black-box model an unlimited number of times to obtain the labels for the adversarial examples during training (in the third approach we will limit this assumption).

4.2 Models with substitute detectors

The overall training scheme of the first two solutions is inspired by either the MalGAN and IDSGAN models [2][4]. In this case, the critic acts as a substitute detector, trying to learn the decision boundaries of the black-box detector by querying the labels for the generated adversarial examples at each iteration. Figure 4.1 summarizes the main steps needed to train the Generative Adversarial Network to fool the Black-box detector and below the pseudo-code of the training algorithm is reported.

4.3 Model without substitute detector

In order to limit the ability of the attacker, another attack scheme has been designed. In the proposed scenario the attacker has the possibility to query the labels for the malicious and benign examples, **only once**, before starting the training. The generator is then trained to modify the non-functional features of the malicious examples to look as a benign ones through a WGAN-GP training scheme. Figure 4.2 shows the training scheme for this solution.

Since the proposed solutions are all based on the WGAN and WGAN-GP models, for each generator's training iteration, the critic must be trained for c extra steps in order to provide an accurate gradient for the generator. The value of d is an hyperparameter and should be tuned to train the critic to optimality at each iteration.

Algorithm 1 GAN with substitute detector - Training Pseudocode.
Requires: the training set, the number of epochs, the batch size, the noise dimension n for the generation of adversarial examples, the number of critic extra steps, the black-box classifier BB to fool and the value of the clipping constant c.
Objective: training of the generator G to optimize the evasion rate of the generated adversarial examples.

```
procedure TRAIN($X, epochs, batch\_size, n, c\_steps, c, BB$)
    ▷ $X$ is the train set of examples
    ▷ $epochs$ is the number of epochs
    ▷ $batch\_size$ is the number of examples used at every step
    ▷ $n$ is the dimension of the noise vector
    ▷ $c_steps$ is the number of extra steps to train the critic
    ▷ $c$ is the clipping constant value
    ▷ $BB$ is the black-box classifier to query for the labels
    ▷ Initialization
    Initialize the generator, $G$, and the critic, $C$, networks
    $G \leftarrow initG()$
    $C \leftarrow initC()$
    $steps \leftarrow \frac{size(X)}{batch\_size}$
    for steps do
            for c-steps do
    ▷                  Sample a batch of benign examples, $B$.
                    $B \leftarrow sample(X, batch\_size, label = 0)$
    ▷                  Sample a batch of malicious examples, $M$, and
    ▷                  generate adversarial examples with G(M,N)
                    $M \leftarrow sample(X, batch\_size, label = 1)$
                    $M \leftarrow G(M, N)$
                    $N \leftarrow random(N, -1{,}1)$
    ▷                  Query the Black-box classifier for the labels
                    $S \leftarrow concat(B, M)$
                    $Y \leftarrow BB(S)$
                    $B \leftarrow S[Y == 0]$
                    $M \leftarrow S[Y == 1]$
    ▷                  Update the weights of C, $w$, according to Eq. (4.1)
                    $w \leftarrow optimizeC(C(B), C(M))$
    ▷                  Clip w in range [-c,+c]
                    $w \leftarrow clip(c)$
            end for
            for g-steps do
    ▷                  Sample a batch of malicious examples, M,
    ▷                  and generate adversarial examples with G(M,N)
                    $M \leftarrow sample(X, batch\_size, label = 1)$
                    $M \leftarrow G(M, N)$
    ▷                  Update the parameters of G, $\theta$, according to Eq. (4.3)
                    $\theta \leftarrow optimizeG(C(M))$
            end for
    end for
end procedure
```

Algorithm 2 GAN without substitute detector - Training Pseudocode.
Requires: the training set, the number of epochs, the batch size, the noise dimension N for the generation of adversarial examples, the number of critic extra steps, the hyperparameter λ.
Objective: training of the WGAN-GP model.

```
 1: procedure TRAIN($X, epochs, batch\_size, n, c\_steps, \lambda$)
 2:     ▷ $X$ is the train set of examples
 3:     ▷ $epochs$ is the number of epochs
 4:     ▷ $batch\_size$ is the number of examples used at every step
 5:     ▷ $n$ is the dimension of the noise vector
 6:     ▷ $c_s teps$ is the number of extra steps to train the critic
 7:     ▷ $\lambda$ is the hyperparameter of the critic cost function
 8:     ▷ Initialization
 9:     Initialize the generator, $G$, and the critic, $C$, networks
10:     $G \leftarrow initG()$
11:     $C \leftarrow initC()$
12:     $steps \leftarrow \frac{size(X)}{batch\_size}$
13:     for steps do
14:             for c-steps do
15:     ▷               Sample a batch of benign examples, $B$.
16:                 $B \leftarrow sample(X, batch\_size, label = 0)$
17:     ▷               Generate adversarial examples with G(N)
18:                 $N \leftarrow random(N, -1,1)$
19:                 $M \leftarrow G(N)$
20:     ▷               Update the weights of C, $w$, according to Eq. (4.2)
21:                 $w \leftarrow optimizeC(C(B), C(M))$
22:             end for
23:             for g-steps do
24:     ▷               Generate adversarial examples with G(N)
25:                 $M \leftarrow G(M, N)$
26:     ▷               Update the parameters of G, $\theta$, according to Eq. (4.3)
27:                 $\theta \leftarrow optimizeG(C(M))$
28:             end for
29:     end for
30: end procedure
```

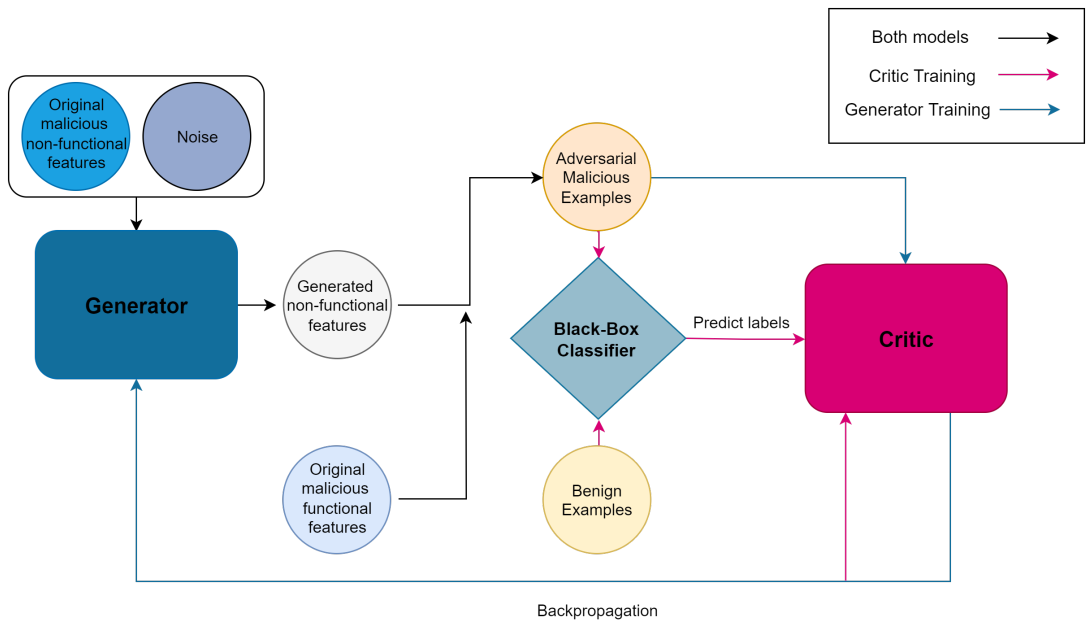

Both models
Critic Training
Generator Training
Original malicious non-functional features
Noise
Generator
Generated non-functional features
Adversarial Malicious Examples
Black-Box Classifier
Predict labels
Critic
Original malicious functional features
Benign Examples
Backpropagation

Figure 4.1: GAN with substitute detector - Training Scheme

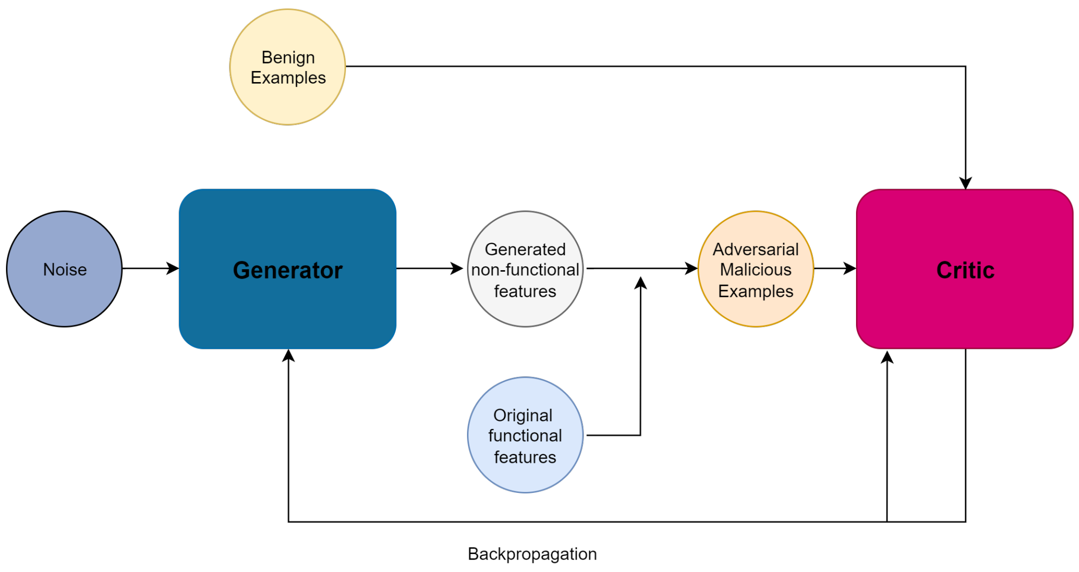

Benign Examples
Noise
Generator
Generated non-functional features
Adversarial Malicious Examples
Critic
Original functional features
Backpropagation

Figure 4.2: GAN without substitute detector - Training Scheme

4.4 Critic and Generator networks description

In the following sections more accurate descriptions of the neural network architectures of the generators and critics are provided.

4.4.1 Critic

The critic is a Deep Neural Network. The most important feature of this model is that, in order to approximate the Wasserstein distance, it's output layer must be a single neuron with a linear activation function, as the critic does not outputs probabilities but scores.

Solutions with substitute detector
To train the critic to learn the black-box classifier, in the scheme in which the critic acts as a substitute detector, both benign and adversarial malicious examples are labelled according to the black-box predictions. Then, the model's weights are updated according to the following loss function:

$$L_C = \max_{w \in W} \mathop{\mathbb{E}}_{x \in BB_{benign}} [f_w(x)] - \mathop{\mathbb{E}}_{x \in BB_{malicious}} [f_w(x)] \tag{4.1}$$

where $BB_{malicious}$ and BB_{benign} are respectively the set of samples labelled as malicious and benign by the black-box classifier and $f_w(x)$ is the score given by the critic. This objective function should let the GAN's critic learn to maximize the scores difference between the samples of the two sets. As the function learned by the critic must satisfy the K-Lipschitz continuity constraint, the values of the weights are clipped after each update in the range $[-c, +c]$. Value of c is an hyperparameter and should be carefully selected.

Solution without substitute detector
For the version without substitute detector, the proposed loss function is based on the WGAN-GP model, which models the noise samples to look as the benign non-functional features:

$$L_C = \max_{w} \mathop{\mathbb{E}}_{\tilde{x} \sim \mathbb{P}_g} [f_w(\tilde{x})] - \mathop{\mathbb{E}}_{x \sim \mathbb{P}_r} [f_w(x)] + \lambda \mathop{\mathbb{E}}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(||\nabla_{\hat{x}} f_w(\hat{x})||_2 - 1)^2] \tag{4.2}$$

Where $\mathbb{P}_g$, $\mathbb{P}_r$ and $\mathbb{P}_{\hat{x}}$ are the data distributions explained in 3.4. In this case the K-Lipschitz continuity constraint is enforced by the gradient penalty term. The value of λ is an hyperparameter.

4.4.2 Generator

The generator is a Deep Neural Network as the critic. For the versions with the substitute detector, the input to the generator are the non-functional features of the original malicious example that we aim to perturb and a noise vector of size M, whereas, for the version without substitute detector, the input is only the noise vector of size M.

As an important difference compared to related works such as MalGAN and ISDGAN, the proposed framework train the generator to be able to produce a set of both numerical and categorical outputs. To do so, the approach is the one proposed in [21]. On top of the last layer we stack a dense layer with a number of neuron equal to the number of categories we aim to represent, for each categorical variable. In the output layer, the outputs of all the dense layers are concatenated.

When training the critic, the generated inputs have to be one-hot encoded, predicted by the black-box detector, and then fed to the critic, whereas, when training the generator, the feature vectors can either be soft-encoded or one-hot encoded, in this way the training of the generator should converge.

The first solution with substitute detector uses a normal softmax activation function for the categorical dense layers, when training the critic the argmax function is applied on each soft-encoded vector to obtain a one-hot encoded vector for each categorical feature, while, when training the generator, we keep the soft-encoded vectors as they are, to be able to back-propagate the gradient through the generator layers.

The second and the third solution, instead use the Gumbel-Softmax as activation function (see eq. 3.13) for the dense categorical layers. The critic is then fed with one-hot encoded vectors as well but the gradient is computed through the soft-encoded output. In this case, the output for the categorical variables is always a one-hot encoded vector both when training the Generator and the Critic. Using a low value for the temperature τ, the soft-encoded output is very similar to the one-hot one, thus, the soft version approximates better the one-hot encoded output. The latter method should provide a better feedback when training the generator since the critic deals only with discrete values of the categorical inputs, while the former method allows the critic to receive continuous inputs for the categorical inputs.

An implementation of the Gumbel-Softmax layer is provided in A.1. For all the three versions, the numerical features are generated by a separate dense layer with hyperbolic tangent activation function.

The original non-functional features in the original malicious examples are substituted by the ones produced by the generator. The new generated malicious example are subsequently fed into the critic. The generator's weights are then updated by the following loss function:

$$L_G = \min_{\theta} - \mathop{\mathbb{E}}_{m,n \in M,N}[f_w(g_\theta(m,n))] \tag{4.3}$$

where M is the original training set of malicious examples, N is the noise vector and θ is the set of weights of the generator network.

In Figure 4.3 is shown a generic scheme of the generator network.

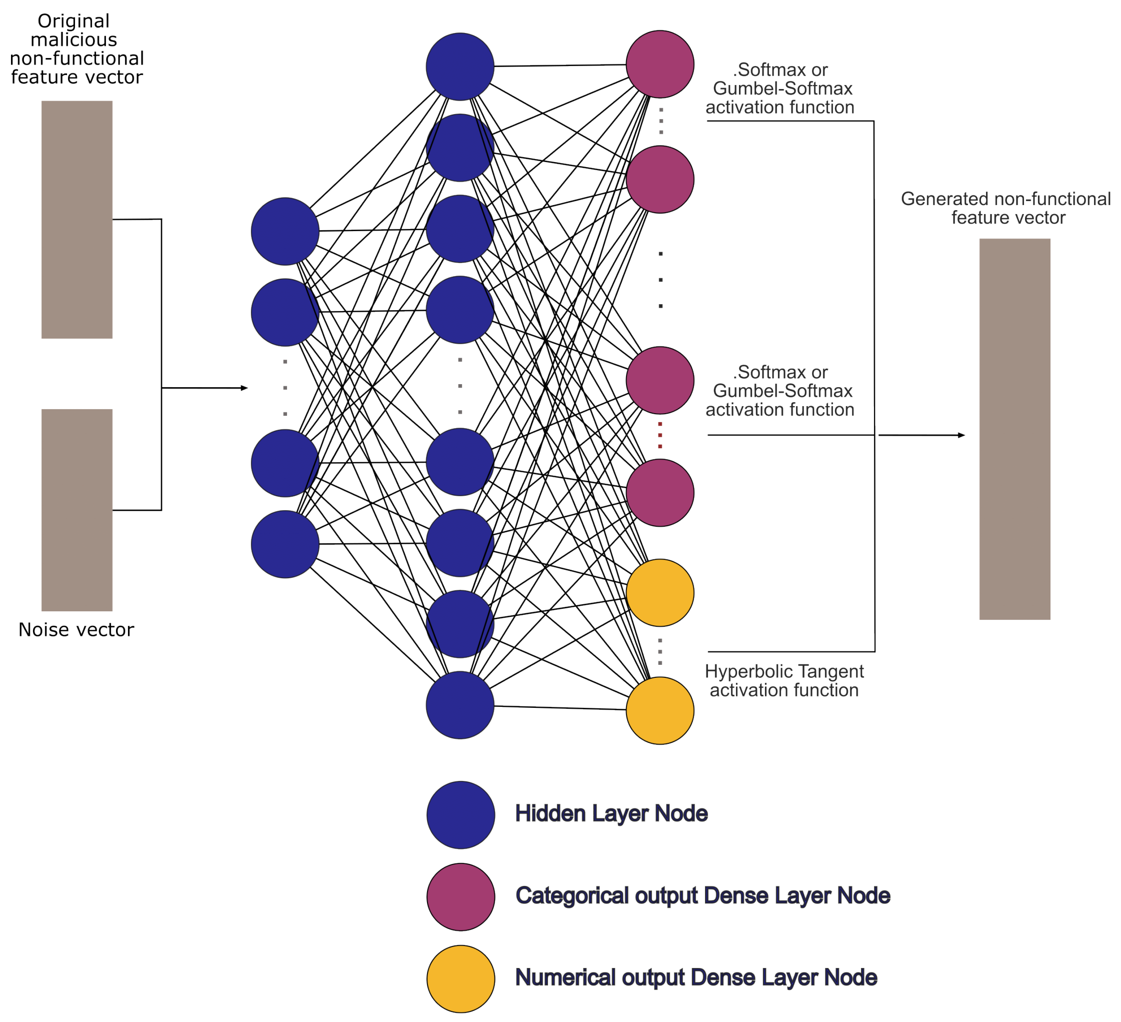

Original malicious non-functional feature vector
Noise vector
.Softmax or Gumbel-Softmax activation function
.Softmax or Gumbel-Softmax activation function
Hyperbolic Tangent activation function
Generated non-functional feature vector
Hidden Layer Node
Categorical output Dense Layer Node
Numerical output Dense Layer Node

Figure 4.3: Generator scheme: the figure shows the architecture of generators version 1 and 2. Version 3 takes as input only the noise vector.

Chapter 5

Development

In this chapter are provided a brief description of the CTU-13 dataset, the division of data into the Black-Box and GAN datasets, their respective pre-processing, and the implementation details of the GAN models, comprehensive of the values used for the hyperparameters.

5.1 Dataset

The CTU-13 dataset is a dataset which contains examples of real network botnet, normal and background traffic. The dataset is divided into 13 scenarios, all of them captured in the CTU University, Czech Republic, in 2011. Each scenario contains background and normal traffic and botnet traffic related to a specific malware and to a different number of infected computers in the network. The authors of the dataset processed the captured packets to obtain bidirectional netflows files and manually labelled the flows examples in a very detailed way. In Table 5.1 are listed the features which describe each example of the CTU-13 dataset.

5.1.1 Black-box and GAN datasets

As a primary constraint to avoid to invalidate the work, the examples used to train the black-box and GAN models must be different, and, moreover, we want the black-box classifier to be capable of recognizing most of the original malicious examples that the GAN is going to modify.

To satisfy these requirements, examples from different scenarios have been selected to build the black-box model dataset, including examples from the same scenario used to train the GAN, to ensure that the black-box classifier can correctly recognize the original malicious traffic flows that the GAN will perturb.
Among all the examples present in the dataset, those that have been chosen to

CTU-13 features	
Feature	Description
StartTime	start time of the attack, each record has a different timestamp in the format yyyy/MM/dd HH:mm:\\.ffffff
Dur	duration of the attack specified in seconds
Proto	protocol used e.g. 'tcp', 'udp', 'icmp'...
SrcAddr	source IP address
Sport	source port address from where the traffic was originated, values are both in hexadecimal and integer format
Dir	direction of the traffic represented with values '->', '?>', '<->', <?>, 'who', '<-', <?'
DstAddr	destination IP address
Dport	destination port address where the traffic was directed, values are bot in hexadecimal and integer format
State	state of the transaction, depends on the protocol used
sTos	source type of service field
dTos	destination type of service field
TotPkts	total transaction packet count
TotBytes	total transaction bytes
SrcBytes	total transaction bytes from source to destination
Label	detailed traffic label

Table 5.1: CTU-13 feature description

build the black-box and GAN dataset are only the ones for which the label starts with "From-Botnet" or with "From-Normal", in fact, as stated by the authors of the CTU-13 dataset: *"Please note that the labels of the flows generated by the malware start with "From-Botnet". The labels "To-Botnet" are flows sent to the botnet by unknown computers, so they should not be considered malicious perse. Also for the normal computers, the counts are for the labels "From-Normal". The labels "To-Normal" are flows sent to the botnet by unknown computers, so they should not be considered malicious perse."*[22].

To build the GAN dataset, examples from only one scenario were selected, corresponding to a specific botnet attack, since the non-functional features that are going to be modified depend on the specific attack we aim to perform.

In Table 5.2 the exact number of examples from each scenario, used to build the Black-box and GAN datasets, is shown. The column "valid flows" indicate the number of examples whose label starts with "From-Normal" or with "From-Botnet".

Black-box dataset pre-processing

The black-box dataset was pre-processed according to the following steps:

- Drop features 'StartTime', 'SrcAddr','DstAddr','State'
- Create a new column 'target' with value 0 if the corresponding label starts with "From-Normal", 1 if it starts with "From-Botnet", 2 otherwise
- Keep only rows with target values equal to 0 or 1

Scenario	Total flow examples	valid flows	Black-box set	GAN set
3	4,710,638	143,125	143,125	-
4	1121076	277,75	277,75	-
5	129,832	5,561	5,561	-
7	114,077	1,732	1,732	-
10	1,309,791	122,157	79,402	42,755
11	107,251	10,873	10,873	-
12	325,471	9,783	9,783	-
13	1,925,149	71,782	71,782	-

Table 5.2: Number of examples from each scenario for Black-box and GAN datasets

- Fill the null values with -1 (null values in columns 'Sport', 'Dport', 'sTos', 'dTos')
- Convert all 'Sport' and 'Dport' values to integer format
- One-hot encode 'dTos', 'sTos', 'Dir' and 'Proto' variables
- Standardize numerical feature in range [0,1] with the following formula:

$$X_{std} = \frac{X - min(X)}{max(X) - min(X)} \tag{5.1}$$

GAN dataset pre-processing

Examples from scenario 10 of the CTU-13 dataset were selected to train and test the GAN, as shown in table 5.2. This scenario mostly contains flow examples of DDoS attacks.

The following pre-processing steps were performed on the GAN dataset before training GAN:

- Drop features 'StartTime', 'SrcAddr','DstAddr','State'
- Create a new column 'target' with value 0 if the corresponding label starts with "From-Normal", 1 if it starts with "From-Botnet", 2 otherwise
- Keep only rows with target values equal to 0 or 1
- Fill the null values with -1 (null values in columns 'Sport', 'Dport', 'sTos', 'dTos')

- Convert all 'Sport' and 'Dport' values to integer format
- One-hot encode 'dTos', 'sTos', 'Dir' and 'Proto' variables
- Standardize numerical feature in range [-1,1] with the following formula:

$$X_{std} = \frac{X - min(X)}{max(X) - min(X)} * (1 + 1) - 1 \quad (5.2)$$

It is important to note that the encoding depends on the data available in the dataset, so GAN and Black-Box datasets have different encodings.

The adversarial attack that has been performed addresses **only the ICMP DDoS malign flows**. In this case the **non-functional features** taken into account are **'Sport', 'Dport', 'sTos', 'dTos'**. The first two are treated as numerical features in the range [0,65535], the other two are categorical variables which values can be '0.0' or '-1.0'.

In the CTU-13 dataset the port number of an ICMP packet does not represent the port used for the connection, obviously, since it would not make any sense, we were not able to fully understand to which feature is mapped, it could probably be the service and the related code of the ICMP protocol.

Before training the GAN, its dataset is divided into Train and Test sets as shown in table 5.3.

Benign Examples	Malign Examples (only ICMP flows)	Train Set Malign	Test Set Malign
5,511	37,133	5,511	31,622

Table 5.3: GAN dataset Train Test split

5.2 Models implementation

In the following chapters will be reported the implementation details of the proposed solutions, including the hyperparameters related to the overall model and those specific to the Generator and to the Critic. The values reported are the one that should be used to reproduce the results.

Note that the training process is **very susceptible to changes in the hyperparameters**. The values indicated in this paper have been tested and allow a smooth training process, when changing the architectures of the two competing networks or the noise vector size or the networks optimizers, the training process could become unstable or, in the worst cases, could never converge.

5.2.1 Attacks with Substitute Detector

The attacks that were performed training the critic as a substitute detector are based on the WGAN model and share the same hyperparameters values.

- noise vector size M : 6
- batch size : 64
- critic extra steps (d) : 7
- epochs : 200
- generator optimizer : RMSProp, α : 10^{-5}
- critic optimizer: RMSProp, α : 10^{-5}
- weight clipping constant c : 0.01

5.2.2 Attacks without Substitute Detector

The GAN model used to perform the black-box evasion attack without substitute detector is based on a different GAN model, that is the one with gradient penalty. Hence, the hyperparameters are different

- noise vector size M : 6
- batch size : 64
- critic extra steps (d) : 5
- epochs: 500

- generator optimizer: Adam, $\alpha : 10^{-4}$, $\beta_1 : 0.0$, $\beta_2 : 0.9$
- critic optimizer: Adam, $\alpha : 10^{-4}$, $\beta_1 : 0.0$, $\beta_2 : 0.9$
- regularization parameter $\lambda : 10.0$

5.2.3 Generator

As described in chapter 4.2 and 4.3 solutions with substitute detector take as input two vector, one containing the non-functional features of the original malicious examples and a noise vector, while the solution without substitute detector take as input only the noise vector.

For each attack version the hidden layers of the generator are three and their dimensions are [32,64,32] respectively, each of them use Leaky ReLU as activation function with slope set to 0.2.

The last hidden layer, as described in chapter 4.4.2, is linked to three dense layers, one for the generation of the numerical variables which has 2 neurons and hyperbolic tangent activation function in order to generate values in the range [-1,1] that is the same range the numerical features in the training set were pre-processed. The other two dense layers are used to generate the corresponding encoded vectors of the categorical variables 'sTos' and 'dTos'.

The summaries of the architectures are reported in table 5.4,5.5,5.6.

Layer type	Output shape	Connected to
input_features	(None, 6)	—
input_noise	(None, 6)	—
concatenate_input	(None, 12)	[input_features, input_noise]
Dense_1	(None, 32)	concatenate_input
Activation_1(LeakyReLU)	(None,32)	Dense_1
Dense_2	(None, 64)	Activation_1
Activation_2(LeakyReLU)	(None, 64)	Dense_2
Dense_3	(None, 32)	Activation_2
Dense_numerical	(None, 2)	Activation_2
Dense_cat_1	(None, 2)	Activation_2
Dense_cat_2	(None, 2)	Activation_2
Activation_3(tanh)	(None,2)	Dense_numerical
Activation_4(Softmax)	(None,2)	Dense_cat_1
Activation_5(Softmax)	(None,2)	Dense_cat_2
concatenate_output	(None, 6)	[Activation_3, Activation_4, Activation_5]

Table 5.4: Generator Model - GAN with substitute detector and softmax activation for categorical feature generation

Layer type	Output shape	Connected to
input_features	(None, 6)	—
input_noise	(None, 6)	—
concatenate_input	(None, 12)	[input_features, input_noise]
Dense_1	(None, 32)	concatenate_input
Activation_1(LeakyReLU)	(None,32)	Dense_1
Dense_2	(None, 64)	Activation_1
Activation_2(LeakyReLU)	(None, 64)	Dense_2
Dense_3	(None, 32)	Activation_2
Dense_numerical	(None, 2)	Activation_2
Dense_cat_1	(None, 2)	Activation_2
Dense_cat_2	(None, 2)	Activation_2
Activation_3(tanh)	(None,2)	Dense_numerical
Activation_4(**Gumbel-Softmax**)	(None,2)	Dense_cat_1
Activation_5(**Gumbel-Softmax**)	(None,2)	Dense_cat_2
concatenate_output	(None, 6)	[Activation_3, Activation_4, Activation_5]

Table 5.5: Generator Model - GAN with substitute detector and Gumbel-Softmax activation for categorical feature generation

Layer type	Output shape	Connected to
input_noise	(None, 6)	—
concatenate_input	(None, 12)	[input_features, input_noise]
Dense_1	(None, 32)	concatenate_input
Activation_1(LeakyReLU)	(None,32)	Dense_1
Dense_2	(None, 64)	Activation_1
Activation_2(LeakyReLU)	(None, 64)	Dense_2
Dense_3	(None, 32)	Activation_2
Dense_numerical	(None, 2)	Activation_2
Dense_cat_1	(None, 2)	Activation_2
Dense_cat_2	(None, 2)	Activation_2
Activation_3(tanh)	(None,2)	Dense_numerical
Activation_4(**Gumbel-Softmax**)	(None,2)	Dense_cat_1
Activation_5(**Gumbel-Softmax**)	(None,2)	Dense_cat_2
concatenate_output	(None, 6)	[Activation_3, Activation_4, Activation_5]

Table 5.6: Generator Model - GAN without substitute detector and Gumbel-Softmax activation for categorical feature generation

5.2.4 Critic

The critic model is a Deep Neural Network classification model that takes as input a vector of size equal to the size of the full network flow example (i.e. not only the features generated by the generator) and has two hidden layers of dimension [100,50] respectively. The output layer consists of a single neuron with linear activation function.
The critic architecture is the same for all the proposed attack versions.

Table 5.7 shows the critic model summary.

Layer type	Output shape
input_features	(None, 20)
Dense_1	(None, 100)
Activation_1(LeakyReLU)	(None,100)
Dense_2	(None, 50)
Activation_2(LeakyReLU)	(None, 50)
output	(None, 1)

Table 5.7: Critic Model

Chapter 6

Experiments

In this chapter the hardware specifics, the black-box classifiers training and evaluation, and the metrics used to evaluate the GAN based attacks will be illustrated.

6.1 Hardware

The training of the GAN and all the experiments were run on a server image in the Polytechnic of Turin's BigDataLab cluster equipped with Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz, 72 logical CPUs, 390 GB available memory.

6.2 Tools

The main tools, used to develop the project, which deserve a mention are:

- TensorFlow: a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- JupyterLab: a web-based user interface for Project Jupyter which allows to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner.

6.3 Black-box Botnet Detectors

Three machine learning models were trained to act as black-box detectors. The three models are a Support Vector Machine (SVM), a Random Forest (RF) and a Multilayer Perceptron (MLP) classifier.

The SVM classifier has a RBF kernel function and the default parameters of the sklearn library.

The RF model parameters are the sklearn default ones except for the number of estimators which is set to 300.

The MLP network has three hidden layers of [100,200,100] neurons and ReLU activaion function. The output layer is made up of one neuron with sigmoid activation function. The MLP classifier has been trained for 100 epochs with batch size set to 512.

All the classifiers were evaluated on the Black-box test set in terms of both accuracy, recall and F1 score.

$$\begin{aligned} acc &= \frac{Y_{correct}}{\#Y} \\ precision &= \frac{TP}{TP+FP} \\ recall &= \frac{TP}{TP+FN} \\ F1\ score &= 2 * \frac{precision * recall}{precision + recall} \end{aligned} \tag{6.1}$$

The accuracy metric is the base metric when evaluating machine learning classifiers and shows the ratio between the correctly classified examples and the total number of examples.

High values of recall are appreciable, since we want the classifiers to be able to recognize the most of the attacks, whereas high values of F1 score indicate that the model is correctly classifying the true class but it is not mispredicting the other class.

	SVM	RF	MLP
Accuracy	99.92%	99.98%	99.96%
Recall	99.90%	99.98%	99.96%
F1 score	99.91%	99.98%	99.96%

Table 6.1: Evaluation Metrics scores for Black-box classifiers on Black-Box test set

6.4 Attacks Evaluation Metrics

The main evaluation metric, used to assess the results obtained from the training process of the GAN, is the **Adversarial Examples Evasion Rate**. The evasion rate is the percentage of generated malicious examples that successfully passed the detection.

$$E_r = \frac{\#X_{adv\_evaded}}{\#X_{adv}} * 100 \tag{6.2}$$

Moreover, the **Transferability Analysis** was performed to evaluate the capacity of the generated adversarial examples to fool the other classifiers. Transferability captures the ability of an attack against a machine-learning model to be effective against a different, potentially unknown, model. To evaluate the transferability rate, we generate adversarial examples with a GAN specifically trained to fool a given black-box detector (e.g. SVM) and, then, we compute the evasion rate on the other black-box models (e.g. RF and MLP).

In order to establish whether the generated features are more similar to those of the benign class or are still similar to those of the malicious class, the **average Minkowski distance** between the generated non-functional features of the generated adversarial examples and the non-functional features of the examples of the normal and botnet classes has been evaluated.

The Minkowski distance metric is a generalization of the Euclidean and Manhattan distance metrics. The Minkowski distance of order p between two points X and Y is defined as:

$$\begin{gathered} X = (x_1, x_2, ..., x_n),\ Y = (y_1, y_2, ..., y_n) \\ D(X,Y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^(1/p) \end{gathered} \tag{6.3}$$

In order to compute the distance between the non-functional features of a generated adversarial example and a set of examples, S, we compute the Minkowski distance between the non-functional features of the generated example and all the examples in a set, S, and then calculate the average.

$$S \in \{S_{normal}, S_{botnet}\},\ Y \in S$$
$$avg\_D(X, S) = \frac{1}{m} \sum_{j=1}^{m} D(X, Y^{(j)}) \tag{6.4}$$

6.5 Results

In the following are shown the best results obtained in terms of the evaluation metrics explained above: evasion rate, transferability analysis and the average Minkowski distance between the non-functional features of the generated examples and both the benign and malicious sets.

6.5.1 Evasion Rate

In Table 6.2 are reported, for each black-box classifier, the original malicious detection rate (True Positive Rate, TPR) and the evasion rate achieved on the GAN test set by each GAN trained to fool the respective black-box model. Each row corresponds to a different version of the attack, the enumeration of the attack versions follows the one shown at the beginning of chapter 4.

An high evasion rate score indicates the success of the attack. GANs already demonstrated to be an effective tool to craft adversarial examples, as shown in other related works [4] [2] [3], so, the results were expected to be better or similar to the ones already obtained in the other works. In this first analysis the attack are evaluated only on the specific black-box classifier that they were trained to fool.

	RF	SVM	MLP
Original TPR on GAN test set	99.99%	100.00%	99.99%
GAN Version 1 - Evasion Rate	97.62%	100.00%	100.00%
GAN Version 2 - Evasion Rate	98.31%	99.99%	100.00%
GAN Version 3 - Evasion Rate	22.36%	96.82%	70.05%

Table 6.2: Evasion Rate scores on GAN test set

As shown in Table 6.2, both version 1 and version 2 of the attack framework achieved great results and almost completely evaded all the black-box detectors. Regarding the two GAN based frameworks with substitute detector, the second version, with the Gumbel-Softmax layer, achieved slightly better results. On the contrary, the third version of the attack performed poorly in terms of evasion rate, even if it achieved appreciable scores on the MPL and SVM models, it cannot fool the Random Forest one.

Still, the main advantage of the third attack version is that it does not require to query the black-box classifier for the labels at each iteration, hence it is faster to train and easier to implement and, still, can produce interesting results. Indeed, worse results were expected for the version 3, since it is not "guided" by the output of the black-box detector in the training phase, but the main threat of this approach is that almost every attacker with a minimum knowledge about Generative Adversarial Networks can launch the attack.

The obtained scores demonstrated that the first two versions of the attacks were successful and that the GAN model can produce good adversarial examples even modifying only a small subset of the original features, which are, in this case, two numerical features, namely 'Sport' and 'Dport', and two categorical features, namely 'sTos' and 'dTos'.

6.5.2 Transferability Analysis

In Table 6.3 are shown the scores achieved by the GANs trained to fool a specific classifier in terms of evasion rate on all the considered black-box models, all the scores have been obtained on the GAN test set.

It is important to note that the way in which the GAN is trained relies on the fact that the generator tries to fool the GAN critic, and then the generated examples are evaluated on another model. For this reason, it should not be surprising that the adversarial examples generated to fool a specific model can transfer and fool other classifiers.

The results obtained in terms of transferability of the adversarial attacks demonstrate the capacity of most of the adversarial examples to fool other, potentially unknown, models. The Random Forest model, which is usually regarded as a robust model against adversarial attacks, shows, indeed, the worst results in terms of evasion rate as it is the model which is more resistant to all the attacks.

The differences in transferability between the version 1 and 2 of the attack is not really appreciable when observing the results obtained, however it seems that, on average, the version with the Gumbel-Softmax layer can generalize a better the evasion attack when evaluated on other other models. The results obtained on

	RF	SVM	MLP
GAN Version 1			
GAN Trained to fool RF	97.62%	100.00%	99.49%
GAN Trained to fool SVM	100.00%	100.00%	100.00%
GAN Trained to fool MLP	4.55%	100.00%	100.00%
GAN Version 2			
GAN Trained to fool RF	98.31%	99.67%	99.69%
GAN Trained to fool SVM	13.52%	99.99%	99.99%
GAN Trained to fool MLP	99.85%	99.98%	100.00%
GAN Version 3			
GAN Trained to fool RF	22.36%	98.23%	67.44%
GAN Trained to fool SVM	19.49%	96.82%	65.84%
GAN Trained to fool MLP	26.64%	95.93%	70.05%

Table 6.3: Evasion Rate Scores Transferability Analysis

the first and second versions of the attack show that the generated examples can almost completely transfer to at least one other black-box classifier.

The third version of the attack obtained the worst scores in terms of transferability, however it is interesting to note that, compared to the other two versions of the attack, the results are stable, independently of the model the GAN was trained to fool, on all the black-box classifiers, while the other attacks achieved unexpected results when the GAN model was trained to fool different black-box classifiers. This is not surprising since, in the third version, the black-box classifier is queried only once before starting, but not involved during the training process. Being a completely different type of attack, compared to the other two versions, the attack without substitute detector shows different properties and advantages such as the possibility to predict with a discrete precision the evasion rate on other black-box classifiers training the model to fool a generic black-box detector.

6.5.3 Minkowski Distance of the non-functional features

In Figures 6.1, 6.2, 6.3 are shown the plots of the average Minkowski distance (with root value, $p = 4$) between the generated non-functional features of the adversarial examples and those of the original malicious and benign sets. Since we are computing the distance between one-hot encode features, namely 'sTos' and 'dTos' (in the range [0,1]), and numerical features, namely 'sport' and 'dport' (in the range [0,65535], the numerical feature have been standardized in the range [0,1] before evaluating the distances, to avoid to bias the computation. If the generated non-functional features are more similar to the benign set, the points should lie below the blue line, otherwise they should be above it.

It is important to note that if the generated non-functional features are similar to the benign set, a domain expert could still recognize the adversarial example and take the necessary countermeasures, Indeed, the main issue of adversarial examples is that, often, they can't be easily recognized by the human eye, and can escape the detection of machine learning classifiers.

The Average Minkowski distance between the generated non-functional features of the adversarial examples and the malicious and benign sets evaluated on the attack version 1 are shown in Figure 6.1. It is evident that all the generated features are all very similar to the benign set and much different from the malign one. The fact that all the points lie almost in the same portion of the plot could indicate a poor capacity in producing different outputs and could be a symptom of mode collapse, meaning that the generated features all assume almost the same values. Passing directly the softmax distribution of categorical features to the GAN critic seems to led the generator to produce the same values for each input.

In Figure 6.2 the Average Minkowski distance between the generated non-functional features of the adversarial examples and the malicious and benign sets evaluated on the attack version 2 are shown. Compared to the first attack version, the one with the Gumbel-Softmax layer seems to generate features with higher variance in the features ranges. Notably, the plot 6.2a and 6.2c show that some points are also closer to the malicious set than to the benign one, meaning that it is harder to recognize them, even for a domain expert.

Finally, in Figure 6.3 the Average Minkowski distance between the generated non-functional features of the adversarial examples and the malicious and benign sets evaluated on the attack version 3 are shown. The generated features, as expected, span across wider ranges of values. This property of the generator is certainly desired as it allows to produce different examples, but at the same time it must be remembered that, for black-box models such as the random forest, most of the examples are unable to evade the detection.

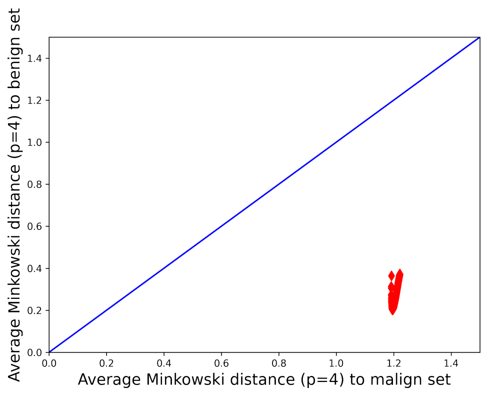
6.5.4 Evaluation of the improvement in modifying also categorical features

In order to asses the improvement that the perturbation of categorical features can bring in the success of a black-box evasion attack, we compare the evasion rate of the attack when using a generator network which handles only categorical features and the attack framework GANV2, which handles both numerical and categorical features.

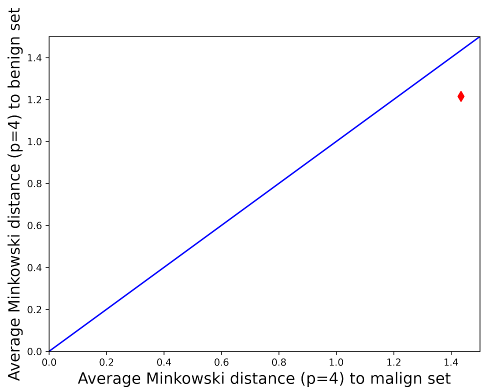
The number of modified features is higher when perturbing also categorical features, so we expect higher evasion rates. In table **??** are shown the results of such comparison.

	RF	SVM	MLP
GAN Version 2 (substitute detector and Gumbel-Softmax)	98.31%	99.99%	100.00%
GAN (substitute detector and only numerical features)	1.24%	97.36%	91.83%

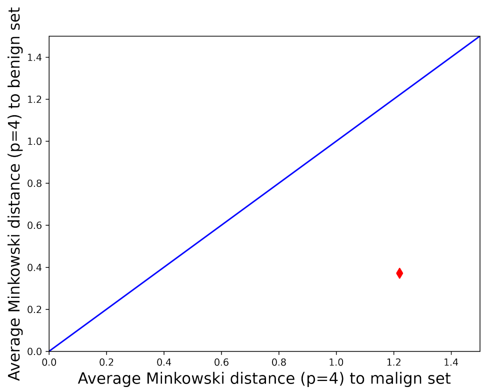
Table 6.4: Evasion rate comparison between perturbing both numerical and categorical features and only numerical features



(a) Random Forest

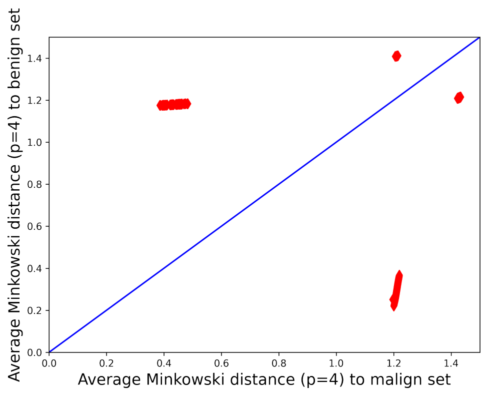


(b) Support Vector Machine

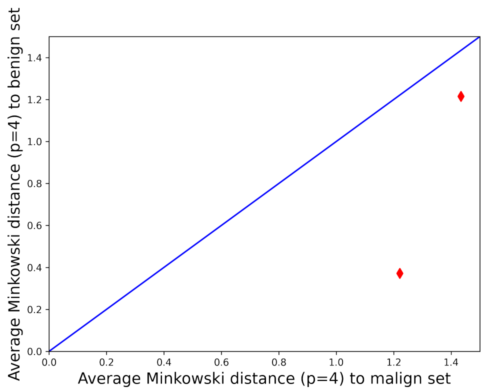


(c) Multilayer Perceptron

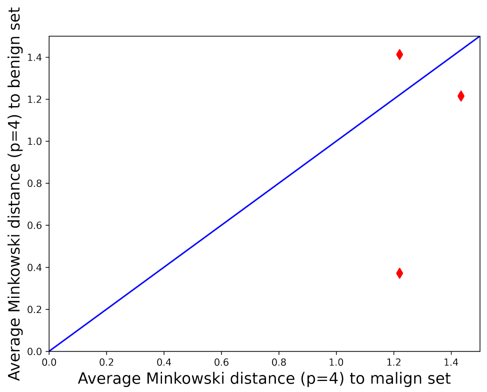
Figure 6.1: Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 1



(a) Random Forest

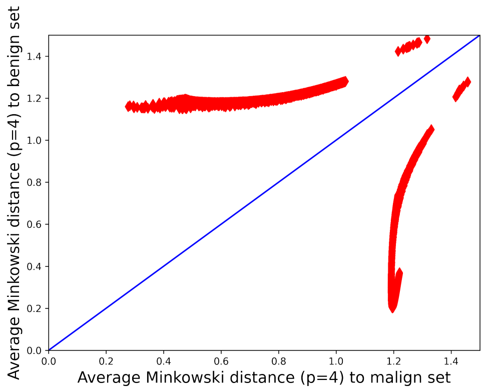


(b) Support Vector Machine

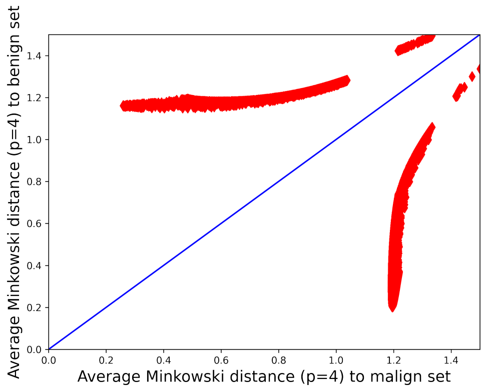


(c) Multilayer Perceptron

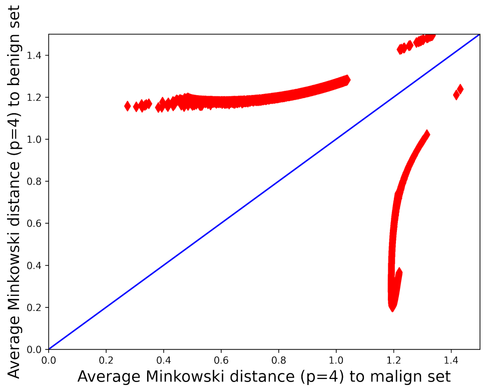
Figure 6.2: Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 2



(a) Random Forest



(b) Support Vector Machine



(c) Multilayer Perceptron

Figure 6.3: Plots of Average Minkowski distance between the generated features of the adversarial examples and the benign and malicious sets - Attack Version 3

Chapter 7

Conclusions

7.1 Success of the attacks

As the first objective of the work is to define whether or not is it possible to fool the black-box botnet detectors, we can evaluate the success of the designed attack frameworks observing the results collected in the experiments.

7.1.1 Models with substitute detector

The first and second versions of the attack frameworks were both able to fool the respective black-box models they were trained for, reason why it is definitely possible to assert that these two attack frameworks were successful and all the black-box classifiers were almost completely evaded. In spite of everything, the transferability analysis shows that in some cases the performances of the generator in terms of evasion rate can decrease drastically.

Regarding the quality of the generated features of the adversarial examples, the generators trained with these methodologies do not provide remarkable results, since they show high similarity among them but they are very different from the ones of the real malicious examples. In this way it could be easy to counter the evasion of such examples. The generator that use the Gumbel-Softmax layer for the generation of the categorical features seems to provide examples whose features are less similar to each other, spanning over a wider range of values. The ability to generate more different outputs is surely due to the stochasticity introduced by the Gumbel-Softmax output layer which allows to sample from the categorical distribution and which still provides a way to differentiate through the sampling operation.

7.1.2 Model without substitute detector

As already discussed, the third attack version is the one that provides the worst results in terms of evasion rate. However, the the last approach is the one that allows to train the generator to produce the biggest quantity of different examples. On the other side, using this approach, results on more robust black-box models can be very bad, and, in general, the attacks have an higher probability to fail. Hence, an improvement on the version three could be a modified loss function or a little relaxation on the ability of the attacker to query the black-box model, in order to find a reasonable trade-off between the evasion rate score and the difference between the generated flow examples. Moreover, this approach requires much less time to train the generator and, in some case, the attack can still be considered successfull.

7.2 Improvement of perturbing categorical features

As shown in Table **??**, by perturbing also the categorical features of the malicious examples it's possible to increase the evasion rate of the attack. In some cases it is also needed to do this so that the attack can be considered successful. This shows a correlation between the number of feature that can be perturbed and the probability that an adversarial example bypass the detection. Surely it is an issue that has to be considered in the domain of machine learning security.

7.3 Efficacy of the Straight-Trough Gumbel-Softmax Estimator

Even if, theoretically, the Gumbel-Softmax distribution and the Straight-Through Gumbel-Softmax estimator provide a better way to generate categorical features, allowing to one-hot encode the categorical feature and well approximate the categorical distribution in a differentiable way, instead of directly passing the continuous representation of the same to the critic, the results and the evidence of the superiority of the approach is not much visible. The second version of the attack framework seems to provide better property in producing different outputs but in terms of evasion rate the version one reaches better or similar scores.

As with real data, one-hot encoded vectors are sent to the discriminator when using the hard Gumbel-softmax and the Straight Through estimator, in this way, the discriminator should be better at discriminating between real (one-hot) and false (not one-hot) data (if the softmax output is passed). Still, the advantage of the method is that it is surely straightforward and easy to implement, but the results, in terms of evasion rate, are similar to the ones in which the Gumbel-Softmax distribution is not applied.

7.4 Reproducibility of the adversarial examples

Even if we considered two of the attacks as successful, major issues can hinder the development of the attack. For instance, in this work we assumed that there were four non-functional features in a ICMP DDoS malicious traffic flow. One of the obstacles in reproducing such adversarial malicious traffic flows could be due mainly from the way the traffic flows are collected and summarized. Moreover, it is not trivial to respect the time constraints and to know how the features are collected and extracted in the target network before hand.

Chapter 8

Future Works

We have demonstrated the feasibility of a black-box attack using a different dataset with respect to the previous works on the same topic (IDSGAN) by adopting a slightly different architecture and approach.

The great weakness of the work is the assumption that the adversarial malicious examples are still valid after changing the non-functional features. The replication of such an attack should be demonstrated in a real world environment and involves several other issues such as the collection of the malicious data to train the GAN model.

In this specific context we assumed that there exist a set of non-functional features for each botnet attack. Before starting to develop strategies and techniques to secure machine learning algorithms against this type of black-box evasion attacks, a further research on the feasibility and complexity of the attack should be performed in this direction. If the assumption on the validity of the adversarial attacks is denied, the feasibility of the attack also falls.

However, a possible way to make the classifiers more robust against black-box evasion attack could be to train several classifiers with different algorithms and choose randomly the classifier that should provide the prediction for each input. In this way the attack based on the GAN should have less percentages of success, since the decision boundaries of the black-box model would change at each input.

At the best of our knowledge there isn't still an effective and standard method to avoid such attacks. Depending on the domain of application some solutions could be applied to mitigate the attack.

Appendix A

Tensorflow main components implementation

A.1 Gumbel-Softmax Layer

Listing A.1: GumbelSoftmaxLayer implementation

```
from typing import Optional
import tensorflow as tf
from tensorflow import Tensor, TensorShape, one_hot, squeeze,
    stop_gradient
from tensorflow.keras.layers import Layer
from tensorflow.keras.utils import register_keras_serializable
from tensorflow.math import log
from tensorflow.nn import softmax
from tensorflow.random import categorical, uniform

TOL = 1e-20
def gumbel_noise(shape: TensorShape) -> Tensor:
    """Create a single sample from the standard (loc = 0, scale = 1)
    Gumbel distribution."""
    # G_i can be approximated with -log(-log(Uniform(0,1)))
    uniform_sample = uniform(shape, seed=0, dtype = tf.dtypes.float64
    )
    return -log(-log(uniform_sample + TOL) + TOL)

@register_keras_serializable(name='GumbelSoftmaxLayer')
class GumbelSoftmaxLayer(Layer):
    "A Gumbel-Softmax layer implementation that should be stacked on
    top of a categorical feature logits."

    def __init__(self, tau: float = 0.2, name: Optional[str] = None,
    **kwargs):
        super().__init__(name=name, **kwargs)
        self.tau = tau

    def call(self, _input):
        """Computes Gumbel−Softmax for the logits output of a
    particular categorical feature."""
        noised_input = _input + gumbel_noise(tf.shape(_input))
        soft_sample = softmax(noised_input/self.tau, −1)
        hard_sample = tf.argmax(soft_sample, tf.dtypes.float64)
        hard_sample = squeeze(one_hot(categorical(log(soft_sample),
    1),         y = tf.stop_gradient(hard_sample) − soft_sample +
    soft_sample
        return y

    def get_config(self):
        config = super().get_config().copy()
        config.update({'tau': self.tau})
        return config
```

A.2 Generator

Listing A.2: Generator build function

```python
def get_generator_model(gumbel = False, substitute_detector = True):
    g_hidden_layers = [32,64,32]
    # Input layer
    noise = layers.Input(shape = (NOISE_DIM,),name = "input_noise")
    x = None
    if substitute_detector:
        example = layers.Input(shape = (INPUT_DIM_GEN,), name = "input_example")
        x = Concatenate(axis = 1)([example,noise])
    else:
        x = noise
    # Hidden layers
    for n,i in zip(g_hidden_layers,range(len(g_hidden_layers))):
        x = layers.Dense(n, use_bias = True, name = f"g_hidden_layer{i+1}")(x)
        x = layers.LeakyReLU(alpha = 0.2)(x)
    numerical_gen = layers.Dense(NUM_VAR, use_bias = True, activation = "tanh", name = 'gen_numerical_variables_output')(x)

    dtos_gen = None
    stos_gen = None
    if gumbel == True:
        dtos_dense = layers.Dense(dtos_nvalues, use_bias = True, name = "dtos_logits")(x)
        stos_dense = layers.Dense(stos_nvalues, use_bias = True, name = "stos_logits")(x)
        dtos_gen = GumbelSoftmaxLayer(name = "dtos_output")(dtos_dense)
        stos_gen = GumbelSoftmaxLayer(name = "stos_output")(stos_dense)
    else:
        dtos_gen = layers.Dense(dtos_nvalues, use_bias = True, activation = "softmax", name = "dtos_output")(x)
        stos_gen = layers.Dense(stos_nvalues, use_bias = True, activation = "softmax", name = "stos_output")(x)
    out = Concatenate(axis = 1)([numerical_gen,dtos_gen,stos_gen])
    g_model = None
    if substitute_detector:
        g_model = keras.models.Model([example,noise], out, name="generator")
    else:
        g_model = keras.models.Model(noise, out, name="generator")
    return g_model
```

A.3 GAN model class with substitute detector

Listing A.3: GAN model class with substitute detector

```python
class GAN(keras.Model):
    def __init__(
        self,
        noise_dim,
        bb_model_type = "MLP",
        bb_model_file = "",
        critic_extra_steps=3,
        gumbel = False,
        substitute = False
    ):
        super(WGAN, self).__init__()

        self.bb_model_type = bb_model_type
        self.bb_model_file = bb_model_file
        self.noise_dim = noise_dim
        self.d_steps = critic_extra_steps
        self.critic = get_critic_model()
        self.generator = get_generator_model(self.gumbel,self.substitute)
        self.black_box_detector = get_blackbox_model(self.bb_model_type, self.bb_model_file)

    def compile(self, d_optimizer, g_optimizer, d_loss_fn, g_loss_fn):
        super(WGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.d_loss_fn = d_loss_fn
        self.g_loss_fn = g_loss_fn

    def train(self, data, epochs = 500, batch_size = 32, gan_weights_dir = "", debug = False):

        # split dataset into ben and mal
        xmal = data[data['target'] == 1].drop('target',axis = 1)
        xben = data[data['target'] == 0].drop('target',axis = 1)
        xmal.reset_index(drop = True, inplace = True)
        xben.reset_index(drop = True, inplace = True)

        print ("------------------------------------------------------------")
        print("Attacking ",bb_model_type," black box model")
        print("Model weights will be stored at ", gan_weights_dir)
        print ("------------------------------------------------------------")

        # Predict labels for benign examples with the blackbox detector
        to_predict = utils.prepare_for_black_box(xben, gan_scaler, gan_encoders, gan_ohe_columns, gan_ohe_indexes , num_col, saved_dict , bb_scaler , bb_columns, bb_encoders, scenario , debug = False)
        yben_blackbox = self.black_box_detector.predict(to_predict).round().reshape(xben.shape[0],)

        d_losses = []
        g_losses = []
        steps = round(xmal.shape[0] / batch_size)

        for epoch in range(epochs):
            d_avg_loss = 0.0
            g_avg_loss = 0.0
            print("Epoch ",epoch+1)
            for step in range(steps):


                for i in range(self.d_steps):

                    # Batch of benign examples
                    xben_batch = xben.sample(BATCH_SIZE)
                    yben_batch = yben_blackbox[xben_batch.index]

                    # Batch of malign examples
                    xmal_batch = xmal.sample(BATCH_SIZE)

                    noise = np.random.normal(size=(xmal_batch.shape[0], self.noise_dim))
                    gen_samples = self.generator([xmal_batch.iloc[:,[1,2,6,7,8,9]],noise], training=True)
                    xgen_batch = utils.gen_samples_and_keep_unmodified(xmal_batch,gen_samples)
                    xgen_batch = pd.DataFrame(xgen_batch.numpy(), columns = columns)

                    # Predict labels of adversarial examples with the blackbox detector
                    to_predict = utils.prepare_for_black_box(xgen_batch, gan_scaler, gan_encoders, gan_ohe_columns, gan_ohe_indexes, num_col, saved_dict , bb_scaler , bb_columns, bb_encoders, scenario , debug = False)
                    ygen_batch = self.black_box_detector.predict(to_predict).round().reshape(to_predict.shape[0],)

                    # Group examples predicted as malign and benign
                    mal = tf.convert_to_tensor(pd.concat([xgen_batch[ygen_batch == 1], xben_batch[yben_batch == 1]]))
                    ben = tf.convert_to_tensor(pd.concat([xgen_batch[ygen_batch == 0], xben_batch[yben_batch == 0]]))

                    with tf.GradientTape() as tape:
                        D_mal = self.critic(mal, training=True)
                        D_ben = self.critic(ben, training=True)

                        # Calculate the critic loss using the critic's scores
                        d_loss = self.d_loss_fn(mal_scores = D_mal, ben_scores = D_ben)
                        d_avg_loss += d_loss

                    # Get the gradients w.r.t the critic loss
                    d_gradient = tape.gradient(d_loss, self.critic.trainable_variables)

                    # Update the weights of the critic using the critic optimizer
                    self.d_optimizer.apply_gradients(zip(d_gradient, self.critic.trainable_variables))

                    # Clip the weights of the critic
                    for w in self.critic.trainable_variables:
                        w.assign(tf.clip_by_value(w, -clip_const, clip_const))

                # Batch of malign examples
                xmal_batch = xmal.sample(BATCH_SIZE)


                with tf.GradientTape() as tape:
                    noise = np.random.normal(size=(xmal_batch.shape[0], self.noise_dim))
                    gen_samples = self.generator([xmal_batch.iloc[:,[1,2,6,7,8,9]],noise], training=True)
                    xgen_batch = utils.gen_samples_and_keep_unmodified(xmal_batch,gen_samples)

                    # Get the critic scores for generated examples
                    D_gen = self.critic(xgen_batch, training=True)

                    # Calculate the generator loss
                    g_loss = self.g_loss_fn(D_gen)
                    g_avg_loss+= g_loss

                # Get the gradients w.r.t the generator loss
                gen_gradient = tape.gradient(g_loss, self.generator.trainable_variables)

                # Update the weights of the generator using the generator optimizer
                self.g_optimizer.apply_gradients(zip(gen_gradient, self.generator.trainable_variables))

            # Early stop condition
            if (epoch > 0) and (abs(g_avg_loss - g_losses[-1]) < 10**(-5)):
                stop = True

            d_losses.append(d_avg_loss/(steps*self.d_steps))
            g_losses.append(g_avg_loss/steps)

            print(f"Critic Loss: {d_losses[-1]}")
            print(f"Generator Loss: {g_losses[-1]}")

            self.generator.save_weights(gan_weights_dir+"/"+bb_model_type+".ckpt")
            if stop == True:
                return (g_losses,d_losses,epoch+1)

        return (g_losses,d_losses,epochs)
```

Bibliography

[1] Mazaher Kianpour and Shao-Fang Wen. «Timing attacks on machine learning: State of the art». In: *Proceedings of SAI Intelligent Systems Conference*. Springer. 2019, pp. 111–125 (cit. on p. 2).

[2] Weiwei Hu and Ying Tan. «Generating adversarial malware examples for black-box attacks based on GAN». In: *arXiv preprint arXiv:1702.05983* (2017) (cit. on pp. 2, 15, 17, 35).

[3] Masataka Kawai, Kaoru Ota, and Mianxing Dong. «Improved malgan: Avoiding malware detector by leaning cleanware features». In: *2019 international conference on artificial intelligence in information and communication (ICAIIC)*. IEEE. 2019, pp. 040–045 (cit. on pp. 2, 15, 35).

[4] Zilong Lin, Yong Shi, and Zhi Xue. «Idsgan: Generative adversarial networks for attack generation against intrusion detection». In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2022, pp. 79–91 (cit. on pp. 2, 15, 17, 35).

[5] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. «Practical black-box attacks against machine learning». In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519 (cit. on pp. 2, 15).

[6] Muhammet Fatih Akbaş, Cengiz Güngör, and Enis Karaarslan. «Usage of machine learning algorithms for flow based anomaly detection system in software defined networks». In: *International Conference on Intelligent and Fuzzy Systems*. Springer. 2020, pp. 1156–1163 (cit. on p. 7).

[7] Fariba Haddadi, Duc Le Cong, Laura Porter, and A Nur Zincir-Heywood. «On the effectiveness of different botnet detection approaches». In: *International conference on information security practice and experience*. Springer. 2015, pp. 121–135 (cit. on p. 7).

[8] Abdurrahman Pektaş and Tankut Acarman. «Deep learning to detect botnet via network flow summaries». In: *Neural Computing and Applications* 31.11 (2019), pp. 8021–8033 (cit. on p. 7).

[9] Matija Stevanovic and Jens Myrup Pedersen. «An efficient flow-based botnet detection using supervised machine learning». In: *2014 International Conference on Computing, Networking and Communications (ICNC)*. 2014, pp. 797–801. DOI: `10.1109/ICCNC.2014.6785439` (cit. on p. 7).

[10] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. «Adversarial classification». In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 99–108 (cit. on p. 8).

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. «Generative adversarial nets». In: *Advances in neural information processing systems* 27 (2014) (cit. on p. 9).

[12] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. «Which training methods for GANs do actually converge?» In: *International conference on machine learning*. PMLR. 2018, pp. 3481–3490 (cit. on p. 9).

[13] Martin Arjovsky and Léon Bottou. «Towards principled methods for training generative adversarial networks». In: *arXiv preprint arXiv:1701.04862* (2017) (cit. on p. 9).

[14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. «Wasserstein generative adversarial networks». In: *International conference on machine learning*. PMLR. 2017, pp. 214–223 (cit. on p. 10).

[15] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. «Improved training of wasserstein gans». In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 11).

[16] J. Hui. «WGAN — Wasserstein GAN & WGAN-GP». In: *[online] Medium. Available at: <https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490> [Accessed 24 June 2022]* (2022) (cit. on p. 11).

[17] E. Benjaminson. «The Gumbel-Softmax Distribution. [online]». In: *Sassafras13.github.io. Available at: <https://sassafras13.github.io/GumbelSoftmax/> [Accessed 27 June 2022].* (2020) (cit. on p. 13).

[18] Eric Jang, Shixiang Gu, and Ben Poole. «Categorical reparameterization with gumbel-softmax». In: *arXiv preprint arXiv:1611.01144* (2016) (cit. on pp. 13, 14).

[19] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. «The concrete distribution: A continuous relaxation of discrete random variables». In: *arXiv preprint arXiv:1611.00712* (2016) (cit. on p. 13).

[20] W. Wong. «What is Gumbel-Softmax? A differentiable approximation to sampling discrete data». In: *[online] Medium. Available at: <https://medium.com/p/7f6d9cdcb90 [Accessed 20 June 2022]* (2020) (cit. on p. 14).

[21] Ramiro Camino, Christian Hammerschmidt, and Radu State. «Generating multi-categorical samples with generative adversarial networks». In: *arXiv preprint arXiv:1807.01202* (2018) (cit. on pp. 15, 22).

[22] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. «An empirical comparison of botnet detection methods». In: *computers & security* 45 (2014), pp. 100–123 (cit. on p. 25).