

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Adversarial Machine Learning on Denial of Service Recognition

Nuno Jorge Dias Carneiro Martins

MASTER'S DISSERTATION



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: José Magalhães Cruz

Co-Supervisor: Pedro Henriques Abreu

July 25, 2019

Adversarial Machine Learning on Denial of Service Recognition

Nuno Jorge Dias Carneiro Martins

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2019

Abstract

Online services have seen a rise on popularity over the last two decades. Social networks, banking applications, blogs and many other services are increasingly being embedded in the daily lives of every citizen, but many of them are targeted by intruders with malicious intent. Attackers often disrupt these services, or use them to obtain personal information, send spam or spread malware, using a high variety of techniques.

Cybersecurity is the area that studies protection against cyber-attacks, offering many defensive measures. With the growing pace at which new attacks are created, conventional detection methodologies based on anomaly detection or attack knowledge are often not enough, and machine learning is adopted.

The usage of machine learning has grown in recent years, mainly due to advances in computing power, and has successfully been applied to a high variety of areas, such as intrusion detection. Although the machine learning classifiers can successfully identify attacks in many cases, they are often themselves a target, by attackers that create specific samples that aim to deceive the classifiers, which are called "adversarial examples".

Adversarial machine learning is an area of study that examines both the generation and detection of adversarial examples, and has been extensively researched specifically in the area of image recognition, where small modifications are performed on images that cause a classifier to be deceived.

The goal of this dissertation was to perform an analysis on the behaviour of adversarial machine learning techniques on Denial of Service attacks, where an attacker aims to disrupt an online service to its intended users by flooding it with seemingly valid requests. To achieve this goal, we first studied different adversarial attack and defense strategies which were initially applied to image recognition, and explored their applications to Denial of Service attack contexts, by analyzing multiple works on adversarial malware and intrusion detection, which are the most similar application scenarios in literature. To the best of the authors knowledge, no adversarial defense strategies were ever explored for these scenarios.

We then proposed multiple experiments on two Denial of Service datasets, where we initially trained multiple state of the art classifiers on the original datasets, and then inserted perturbations on the datasets using four adversarial attack strategies. The results showed that there was a significant deterioration on the performance of all classifiers on both datasets, with the Denoising Autoencoder being the most resilient classifier for attacks with larger perturbations. We then tested four different adversarial defense strategies, and all strategies showed an improvement in resisting adversarial examples compared to the baseline classifiers against adversarial attacks, with defense strategies that revolve around including adversarial examples in the training set being the most effective ones.

Resumo

Serviços online têm visto um aumento na popularidade nas últimas duas décadas. Redes sociais, aplicações bancárias, blogs e muitos outros serviços estão cada vez mais presentes no cotidiano de cada cidadão, mas muitos deles são alvos de intrusos com intenções maliciosas. Os atacantes geralmente interrompem esses serviços ou utilizam-nos para obter informações pessoais, enviar spam ou espalhar malware, usando uma grande variedade de técnicas.

Cibersegurança é uma área que estuda proteção contra ciberataques, oferecendo diversas medidas defensivas. Com o crescente ritmo com que novos ataques são criados, métodos convencionais de detecção baseadas em detecção de anomalias ou conhecimento prévio dos ataques são muitas vezes insuficientes, sendo machine learning usado.

A utilização de machine learning tem aumentado nos últimos anos, principalmente devido aos avanços no poder computacional de dispositivos, e tem sido aplicado com sucesso numa grande variedade de áreas, como detecção de intrusões. Embora os classificadores machine learning possam identificar com sucesso os ataques em muitos casos, eles também podem ser um alvo por invasores que criam amostras específicas cujo objetivo é enganar os classificadores, que são chamados de amostras adversárias.

Adversarial machine learning é uma área de estudo que examina tanto a geração como a detecção de amostras adversárias, e tem sido vastamente pesquisada na área de reconhecimento de imagem, onde pequenas alterações são realizadas em imagens que fazem com que um classificador realize previsões incorretas.

O objetivo principal desta dissertação foi realizar uma análise do comportamento de diferentes técnicas de adversarial machine learning em ataques de negação de serviço, onde um atacante tenta afetar a disponibilidade de um serviço online, ao inundá-lo com pedidos aparentemente válidos. Para alcançar este objetivo, começamos por estudar diferentes estratégias de ataque e defesa com adversários que foram inicialmente aplicadas a reconhecimento de imagem, e exploramos as suas aplicações a ataques de negação de serviço ao analisar múltiplos trabalhos sobre detecção de intrusões e malware com adversários, já que é o cenário de aplicação mais semelhante na literatura existente. Pelo conhecimento dos autores, a aplicação de defesas adversárias nunca foi explorada nestes contextos.

Assim, nós propusemos múltiplas experiências a realizar em dois datasets de negação de serviço, onde inicialmente treinamos múltiplos classificadores estado da arte nos datasets originais, e depois inserimos perturbações nos datasets usando quatro estratégias de ataque adversário. Os resultados mostraram que houve uma perda significativa no desempenho de todos os classificadores em ambos os datasets, com o Denoising Autoencoder sendo o classificador mais resistente contra ataques com maiores perturbações. De seguida testamos quatro estratégias de defesa contra adversários, e todas mostraram uma melhoria no desempenho relativamente aos classificadores base usados contra os ataques adversários, com estratégias que envolvem a inclusão de amostras adversárias no conjunto de treino sendo as mais eficazes.

Acknowledgements

Firstly, I would like to thank my supervisors, José Magalhães Cruz and Pedro Henriques Abreu, for their continuous guidance and availability throughout this dissertation, and for always proposing new ideas to explore. I would also like to thank Professor Tiago Cruz from the University of Coimbra for providing constant feedback on the work developed.

Secondly, I am also grateful to my family, especially my parents, for providing constant support for all these years.

Nuno Jorge Dias Carneiro Martins

*“Computers are like humans -
they do everything except think.”*

John von Neumann

Contents

1	Introduction	1
1.1	Context	1
1.2	Goals	3
1.3	Contributions	3
1.4	Document Structure	4
2	Background knowledge	5
2.1	Machine Learning Fundamentals	5
2.1.1	Machine Learning Tribes	5
2.1.2	Neural Network Architectures	7
2.1.3	Transfer Learning	9
2.2	Performance Metrics	10
2.2.1	Classification Metrics	10
2.2.2	Performances Comparison	12
2.3	Adversarial Machine Learning	12
2.3.1	Fundamentals of Adversarial Machine Learning	12
2.3.2	Adversarial Attacks	14
2.3.3	Adversarial Defenses	19
2.4	Conclusions	24
3	Literature Review	27
4	Proposed Arquitecture	35
4.1	Datasets Description	35
4.2	Datasets Preprocessing	36
4.2.1	Samples Selection	36
4.2.2	One-Hot Enconding	37
4.2.3	Min-Max Normalization	37
4.2.4	Feature Selection	37
4.3	Machine Learning Algorithms	37
4.4	Adversarial Example Generation	38
4.5	Adversarial Defenses	39
4.6	Pipeline	40
4.7	Technologies	40
5	Experimental Results	41
5.1	Original Data Results	41
5.2	Adversarial Attacks	42

CONTENTS

5.2.1	Approaches Comparison	42
5.2.2	Feature Perturbation Analysis	44
5.3	Adversarial Defenses	46
5.3.1	Adversarial Training	46
5.3.2	Separate Binary Classifier	48
5.3.3	Defensive Distillation	49
5.3.4	MagNet	50
5.3.5	Approaches Comparison	51
6	Conclusions and Future Work	55
	References	57
A	Datasets feature list	63
B	Feature selection	67
C	Optimal parameters	69

List of Figures

1.1	Simple scheme of a Distributed Denial of Service attack. The attacker controls multiple zombies which flood the target server making its services unavailable to regular users.	2
2.1	Scheme of a basic autoencoder.	7
2.2	Scheme of a denoising autoencoder.	8
2.3	The general flow of a GAN. Only one loss function is used, which is be used to train both the generator and the discriminator.	9
2.4	Distillation: Softmax temperature illustrated.	10
2.5	FGSM - By using a ϵ of 0.007, which only allows at most 1 bit to be changed in a 8 bit representation of each channel of a pixel, the perturbation can lead to a misclassification.	16
2.6	JSMA on MNIST dataset. The original samples are found in the diagonal, with all other elements being adversarial examples.	17
2.7	Deepfool - a sample x_0 is added a perturbation r in the direction of the hyperplane that separates the original class and the target class.	17
2.8	Structure of an AdvGAN. The loss function used to train the generator is the combination of the loss of the discriminator and the loss of the target classifier. . . .	18
2.9	Overview of defensive distillation technique. A neural network F was initially trained and performed predictions under a certain temperature T . The predicted labels distribution is then used to train a second neural network with the same , which is theorized to be more resilient to adversaries.	21
2.10	Framework of feature squeezing. When the difference of predictions on different models using different/no squeezers exceeds a threshold, the input is considered adversarial.	22
2.11	Transferability defense: instead of assigning a larger credebility to the target class during training, the network is trained to assign it to the NULL label, proportionally to the magnitude of the perturbations.	22
2.12	MagNet defense strategy. Using an ensemble of detectors, if any considers a sample to be adversarial, it is considered to be adversarial, removing samples with high magnitude perturbations. Regular samples and adversarial samples with small perturbations are then directed to the reformer, which denoises the data and directs it to a classifier that performs predictions.	23
4.1	Pipeline of the experiments.	35
B.1	Feature selection process using Recursive Feature Elimination on both datasets. .	67

LIST OF FIGURES

List of Tables

2.1	Confusion Matrix	10
2.2	Datasets used in literature review.	15
3.1	Summary of works on adversarial machine learning applied to intrusion and malware scenarios.	32
4.1	Optimized parameters for each algorithm.	38
4.2	Tweaked parameters and distance metrics for each attack.	38
5.1	Baseline results on the NSL-KDD dataset (ROC-AUC: left, F1-score: right). . . .	41
5.2	Baseline results on the CICIDS 2017 dataset (ROC-AUC: left, F1-score: right). .	42
5.3	Parameters used by the attacks (10 attacks in total).	42
5.4	Adversarial attacks on the NSL-KDD dataset (AUC).	43
5.5	Adversarial attacks on the NSL-KDD dataset (F1-score).	43
5.6	Adversarial attacks on the CICIDS 2017 dataset (AUC).	43
5.7	Adversarial attacks on the CICIDS 2017 dataset (F1-score).	43
5.8	Analysis on modified features by JSMA attack with $\vartheta=0.3$	45
5.9	Analysis on modified features by Deepfool attack with Overshoot=1.	45
5.10	Analysis on modified features by C&W attack.	45
5.11	Adversarial training on the NSL-KDD dataset (AUC).	46
5.12	Adversarial training on the NSL-KDD dataset (F1-score).	47
5.13	Adversarial training on the CICIDS 2017 dataset (AUC).	47
5.14	Adversarial training on the CICIDS 2017 dataset (F1-score).	47
5.15	Separate Binary Classifier on the NSL-KDD dataset (AUC).	48
5.16	Separate Binary Classifier on the NSL-KDD dataset (F1).	48
5.17	Separate Binary Classifier on the CICIDS 2017 dataset (AUC).	48
5.18	Separate Binary Classifier on the CICIDS 2017 dataset (F1).	49
5.19	Distillation on the NSL-KDD dataset (AUC).	49
5.20	Distillation on the NSL-KDD dataset (F1).	49
5.21	Distillation on the CICIDS 2017 dataset (AUC).	50
5.22	Distillation on the CICIDS 2017 dataset (F1).	50
5.23	MagNet on the NSL-KDD dataset (AUC).	50
5.24	MagNet on the NSL-KDD dataset (F1).	51
5.25	MagNet on the CICIDS 2017 dataset (AUC).	51
5.26	MagNet on the CICIDS 2017 dataset (F1).	51
5.27	Defense strategies comparison on NSL-KDD dataset (AUC).	52
5.28	Defense strategies comparison on NSL-KDD dataset (F1).	52
5.29	Defense strategies comparison on CICIDS dataset (AUC).	52
5.30	Defense strategies comparison on CICIDS dataset (F1).	52

LIST OF TABLES

A.1	Complete list of features on the NSL-KDD dataset.	64
A.2	List of features on the CIC-IDS 2017 dataset. Some features were suppressed due to being statistical information about a certain property (identified with (s)) . . .	65
B.1	List of features after the selection step (NSL-KDD: left, CICIDS 2017: right) . .	68
C.1	Optimal parameters for classifiers on original NSL-KDD dataset.	69
C.2	Optimal parameters for classifiers on original CICIDS 2017 dataset.	69

Abbreviations

AUC	Area Under Curve
AE	Autoencoder
CNN	Convolutional Neural Network
C&W	Carlini & Wagner
DAE	Denoising Autoencoder
DoS	Denial of Service
DDoS	Distributed Denial of Service
Dfool	Deepfool
DNN	Deep Neural Network
DT	Decision Tree
ENS	Ensembles
FGSM	Fast Gradient Sign Method
FNR	False Negative Rate
FPR	False Positive Rate
GAN	Generative Adversarial Network
IDS	Intrusion Detection System
IoT	Internet of things
IP	Internet Protocol address
JSMA	Jacobian based Saliency Map Attack
KNN	K-Nearest-Neighbours
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
LR	Linear Regression
MITM	Man-in-the-Middle
MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology database
NB	Naive-Bayes
RBF	Radial Basis Function
RF	Random Forest
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristics
SVM	Support Vector Machine
TPR	True Positive Rate

Chapter 1

Introduction

Cyber-security is the practice of protecting computing systems and networks from digital attacks, and a rising concern in the Information Age[71].

Although most systems today are built with increasing security, there is still a vast amount of vulnerabilities, mainly due to outdated software, non-secure protocols and systems and human error. Cyber-attacks can target any infrastructure, from cloud systems to IoT devices, in the most various forms[71].

In 2016, the Internet service *Yahoo!* announced that in 2014, it was victim of one of the biggest data breaches in history [9], with 3 billion accounts' personal information being compromised. In 2018, the American multinational hospitality company *Marriot International* reported a breach of 500 million customers' personal information [8]. All of this indicates that information security is far from perfect.

1.1 Context

There is a wide variety of cyber-attacks that take very different approaches.

Denial of Service (DoS) attacks consist of an attempt to prevent regular users from using a service, by exhausting it either with a large amount of regular requests, or maliciously built ones. The attacker typically controls a large network of nodes which it uses to flood the target with data (figure 1.1). Since most of these 'zombie' nodes are making seemingly valid requests, the target can't distinguish regular users and attackers, leading to a forced shutdown, or even a crash of the systems[61]. These are one of the most common types of cyber attacks, being easy to implement and highly destructive [44].

To perform the detection of cyber-attacks, **Intrusion Detection Systems (IDS)** are used. Contrary to regular firewalls, which normally only analyze superficial information of connections, such as IPs, ports and packet headers, IDSs perform a deeper analysis on the packets information [58].

Introduction

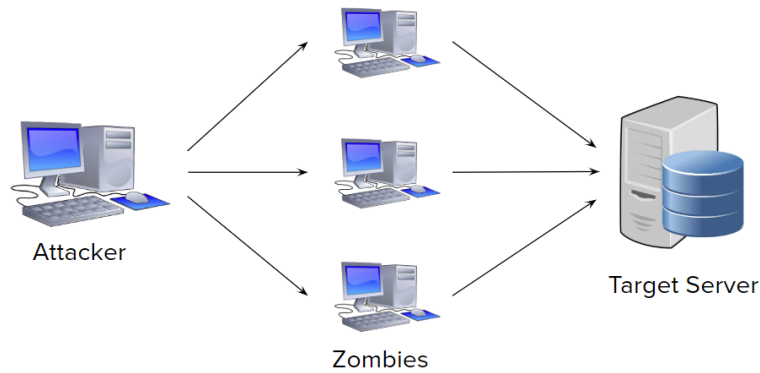


Figure 1.1: Simple scheme of a Distributed Denial of Service attack. The attacker controls multiple zombies which flood the target server making its services unavailable to regular users.

IDSs typically use signature based detection by searching for known attack patterns to identify them, or anomaly based detection, by modeling normal activity and flagging behaviours that deviate from the norm.

With the growth of the diversity of attacks in recent years, machine learning approaches are being widely used. A survey by Hanan Hindy et al. [17] showed that over 80% of studies on intrusion detection take machine learning approaches over statistical and knowledge based approaches.

The usage of machine learning has seen a rise in popularity mainly due to advances in computing power. Its use covers various areas, such as speech and image recognition, search engines, healthcare, finance, and cyber-security[68], specifically in IDSs[60]. A downside of using machine learning techniques to perform classifications is the possibility of existence of adversaries that try to circumvent the classifiers, by searching for "holes" in the classification process of the models. The area that studies these types of attacks is called **Adversarial Machine Learning**, and has been extensively explored in some areas such as image classification and spam detection; its exploration is still small in other areas though, such as intrusion detection[39].

When a machine learning model is trained, it finds patterns in the data which it later uses to perform predictions. By carefully modifying data samples, it is possible to exploit these patterns and cause the model to misclassify samples [67]. These modified samples are called **Adversarial Examples**.

The source of the data can also be an open target, as an adversary can temper the data being used to train the classifier, deteriorating the training process and opening possibilities for adversarial examples.

We performed a thorough study on multiple applications of adversarial machine learning to intrusion and malware detection scenarios, since these have similarities to denial of service scenarios, and we found that a wide variety of adversarial attacks were tested and proven effective at deteriorating the performance of classifiers. However, the variety of datasets in intrusion detection is very small, with only one dataset being used, and the effectiveness of adversarial defense strategies was never tested to the best of the authors knowledge.

1.2 Goals

In this dissertation, we aim to extend a work produced by Ivo Frazão et al. [15], [16] on the detection of denial of service attacks with machine learning classifiers, by performing a study on the behaviour of different machine learning algorithms on adversarial DoS datasets, which are generated using adversarial attacks on regular DoS datasets. Furthermore, we also aim to study different adversarial defense strategies, by analyzing their performance on the previously generated attacks.

To achieve these goals, several experiments were prepared, which are described in more detail in chapter 4, with the objective of answering the following questions:

1. **Can the generation of adversarial Denial of Service datasets deteriorate the performance of classifiers?**
 - We selected two denial of service datasets and trained different classifiers on them. We then tested these models on both normal and perturbed versions of the dataset using cross validation.
2. **Which classifiers offered higher resilience to adversarial attacks?**
 - We used six different machine learning algorithms to perform classification on the datasets.
3. **Which attack strategies are the most effective in each scenario?**
 - We tested four deep learning based adversarial attack strategies to perturb the datasets in order to cause models to misclassify.
4. **Can adversarial defense techniques be effective in a Denial-of-Service context?**
 - We tested the effectiveness of adversarial defense strategies, which are certain architectures of classifiers or techniques that offer resilience to adversarial attacks.
5. **Which defense techniques are the most effective in each scenario?**
 - In total, we tested four different adversarial defense strategies against the adversarial datasets.

1.3 Contributions

During the development of this dissertation, the following contributions were produced:

- Nuno Martins, José M. Cruz, Tiago Cruz, Pedro H. Abreu. *"Analyzing the Footprint of Classifiers in Adversarial Denial of Service Contexts"*. EPIA 2019 Conference on Artificial Intelligence (Accepted).

- Nuno Martins, José M. Cruz, Tiago Cruz, Pedro H. Abreu. "*Adversarial Machine Learning applied to Intrusion and Malware Scenarios: a systematic review*" (Submitted).

1.4 Document Structure

The structure that this document follows is:

- **Chapter 2** - background knowledge of conventional terms and techniques used in machine learning and adversarial machine learning;
- **Chapter 3** - review on applications of adversarial machine learning concepts to intrusion and malware detection scenarios;
- **Chapter 4** - the proposed architecture for the experiments;
- **Chapter 5** - analysis on the results obtained from the experiments proposed;
- **Chapter 6** - conclusions of the dissertation and future work proposal.

Chapter 2

Background knowledge

In this section, various concepts will be presented on the area of machine learning and cybersecurity that will be used throughout this dissertation. We first explore state of the art machine learning algorithms, along with special neural network architectures, and some performance metrics used to evaluate the performance of classifiers. We then explore adversarial machine learning, by first showing a taxonomy that was proposed to distinguish different types of adversarial attacks and attackers, and then show different attack and defense techniques.

2.1 Machine Learning Fundamentals

Machine learning, a narrower field of Artificial Intelligence, comprises the study of algorithms that computer systems mainly use to perform classification tasks without external instructions, by means of trained classifiers.

2.1.1 Machine Learning Tribes

Pedro Domingos distinguishes five tribes of algorithms [13]:

- **Symbolists** - These algorithms essentially focus on inverse deduction. Instead of starting with the premise and looking for conclusions, symbolists start with some premises and conclusions, and try to fill the gaps in between. Among the symbolists, the most notorious are Decision Trees and Random Forests.

A decision tree is a tree-structure resembling a flowchart where every node represents a test to an attribute, each branch represents the possible outcomes of that test, and the leaves represent the class labels. The paths from root to leaves represent the decision rules.

A random forest is an ensemble learning method that builds a large group of independent decision trees, and outputs the mode of the label predictions of all the trees. This method has higher computing costs, but tends to reduce overfitting of the data, which happens when

Background knowledge

a model adapts too strictly to a particular set of data, having poor capabilities for generalization.

- **Connectionist** - This tribe focuses on reverse engineering the human brain. This approach involves creating artificial neurons and connecting them in neural networks. All connectionist algorithms revolve around the usage of neural networks.

Neural networks (NN) are a learning framework that aims to mimic animals brains. The main concept is that a group of neurons are organized among layers, with each layer connected to surrounding layers. The training process consists of adjusting the weights of the connections among layers, until the output of the final layer represents the correct labels. The process of adjusting weights is typically backpropagation, a method that, given an objective function in the last layer, adjusts the weights of the entire network given the gradient of this function in respect to every weight.

- **Evolutionaries** - The evolutionaries focus on applying the idea of genomes and DNA in the process of data. The survival and offspring of units is essentially considered the performance data.

Genetic algorithms are a set of evolutionary algorithms which take inspiration from genetic evolution observed in living beings. The algorithms typically start with a set of individuals (data), and through processes of mutation and reproduction between multiple individuals, new populations are generated. With the progress of the algorithms, the most fit individuals have a higher chance of reproducing, leading to a more fitting population on each iteration.

- **Bayesians** - Focused on applying probabilistic inference, such as the Bayes Theorem, applying "a priori" thinking, with the belief that different outcomes have different probabilities.

Naive Bayes (NB) is a machine learning algorithm that consists of applying the Bayes theorem in order to find a distribution of conditional probabilities among class labels, with the assumption of independence between features. It has the advantage of being highly scalable.

- **Analogizers** - Focuses of the idea that close elements are more strongly related, essentially matching related pieces of data. Among the analogizers, the most popular ones are K-Nearest-Neighbours (KNN) and Support Vector Machines (SVM).

KNN is a classification algorithm that uses a distance function (e.g. Euclidean distance) in order to find the K closest elements in the feature space. The final label will depend on the mode of the distribution of classes in those neighbours.

SVM is a binary classification algorithm which creates an hyper plane that separates the data. On each side of the hyper plane are the classes used for classification, and the objective is to maximize the gap perpendicular to the plane, allowing better generalization. Through the usage of kernel functions (e.g. Gaussian), it is possible to generate hyper planes in higher dimensions, when the classes are not linearly separable.

2.1.2 Neural Network Architectures

In this section we'll explore some specific types neural networks that are used either in the literature review or in this dissertation's experiments.

2.1.2.1 Autoencoders

Autoencoders (AE) are a type of neural network that aims to reconstruct data from the input layer into the output layer with a minimal amount of distortion [62]. Since the objective function is calculated with a comparison between the input and output, autoencoders are an unsupervised learning algorithm, since they don't require class labels to be trained.

An AE is composed of an encoder, a decoder and a bottleneck layer. The function of the encoder is to compress the input to a more compact representation, which is present in the bottleneck layer, and the function of the decoder is to reverse this compact representation back to the original one. The bottleneck layer connects the encoder and the decoder (figure 2.1), and possesses less neurons than the input and output layers. This prevents the network from perfectly copying the input layer to the output layer, and forces it to generalize. The compact representation of the data in the bottleneck layer is also often more robust to signal noise [62].

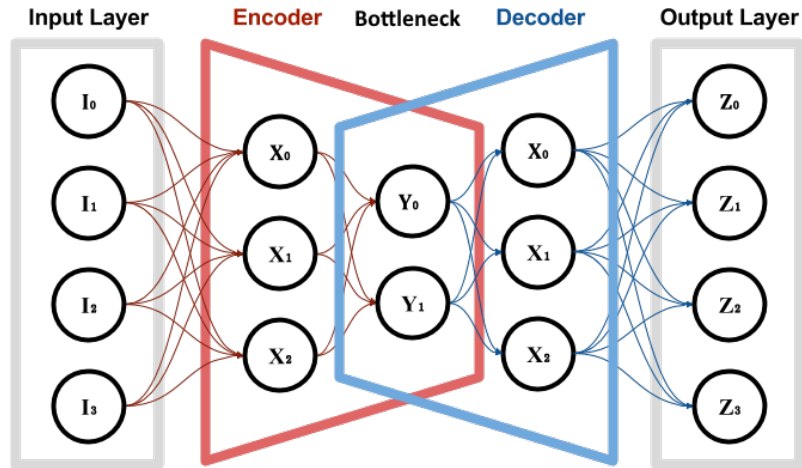


Figure 2.1: Scheme of a basic autoencoder.

2.1.2.2 Denoising Autoencoders

Denoising autoencoders (DAE) are an extension of regular autoencoders, which are used mainly to remove noise from the input, instead of obtaining a simpler representation, as do normal AEs [69]. This is achieved by initially corrupting the data before feeding it to the input layer during training process. This corruption can be done through a multitude of ways, such as setting random features to 0, or applying gaussian noise (noise following a normal distribution). After the network makes its prediction, the loss function is calculated in respect to the original data (not to the corrupted sample), which trains the network to reconstruct the original input. Since the network is thus

often incapable of perfectly reconstructing the inputs because of the noise, a larger hidden layer can be used (figure 2.2), which allows the capture of noise robust features, representing the joint distribution of the inputs for a more precise reconstruction.

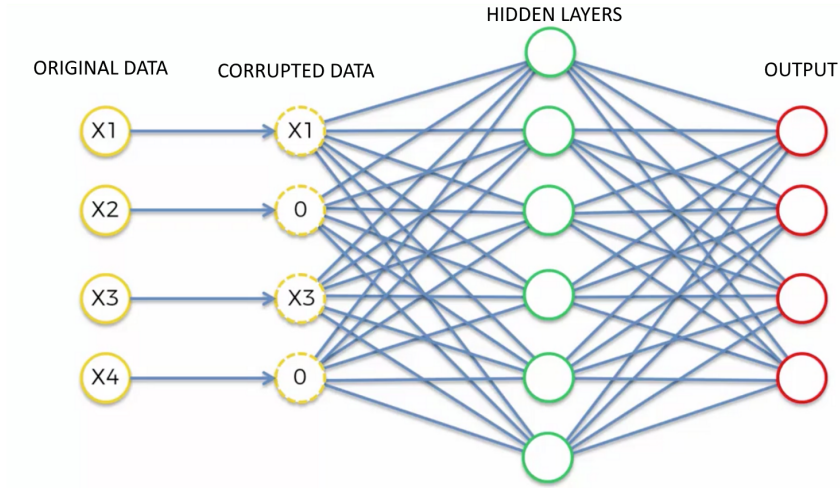


Figure 2.2: Scheme of a denoising autoencoder.

On this dissertation, denoising autoencoders were used to perform detection of adversarial examples, as these are many times considered normal samples with noise added, and DAEs have the potential to partially remove this noise.

2.1.2.3 Generative Adversarial Networks

Generative adversarial networks consist of two neural networks that are matched against each other [57]. One of the networks, called the **Generator**, produces samples by multiple possible means, such as by introducing noise to valid samples, or producing them from scratch using only a random latent variable as seed. The other network, called the **Discriminator**, receives real samples and adversarial samples created by the Generator, and tries to distinguish them. A loss function is then calculated based on the predictions of the Discriminator, which is then used to adjust the weights of both the Discriminator and the Generator (figure 2.3).

In essence, the two networks are playing a game, by improving each other through competition: the Generator tries to create samples that will deceive the Discriminator, and the Discriminator tries to correctly classify the samples, and a single loss function is used to train both models.

One of the biggest problems with regular GANs is that using exclusively gradient descent to train the networks often leads to mode collapse to a parameter setting, always emitting the same point [50]. So, the optimization becomes highly unstable [28]. Wasserstein GANs were introduced as a means to solve this problem. Instead of using the gradient of the loss function of the neural networks to train the models, they used the Wasserstein distance, also known as the Earth-mover

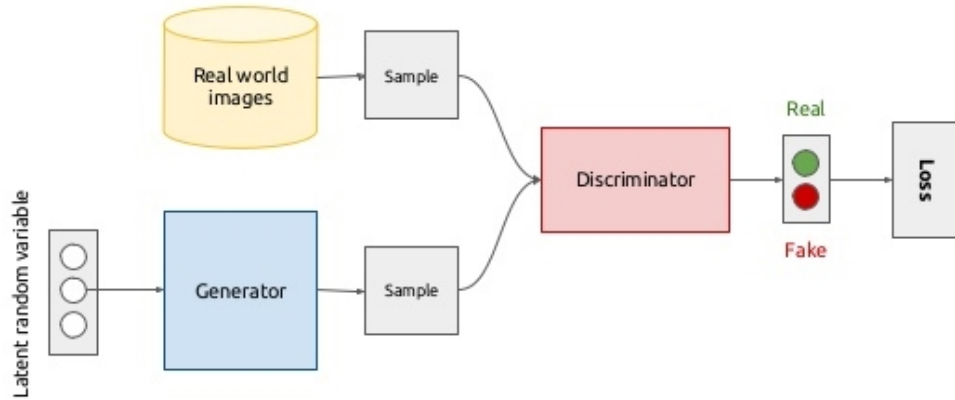


Figure 2.3: The general flow of a GAN. Only one loss function is used, which is be used to train both the generator and the discriminator.

distance, which measures the distance between two distributions [28]. This technique was shown to eliminate many of the shortcomings of vanilla GANs, providing a smoother training [28].

2.1.3 Transfer Learning

Transfer learning is an area of machine learning which focuses on the transfer of knowledge between models. This range of techniques is especially useful in computer vision, where training a new neural network from scratch can prove a very expensive task, and by reusing networks previously trained in other tasks, a lot of time and computing power is saved, with minimal loss in the final performance. This takes advantage of the sequencality in the abstraction of features that occurs in neural networks. The first layers represent simpler features, such as edges and simple shapes, and the last layers represent more complex forms. When both tasks are similar in nature (e.g. image recognition), the first layers can be re-utilized.

Distillation is a technique initially designed to train a neural network with knowledge extracted from another neural network, with the intent of reducing the computational complexity of the classifier in order to deploy it in more constrained devices, such as smartphones, or other devices without fitting hardware [53].

The process of Distillation consists of creating a new simpler network that mimics a teacher network, by approximating its output to the output of the teacher network (the loss function of the learning network is a comparison between its output and the output of the teacher). The problem is that, on a vastly trained teacher network, the class distribution will be very uneven, and the correct classes will have very high values, while the wrong ones will be close to 0, which results in an output that exclusively represents the ground truth labels. A new concept is then introduced called **Softmax Temperature**, that softens the output of the teacher model, distributing it more

evenly throughout other classes, facilitating the training process for the learning network (figure 2.4), since it facilitates learning the classes distributions for inputs.

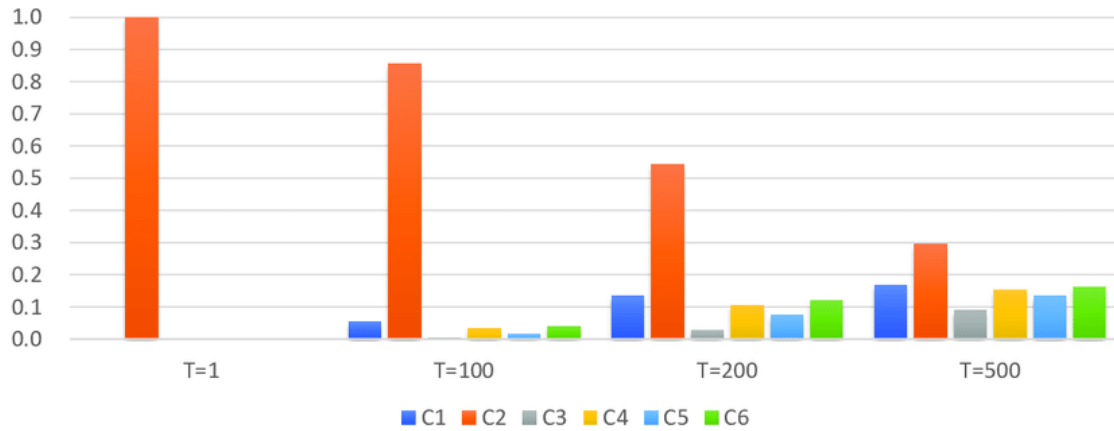


Figure 2.4: Distillation: Softmax temperature illustrated.

Distillation can be used to create more robust classifiers to adversarial examples, as will be later explored.

2.2 Performance Metrics

Evaluating the performance of a classifier is an essential factor on any project. Several metrics have been proposed to evaluate the performance of models, with each having its advantages and disadvantages [54].

2.2.1 Classification Metrics

Classification is the process of identifying the set categories to which a certain sample belongs to, such as identifying if a connection is malicious or not. There are a wide variety of metrics that can be used to evaluate the performance of a classifier, each with its advantages and disadvantages.

Confusion matrixes can be used on binary classification problems, and give us numerical values of the predictions of the classifier, with the diagonal containing the correct predictions, and other cells containing failed predictions (table 2.1).

Table 2.1: Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	Number of True Positives	Number of False Negatives
Actual Negative	Number of False Positives	Number of True Negatives

Accuracy is the simplest form of evaluation, and consists of the ratio of correct and total predictions.

Background knowledge

$$Accuracy = \frac{Correct\ predictions}{Total\ Predictions}$$

Its result is easy to interpret, but it can often be misleading when the ratios between different classes are very different.

For example, if there are 98 samples of one class, and 2 of another class, and the classifier predicts all 100 at being from the first class, we will achieve an accuracy of 98%, which can be misleading because it doesn't indicate that the model is effective, since all it did was predict the first class for every sample.

Error rate is similar to accuracy, but instead measures the ratio of incorrect and total predictions.

$$Error\ rate = \frac{Incorrect\ predictions}{Total\ Predictions}$$

Sensitivity, also known as True Positive Rate, refers to the fraction of positive samples that are correctly classified. Specificity, also known as True Negative Rate, measures the fraction of negative samples that are correctly classified.

$$Sensitivity = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives}$$

Precision indicates the fraction of positive samples from all the samples predicted as positive by the model. Recall indicates the fraction of positive samples that are correctly classified from all the correctly classified samples.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + True\ Negatives}$$

F1 represents the a weighted average between Precision and Recall, and is calculated with the harmonic mean between **Precision** and **Recall**.

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Area Under Curve (AUC) is a popular evaluation metric which combines **TPR** (also known as **Sensitivity**) and **FPR** into one value.

The value of AUC is obtained by plotting the TPR against FPR, producing the Receiver Operating Characteristics (ROC) curve, and calculating the area under the curve, and the main advantage over accuracy is that it is scale invariant.

2.2.2 Performances Comparison

There have been proposed several techniques to compare the performance of two or more algorithms across different datasets [70]. For this dissertation, we chose to use the **Friedman Test**, since its non-parametric.

The test consists of ranking each algorithm for every testing dataset, followed by averaging out the ranks for each algorithm. The **Nemenyi Test** can be performed afterwards to test if we reject the null hypothesis which states that two algorithms have similar performance. This can be performed by computing the critical difference (CD):

$$CD = q \sqrt{\frac{k(k+1)}{6N}}$$

Here, q is a pre-computed critical value for a certain confidence level (we used 95%), k is the number of classifiers, and N is the number of testing sets. If the difference between the average ranks of two classifiers is above the critical difference, we can reject the null hypothesis for that pair.

2.3 Adversarial Machine Learning

Adversarial machine learning comprises a group of techniques that aim to produce malicious inputs that cause models to misclassify them.

In this section we will explore theory and taxonomies on adversarial machine learning, along with adversarial attack and defense strategies.

2.3.1 Fundamentals of Adversarial Machine Learning

Huang. proposed a formal taxonomy in order to model an attack in adversarial environments [64], introduced models for the adversaries knowledge about the system, and analyzed application-specific factors.

According to the taxonomy proposed, an attack can be classified by the following properties:

- **Influence** - refers to the influence the attack has over the training data. It can either be a **Causative** attack if it can influence the training data of the classifier (Poison attacks), or **Exploratory**, if the attack only aims to probe the training data in order to obtain information that can prove useful in the process of the attack.

Background knowledge

- **Security Violation** - refers to the violation that the attack aims to achieve. These can be **Integrity** attacks when the aim is for the attack to be classified as normal (False Negative), **Availability** attacks, when the aim is to cause classification errors of any type (False Negative or False Positive), rendering the model useless, or **Privacy** attacks, when the aim is to obtain information from the learner, violating the users privacy.
- **Specificity** - refers to target of misclassification. These can be **Targeted** attacks when the intent is for the attacks to be misclassified into a certain class/group of classes, or **Indiscriminate** when there is no specific class to be targeted, and the objective is only to cause any False Negative.

Furthermore, an analysis of the knowledge the attacker can have over the system is also important for the detection system. Knowledge of the underlying model can provide the adversary with means to exploit the system and make informed decisions. The knowledge of the underlying system can be formalized into:

- **Machine learning algorithms** - refers to the knowledge of the algorithm(s) used by the detection system.
- **Data source** - the knowledge of the source of the training data used to prepare the classifier(s).
- **Data** - knowledge of the exact data used to train the classifier(s).
- **Feature Selection** - choice of features for the models.
- **Output** - refers to the knowledge the attacker has over the output by the classifier over certain inputs.

Overall, the attacks can be classified as white-box attacks or black-box attacks, depending on the knowledge the attacker has over the system. This can be an important information, as lack of knowledge of the system imposes restrictions on the techniques an attacker can use to generate adversarial attacks.

Another limitation that must also be considered when preparing a detection system are the limitations of the application and the environment. These limitations are expressed in the features used in the classifiers. Features are the variables that represent the world space and can be used by the classifier to perform evaluations. Several limitations can arise when an adversary attempts to manipulate features in order to deceive the classifier. These can be organized as:

- **Domain Limitations** - refers to the limitations an attacker faces when interacting with an application domain, which includes how the user can interact with the application, and what features can realistically be changed by the attacker. For example, when a classifier is used exclusively by an individual to detect spam email, an assumption can be made that the user will not willingly corrupt the data in order to deteriorate the classifier.

- **Contrasting Feature Spaces** - these refer to the limitations an attacker faces when altering features. Depending on the domain, there are different limitations to how an attacker can alter the features. The attacker might be able to alter a wide variety of features (e.g. including or not a word in an email), while being limited in altering others (e.g. faking spam email addresses as real ones). There is also a limitation on how much the attacker can alter a certain feature. For example, during a Denial of Service attack, if the time between packets is too long, the attack might prove ineffective.
- **Contrasting Data Distributions** - refers to the limitation that appears in the conflict between the distributions of data from an adversarial input and a regular input. Certain features regularly follow certain distributions, and when the adversarial input doesn't fit them, it is more easily flagged. For example, in the detection of spam emails, the distribution of words of regular emails follows the same pattern as the most used words globally. When an attacker forges adversarial emails, he is limited in the choice of words, by not being able to use many unconventional ones, as they are more easily flagged.

2.3.2 Adversarial Attacks

Several adversarial attack techniques have been proposed, with a trade-off on performance, complexity, computational efficiency and application scenario (black-box and white-box).

In white-box attacks, the attacker has knowledge over the training data, model parameters, and other useful information about the classifier. In black-box scenarios, the attacker has little to no knowledge of the classifier, and thus is severely limited.

Most of the attack methods presented were only tested on image domains, but most can equally be applied to tabular data, such as an intrusion dataset, since they are not data dependant attacks, and a feature normalization step can squeeze features to a certain interval, much like the range of available values of each pixel on an image, creating a similar scenario to images. This poses a security threat if an attacker can manipulate multiple features used by the classifiers, allowing the application of adversarial attacks to manipulate intrusions or malware to go undetected by the classifiers.

Gradient based attacks typically introduce perturbations optimized for certain distance metrics between the original and perturbed samples. The three mainly used distance metrics in literature are:

- L_∞ - minimize the maximum amount of perturbation introduced to any feature of the sample;
- L_0 - minimize the amount of features perturbed of the sample;
- L_2 - minimize the Euclidean distance between original and adversarial samples.

In table 2.2, there is a list of datasets that were used in the following studies.

Table 2.2: Datasets used in literature review.

Name	Type
MNIST [7]	Handwritten numbers
GTSRB [4]	Traffic Signs
CIFAR10 [2]	Object recognition
NSL-KDD [51]	Intrusion dataset
Kaggle Microsoft Malware Classification Challenge [6]	Malware dataset

2.3.2.1 Limited-memory Broyden–Fletcher–Goldfarb–Shanno Attack

Szegedy et al. proposed one of the first gradient methods to generate adversarial examples using **box constrained Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)** optimization [59]. Given an input image, this method searches for a different image that is similar to the first, under L_2 distance, that is mislabeled by the classifier. By adding noise to the original image, the problem can be represented as a minimization problem, where the objective is to minimize the distance between the original image and the adversarial one:

$$\min ||r||^2 \text{ subject to :}$$

$$f(x+r) = l$$

Here, x is the original image, r is the perturbation, f is the loss function of the classifier and l the incorrect prediction/label. Since this is a non-trivial problem, the authors approximate it by using L-BFGS optimization algorithm to solve it (the box constraint is needed to limit the possible values of the variables, which are pixels in image scenarios). Although this method is effective at producing adversarial examples, it is impractical, since it uses a computationally expensive algorithm to search for an optimal solution.

2.3.2.2 Fast Gradient Sign Method Attack

Goodfellow et al. proposed a simple and fast gradient based method to generate adversarial examples called **Fast Gradient Sign method (FGSM)** [52], with the aim of minimizing the maximum amount of perturbation added to any pixel (L_∞ distance metric) of the image to cause misclassification. The rationale of this technique is to compute the gradient of the loss function with respect to the input (e.g. using backpropagation), and alter the existing data by adding a perturbation ε with the sign of the gradient. The formula for the attack is the following:

$$\eta = \varepsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

where η is the perturbation, ε is a hyper-parameter controlling the magnitude of the attack, J is the cost function, ∇_x is the parameters of the model, x the input and y the target label (Figure

2.5). By increasing the value of ϵ , the magnitude of the attack becomes greater, but the image will become more noticeably distorted.

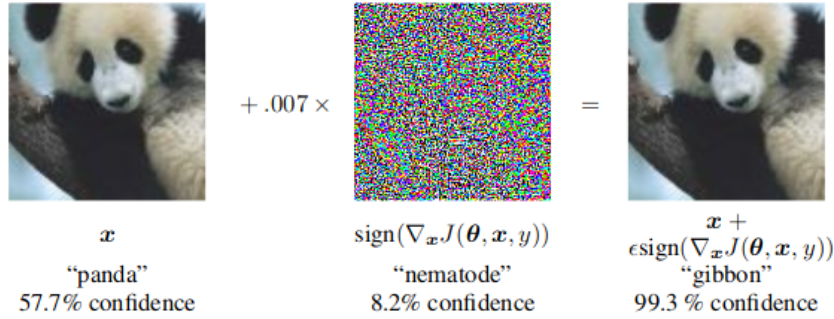


Figure 2.5: FGSM - By using a ϵ of 0.007, which only allows at most 1 bit to be changed in a 8 bit representation of each channel of a pixel, the perturbation can lead to a misclassification.

Although this method is less effective than other state of the art techniques for generating adversarial attacks and is noticeable to humans, it is one of the most efficient at computing time, allowing fast generation of adversarial examples.

2.3.2.3 Jacobian Based Saliency Map Attack

Papernot et al. proposed a new adversarial sample generation technique called **Jacobian based Saliency Map Attack (JSMA)** [56], that uses feature selection, unlike the previous methods, with the aim of minimizing the number of pixels modified (L_0 distance metric) while causing misclassification.

This method revolves around the computation of saliency maps for an input sample, which contain the saliency values for each input feature. These value indicates how much the modification of that feature will perturb the classification process, and is different for each target class. The features are then selected in decreasing order of saliency value, and each is modified accordingly by ϑ . The process finishes when misclassification occurs, or a threshold number of modified features is reached (figure 2.6).

We can conclude that, compared to FGSM, this method is able to generate adversarial inputs with similar success rate, but vastly less feature modification, at the expense of higher computing cost.

2.3.2.4 Deepfool

Moosavi-Dezfooli et al. proposed an untargeted adversarial sample generation technique called **DeepFool** [55], with the aim of minimizing the euclidean distance between perturbed samples and original samples (L_2 distance metric).

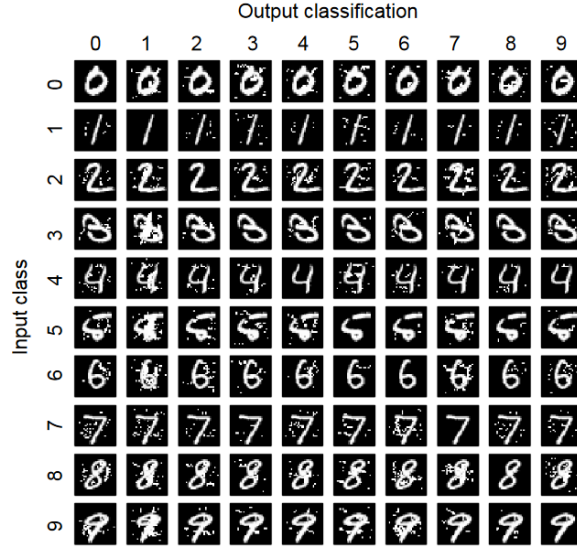


Figure 2.6: JSMA on MNIST dataset. The original samples are found in the diagonal, with all other elements being adversarial examples.

The generation of the attack consists of the analytical calculation of a linear decision boundary that separates samples from different classes, followed by the addition of a perturbation perpendicular to that decision boundary (figure 2.7).

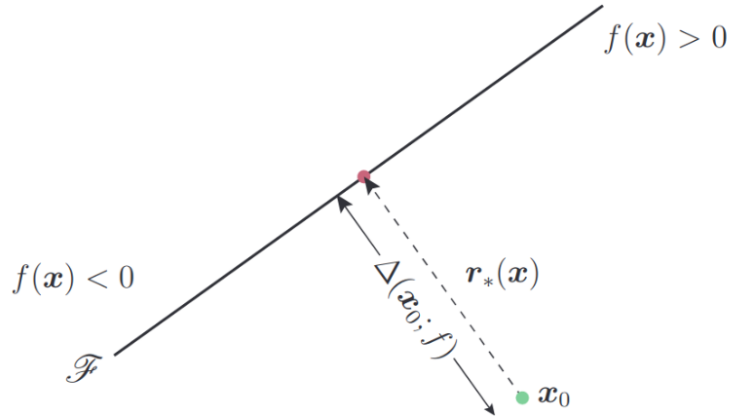


Figure 2.7: Deepfool - a sample x_0 is added a perturbation r in the direction of the hyperplane that separates the original class and the target class.

In neural networks, these decision boundaries are almost always not linear, so they add the perturbations iteratively by performing the attack multiple times, finishing when an adversary is found. The overshoot parameter is used as a termination criterion to prevent vanishing updates. Although this method allows the generation of adversarial samples with less perturbations than

FGSM and JSMA, and with higher misclassification rate, it is more computationally intensive than both.

2.3.2.5 Carlini & Wagner Attack

Carlini and Wagner created a new attack based on the L-BFGS attack, called **Carlini & Wagner attack (C&W)** [30], by representing the attack as an optimization problem. Instead of directly using the loss function in the optimization process, the authors propose other objective functions, study different distance functions for each of the three distance metrics, and test other optimizers that don't use box constraints by using the hyperbolic tangent ($\tanh$) to find the optimal perturbations in the specified ranges. The authors observed that the attack was able to defeat multiple adversarial defense techniques, such as defensive distillation.

2.3.2.6 Generative Adversarial Networks

Generative Adversarial Networks (GAN) have also been used to generate adversarial attacks.

Chaowei Xiao et al. created a modified version of GANs called AdvGAN[23], and were able to successfully generate adversarial examples. Instead of only having a generator and a discriminator, the authors also included the target classifier, or an approximated version of it using transfer learning, on the GAN. They then used the sum of the loss functions of the discriminator and the classifier to create one loss to be used to train the generator. Both losses are used because the loss of the discriminator indicates how far the generated samples are from the real samples, forcing the generator to add smaller perturbations, while the target classifier loss is used to evaluate the success of the misclassification of those samples (figure 2.8).

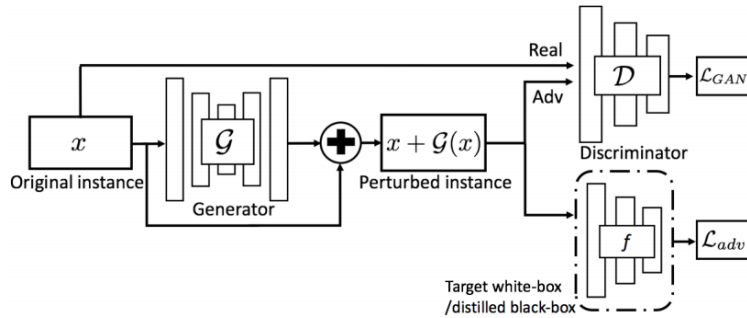


Figure 2.8: Structure of an AdvGAN. The loss function used to train the generator is the combination of the loss of the discriminator and the loss of the target classifier.

Although the generation of adversarial samples with GANs is effective, the distribution of the perturbations is more unpredictable than the previously mentioned gradient based methods, which all introduce perturbations under a certain distance metric, and the generation can also be highly unstable, even for Wasserstein GANs [12].

2.3.2.7 Zeroth Order Optimization Attack

Zeroth-order Optimization attack (ZOO) was proposed by Chen et al. [32], and allows the estimation of the gradient of the classifiers without access to the classifier, which is ideal for a black box attack. The method consists of iteratively adding perturbations to each feature of the samples and querying the classifier to estimate its gradient and Hessian of these different features. This makes it an oracle based technique, not requiring the training of substitute models, and the attacker doesn't require information on the classifier. Although this method was proven effective at estimating the gradient, it can require a large amount of queries to the oracle, which can be used to detect the attacker.

2.3.2.8 Transferability of Adversarial Attacks

Goodfellow et al. explored an interesting property of adversarial inputs, which is transferability across models [52]. The authors observed that, when an adversarial input is successfully misclassified by a model, it will often be misclassified by other models, even when they have different architectures or were trained in different datasets. It was also observed that they often agree on the predicted class. This allows an attacker to perform black-box attacks by training their own substitute model, either by using transfer learning techniques or training on the same dataset, and using that model to generate adversarial samples that can target that classifier [49].

2.3.3 Adversarial Defenses

There are a wide variety of approaches that can be taken to make classifiers more resilient to adversarial attacks. These countermeasures can follow two approaches [42]:

- Reactive Defenses: focus on detecting adversarial examples separately from an already trained classifier;
- Proactive Defenses: focus on building classifiers that are robust to adversarial examples.

2.3.3.1 Adversarial Training

Goodfellow et al. proposed **Adversarial Training** [52], the first proactive defense method to improve resilience to adversaries which consists of including adversarial examples in the training set. Instead of directly including adversarial examples in the training set, the authors propose the usage of an adversarial objective function, which replicates the same effect. The formula used by the authors is the following:

$$\tilde{J}(\theta, x, y) = \alpha * J(\theta, x, y) + (1 - \alpha) * J(\theta, x + \text{sign}(\nabla_x J(\theta, x, y)))$$

Here, $\tilde{J}$ is the final objective function, which is the sum of the loss on the normal sample and the loss on an adversarial sample using FGSM, with α controlling the influence of each loss function on the final loss.

The authors concluded that this technique was effective at regularizing the classifier, although misclassifications still occurred. The main disadvantage of this method is that it needs to be trained against specific types of attacks to be resilient to them, being mostly effective against L_{inf} attacks, such as FGSM [37]. The authors were able to reduce the error rate from 89.4% to 17.9% using adversarial training against FGSM on the MNIST dataset [52]. Swami et al. further explored this technique by showing superior effectiveness when introducing perturbations on intermediate layers of deep neural networks instead of the input layers [40].

2.3.3.2 Gradient Masking

Gradient Masking comprises a group of proactive defensive techniques which assume that "if the model is non-differentiable or if the model's gradient is zero at data points, then gradient based attacks are ineffective"[14].

One form of gradient masking is Gradient Hiding, which consists on using non differentiable models to perform classification, such as Decision Trees. This prevents the adversary from estimating the gradient and using it to generate adversaries.

Gradient Smoothing comprises a range of techniques that smooth out the model's gradient, causing numerical instabilities which make the estimation of gradient more difficult.

Although the objective of preventing the adversary of estimating the gradient was achieved, Papernot et al. concluded that, both under black-box and white-box scenarios, the training of substitute classifiers to estimate the gradient is an effective technique to generate adversaries because of the transferability property, making gradient masking an ineffective defense technique [47], [48].

2.3.3.3 Defensive Distillation

Another proactive defense technique was proposed by Papernot et al. called **Defensive Distillation** [46]. The authors propose the usage of distillation as a means to improve the robustness of a classifier to adversarial inputs. Instead of using distillation in the conventional way to train a different, simpler model from a teacher model to allow deployment on resource constrained devices, it is proposed to use the knowledge extracted during the distillation process to improve the classifier itself at detecting adversarial samples. By increasing the temperature at which a network is trained on the softmax layer, the teaching network outputs have less confidence on the ground truth label and show a higher distribution across classes. It is then proposed to use this output to train a second network with the same structure as the first. The structure is not changed since the objective is to make the network more resilient, and not more computationally efficient. The authors observe that, by using the entropy generated in the softmax layer to train the same network, it will become more resilient to adversarial samples, as it prevents the network from fitting too tightly to the ground truth class labels. An overview on the proposed technique is in figure 2.9.

To test the proposed method, the authors used MNIST and CIFAR10 datasets, and the results showed that distillation reduces the success of adversarial samples from 95.89% to 0.45% on the

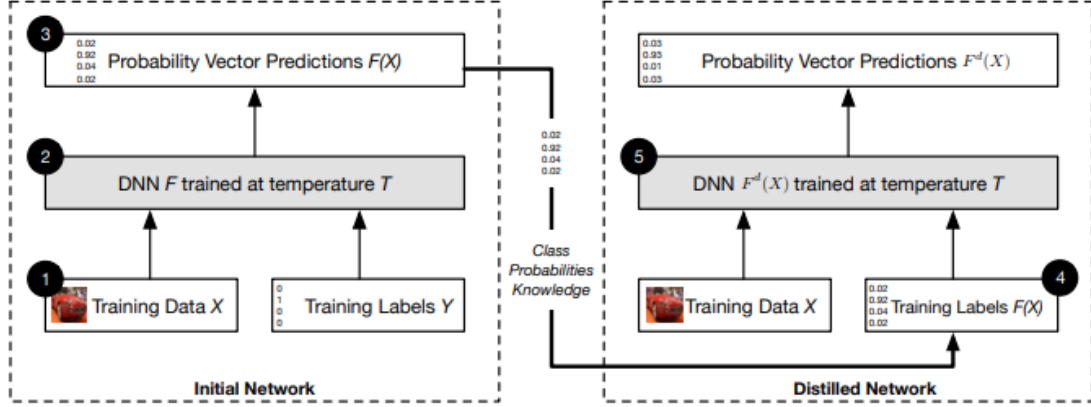


Figure 2.9: Overview of defensive distillation technique. A neural network F was initially trained and performed predictions under a certain temperature T . The predicted labels distribution is then used to train a second neural network with the same , which is theorized to be more resilient to adversaries.

MNIST dataset, and 78.9% to 5.11% on the CIFAR10, with variations of accuracy in the detection of normal samples in the range of -2% and 2%.

2.3.3.4 Feature Squeezing

Weilin Xu et al. proposed **Feature Squeezing** [41] as a means to combat adversarial examples. The intuition behind this technique is to compress the features of the sample (the pixels of the image in their case) and perform classification on the compressed sample. If the prediction by the classifier on the compressed sample is substantially different from the prediction of the original sample, it is considered an adversarial example. The authors tested several compression methods, namely Bit depth compression, Median Smoothing and Non local means, but found that the best method was largely dependent on the dataset. An overview of the feature squeezing framework can be seen in figure 2.10.

This technique was proven effective on MNIST, CIFAR10 and ImageNet datasets against many state of the art adversarial attacks, such as FGSM, JSMA, Deepfool and C&W under different distance metrics, showing an improvement between 20% and 70% on the detection rate for all datasets, with Bit Depth squeezing being the most effective on the MNIST dataset and Median Smoothing the most effective in CIFAR10 and ImageNet.

2.3.3.5 Transferability Block

Hossein Hosseini et al. proposed a new proactive technique to block the transferability of adversarial examples across different models on black-box environments[35].

It was previously shown that adversarial examples crafted to target a specific classifier often work on different classifiers, even if their training data or architecture are different. Instead of

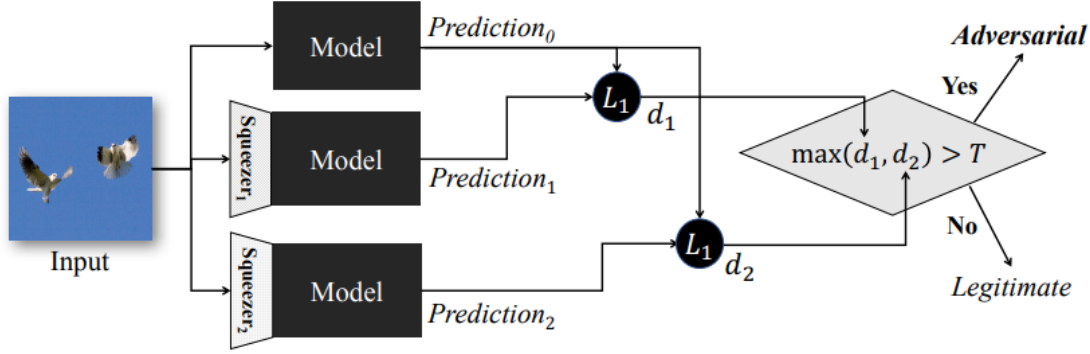


Figure 2.10: Framework of feature squeezing. When the difference of predictions on different models using different/no squeezers exceeds a threshold, the input is considered adversarial.

trying to create a defense mechanism that assigns adversarial examples to their original label, the authors propose discarding adversarial examples, by creating a new class called NULL, which indicates if the input is suspicious.

The authors start by training a classifier exclusively with clean data. After the initial training, they compute a function that outputs the NULL probabilities, which represent the probability of a sample being adversarial based on the amount of perturbations present. They introduce perturbations using a brute-force method, which iteratively modifies certain features, and repeat this process for multiple number of features perturbed. The third and final step is to perform adversarial training, and deciding the label based on the NULL probabilities (figure 2.11).

Labels	0	1	2	3	4	5	6	7	8	9	NULL	
Original Labeling	1	0	0	0	0	0	0	0	0	0	0	
NULL Labeling	1	0	0	0	0	0	0	0	0	0	0	
Original Labeling	0.7	0	0.3	0	0	0	0	0	0	0	0	
NULL Labeling	0.7	0	0	0	0	0	0	0	0	0	0.3	
Original Labeling	0.5	0	0.5	0	0	0	0	0	0	0	0	
NULL Labeling	0.5	0	0	0	0	0	0	0	0	0	0.5	
Original Labeling	0.1	0	0.9	0	0	0	0	0	0	0	0	
NULL Labeling	0.1	0	0	0	0	0	0	0	0	0	0.9	

Figure 2.11: Transferability defense: instead of assigning a larger credibility to the target class during training, the network is trained to assign it to the NULL label, proportionally to the magnitude of the perturbations.

Although this method is not intended to identify the true label of a certain adversarial sample, it is one of the most effective techniques in literature at identifying adversarial examples.

The results show the transferability rate of different attacks on the MNIST and GTSRB datasets reduced from over 55% to less than 6%.

2.3.3.6 Universal Perturbation Defense Method

Naveed Akhtar and Jian Liu proposed the **Universal Perturbation defense method** as a defense method against adversaries [25]. The method consists of placing a *Perturbation Rectifying Network* (PRN) before the input layer of the target classifier. This network is trained using images with and without perturbations, and the layers of the classifying network frozen. This process creates a network that is effectively denoising the inputs and, by evaluating the difference on the inputs and the outputs of the PRN, extracts discriminative features, which are used to train a binary classifier that is able to identify an adversarial sample. The results on the GoogleNet dataset show that the detection rate of adversarial attacks was between 91.6% and 97.5%, both against L_{inf} and L_2 attacks.

2.3.3.7 MagNet

Dongyu Meng and Hao Chen proposed a defensive framework called **MagNet** [38]. The framework consists of two main modules: a detector and a reformer.

The authors consider that a classifier misclassifies an adversarial attack for two reasons: the example is distant from the boundary of the manifold of the normal examples, or the example is close to the manifold of normal examples, but the classifier doesn't generalize well.

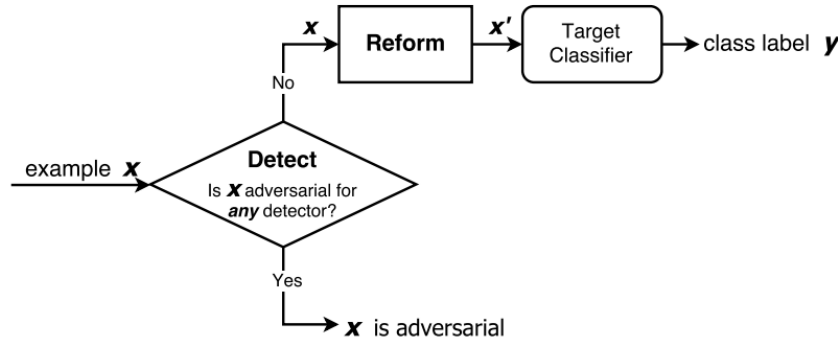


Figure 2.12: MagNet defense strategy. Using an ensemble of detectors, if any considers a sample to be adversarial, it is considered to be adversarial, removing samples with high magnitude perturbations. Regular samples and adversarial samples with small perturbations are then directed to the reformer, which denoises the data and directs it to a classifier that performs predictions.

The purpose of the detector is to defend against the first case (adversarial examples distant from the manifold), and it is built using an autoencoder. The autoencoder is trained on normal

samples, and during deployment, it rejects samples that deviate substantially from the manifold. It does this by verifying the loss function, which is the mean squared error, and rejects samples with an error above a threshold, which was in this case the maximum loss observed during training.

The reformer then receives samples that were classified as normal by the detector, and "denoises" the inputs, to remove small perturbations that were not caught by the detector, and approximate the adversarial samples to normal samples. The output of the reformer is then input on a classifier, which will perform classification among the normal classes.

The authors tested the proposed technique on the MNIST and CIFAR datasets, using FGSM, IGSM (an iterative variation of FGSM), DeepFool and C&W. All attacks were effective at decreasing the performance of the classifiers, with C&W being the most effective by reducing the accuracy to 0%, and DeepFool to 19.1%. After deploying the defense, all accuracies improved, with all being above 92% on the MNIST dataset, and above 77.5% on the CIFAR dataset.

2.3.3.8 Separate Binary Classifier

Zhitao Gong et al. proposed an alternative to adversarial training which consists of training a separate classifier to exclusively detect adversarial examples [33]. Since the authors didn't formally name the proposed defense, we use the term "**Separate Binary Classifier**" to identify it. Instead of including adversarial examples on the training set, and training the classifiers to detect the original label of the adversarial examples, the authors propose training a separate binary classifier to detect if a sample is adversarial or not. Samples that are considered adversarial are ignored, while samples considered clean are passed on to a classifier trained on normal samples. The main advantage of this technique over adversarial training is that it serves as a pre-processing step, not requiring modifications on an already trained model, making it easy to deploy.

After testing the technique on MNIST and CIFAR datasets, while attacking them with FGSM attack, the authors achieved over 99% accuracy on both datasets, although when they tested on perturbations with smaller magnitude than the ones present in the training set, the accuracy drastically dropped to 0.003%, meaning that there is a limitation on the generalization of the model, which the authors argued is also a problem in adversarial training and distillation.

2.4 Conclusions

The initial part of our experiments is to use regular classifiers to perform classification on normal and adversarial data. To do this, we chose to use at least one classifier from each tribe according to Pedro Domingos [13]: **Decision Tree**, **Random Forest**, **Naive Bayes**, **Support Vector Machine**, **Neural Network** and a **Denoising Autoencoder** combined with a Neural Network. We didn't include evolutionary algorithms since those are not optimal to perform classifications.

To generate adversarial attacks, we decided to use at least one attack for each distance metric: **Fast Gradient Sign Method**, **Jacobian Based Saliency Map Attack**, **Deepfool** and **Carlini & Wagner Attack**. We included C&W to test its effectiveness against distillation defense, since it was claimed by the authors that this attack could defeat it.

Background knowledge

Finally, we decided to use four defense strategies: **Adversarial Training**, **Separate Binary Classifier**, **MagNet** and **Defensive Distillation**. In this selection we were more constrained, since most other defenses, such as Feature Squeezing, are more specific to image scenarios.

Background knowledge

Chapter 3

Literature Review

In this chapter, we explore different works that have applied adversarial machine learning to intrusion and malware detection scenarios. We didn't specify our search to Denial of Service contexts, since there are very few works that apply adversarial machine learning specially to this case, and because these scenarios are very similar in application of the strategies. The search was performed on multiple science databases, such as **IEEE Xplore**, **Springer**, **arXiv**, **ScienceDirect** and **Research Gate**, as well as regular search engines such as **Google**, using the keywords "*Intrusion Detection*", "*Malware Detection*" and "*Adversarial Machine Learning*". We selected all articles found on intrusion detection, and selected the most cited on malware detection.

Maria Rigaki and Ahmed Elragal first tested the effectiveness of adversarial attacks in an intrusion detection scenario [39]. The authors performed tests on the NSL-KDD dataset [51], by using **FGSM** and **JSMA** to generate Targeted attacks, and used 5 models to perform classification: Decision Tree, Random Forest, Linear Support Vector Machine, Voting ensembles of the previous three classifiers and a Multilayer Perceptron. The results on JSMA showed that all classifiers accuracy was affected, with Linear SVM being the most substantial one with a drop of 27%. The drop on F1-score and AUC was also notable, especially on the Linear SVM and Voting ensemble. Overall, the most resilient classifier was Random Forest, which suffered smaller performance drops across all metrics, with a drop of 18% on the accuracy and 6% on the F1 score and AUC. The authors made an important remark on the percentage of features modified by the attacks: FGSM modifies 100% of the features on every sample, while JSMA only modified on average 6% of the features. This makes JSMA a more realistic attack, since in the field of intrusion detection, there are domain specific limitations to the attacker relating to what features he can modify, which is coherent with the taxonomy of Huang et al. [64]. It was also noted that, although the explored attacks proved their effectiveness against unknown classifiers, they still required knowledge about the preprocessing of the data, such as the features used for classification, and the effectiveness of attacks under other circumstances was not tested.

Zheng Wang also tested the performance of adversarial attacks on the NSL-KDD dataset [51], using four different adversarial attack techniques to attack a Multi Layer Perceptron classifier

(MLP): **FGSM**, **JSMA**, **Deepfool** and **C&W** [22]. On the original dataset, an AUC of 0.94 was achieved by the classifier on the normal data. The results on the attacks showed that they were effective at decreasing AUC, with C&W being the least effective attack, with an AUC of 0.80, and targeted FGSM being the most effective one, with an AUC of 0.44. JSMA was again identified as the most realistic attack, since a small range of features was modified, producing an AUC of approximately 0.5 for all classes. The authors also compared different attacks on the modified features, having found that there are mainly seven features used by almost all attacks, also discussing how an attacker can manually modify each one to perform adversarial attacks on a real scenario.

Zilong Lin et al. proposed IDSGAN [18], a Generative Adversarial Network (GAN) application to generate indiscriminate adversaries on the NSL-KDD dataset [51]. By training a Generator to produce adversarial malicious traffic from real malicious traffic, and a discriminator that distinguished normal traffic from adversarial, following the Wasserstein GAN architecture [28], the authors were able to generate adversarial network traffic. The effectiveness of the attacks was tested against a multitude of classifiers, namely Support Vector Machine, Naive Bayes, Multi-layer Perceptron, Logistic Regression, Decision Tree, Random Forest and K-nearest neighbors. By applying this concept to the NSL-KDD dataset, the results showed that the detection rates of adversarials on all classifier dropped from over 70% on the original data to less than 1% for all classifiers and types of attacks. To make the attacks more realistic, the authors prevented the IDSGAN from modifying functional features of the attacks, which are features that would affect the performance of the intrusions, such as time-based features for Denial of Service attacks, or content based features for User-to-Root attacks. However, they observe no notorious difference from the performance with the inclusion of functional features, proving thus the effectiveness of IDSGAN at creating adversarial examples, although there were no limitations to the amount of noise introduced by the technique.

The authors concluded that IDSGAN is effective in generating adversarial attacks, even when limiting the number of perturbed features to realistically and functionally modifiable ones, by reducing the detection rate to close to 0%. For future work, it was proposed to test the technique on more types of attacks.

Qiao Yan et al. followed a similar approach to Zilong Lin et al. [18] and applied Wasserstein GANs to the NSL-KDD dataset, exploring exclusively the generation of adversarial examples for Denial of Service samples [12]. To decide what features are realistically modifiable, the authors split them into four categories: *Basic features* refer to most elementary properties of the connection; *Content features* refer to the content of the connection, and are not important for DoS attacks; *Traffic features over a two second window* refer to statistical features over all the connections performed in the last two seconds; *Traffic features over 100 connections window* are statistics for the 100 previous connections, and are also considered not important to DoS attacks. Therefore, the authors only allow the modification of Content features and Traffic features of 100 connections windows. To prevent the modifications of these features, a Converter was added between the Generator and the Discriminator, which replaces the unmodifiable features with their original values.

After training the WGAN by generating adversarial DoS samples by approximating them to the distribution of normal traffic samples, the authors were able to decrease the True Positive Rate of a Convolutional Neural Network classifier from 97.3% to 47.6%, showing the effectiveness of the attack while maintaining the integrity of the Denial of Service attack by not modifying its functional features.

It is concluded that DoS-WGAN is effective at generating adversarial examples against a Convolutional Neural Network based IDS. Information entropy was calculated for all scenarios and its stability through the training confirms that WGAN is more stable than other models. The diversity of attacks was also observed to be greater when using WGAN.

To further extend the previous works to a black-box setting, Yang et al., using the NSL-KDD dataset, attacked the classifiers with three different black-box techniques: C&W, ZOO (Zeroth-order Optimization) and GAN [24]. For the first scenario, a substitute classifier was trained and used to generate adversarial samples using C&W to attack a separately trained model. On the second scenario, to attack the target classifier without training a substitute one, ZOO was used by querying the classifier to estimate the gradient and produce adversarial attacks. For the third scenario, a Wasserstein GAN (WGAN) was trained to generate adversarial samples. The results achieved were the following: with the C&W attack, a deterioration of 24% in the F1 score was achieved, from 0.898 to 0.687; with the ZOO attack, the deterioration was more notorious, with 70% on the F1 score, from 0.898 to 0.273; with the WGAN attack, the F1 score reduced by 62%, from 0.989 to 0.350. The authors concluded that, although ZOO was the most effective technique, it was also the least realistic, as it required an excessive amount of queries to generate adversarial examples, which would be detected in a real scenario. They also concluded that the GAN technique was very powerful, although the training procedure was still unstable, even when using Wasserstein GAN, and suffered from model collapse and convergence failure.

Their work showed the practicality of crafting adversarial attacks using more varied techniques against deep neural networks, with all attacks being effective at deteriorating the performance of the classifiers. It is also noted that the attacks are effective when the attackers have no access to the classifier used by the IDS (black-box attack), but, for greater effectiveness, knowledge of the parameters of classifier can be advantageous.

Joseph Clements et al. attacked Kitsune [10], a Network IDS system recently proposed by Yisroel Mirsky et al. [19]. The IDS is based on an ensemble of autoencoders, which receives a feature vector of preprocessed packets. The ensemble was trained exclusively on regular data, and the autoencoders made to rebuild regular instances of data, using root-mean-squared-error (RMSE) as an objective function. The IDS then performed classification during deployment based on the RMSE of input samples, triggering an alarm when it is above a threshold value. The authors attacked Kitsune on the Mirai botnet dataset [27] by using four adversarial attack techniques: FGSM, JSMA, C&W and ENM (Elastic Net Method [31]). When performing integrity attacks, all techniques achieved 100% False Negative Rate (FNR). Under availability attacks, only C&W and ENM achieved 100% FNR, with FGSM achieving 4% and JSMA 0%. It was concluded that

the attacks were effective against Kitsune, but that the adversary required knowledge of the data representation to perform the attacks. The authors also predict that, as feature representations in NIDS become more automated and less human knowledge will be necessary in the future, adversarial attacks will become a larger concern.

Biggio et al. proposed a technique to generate adversarial examples and test it on malware classifiers for PDF files [29] using the Contagio dataset [3]. A black-box scenario is assumed, where the attackers only have knowledge of the distribution of the training data and the feature space used by the target. The method proposed consists of optimizing the following formula:

$$x^* = \min_x g(x)$$

$$\text{subject to } d(x, x^0) < d_{max}$$

Here $g(x)$ is the loss of the replicated classifier, x is the original sample, x^0 is the desired adversarial sample and x^* is the generated sample. Since this is a non-trivial problem the authors propose using optimization methods to solve it similiarly to L-BFGS attack, but using gradient descent instead. A problem with using gradient descent in this scenario is that it might lead to local minimums, or unsupported regions since the distribution of the data known by the attacker might not be exact. As such, the authors modified the formula, by minimizing $\hat{g}(x)$ while maintaining the generated samples in highly populated regions (of regular samples). The authors then applied this technique to generate adversarial malware on PDF files, by using a feature representation proposed by Maiorca et al. [63] which consists of the tally of occurences of keywords. The results showed that False Negative rates of up to 100% were achieved for both Neural Network and SVM classifiers.

The authors believed that these results could be extended to non-differentiable classifiers, such as k-nearest-neighbors and decision trees, and also proposed the usage of other techniques such as bagging or random subspace method.

Weiwei Hu and Ying Tan generated adversarial malware examples in a black-box oracle scenario using a new system based on GANs called MalGAN [36] on a malware dataset [5]. The system consisted of a generator, a substitute detector (discriminator) and a black box detector. The generator received real malware samples, along with a noise vector which works as a seed for the output (an adversarial sample). These adversarial samples, along with benign samples, were then input to the substitute model, which predicted if they were malware or not. The main difference between this system and other GANs is that the ground truth labels are not used to train the system, but, instead, the output of the black box model (the target model) is used, since the objective is to deceive that classifier. The authors attacked six classifiers from different architectures: Random Forest, Linear Regression, Decision Tree, Support Vector Machine, Multi-layer perceptron and a Voting Ensemble of the previous classifiers. On normal data, all classifiers achieved True Positive Rates of over 90%, with Decision Tree showing the best performance of 98%. After generating

adversaries, the true positive rate dropped to below 0.20% for all classifiers, both for the training and testing sets.

Grosse et al. attacked a binary neural network classifier trained on the Drebin dataset, an Android malware dataset [34]. Since adding perturbations to dynamically gathered features is hard, the authors took the simpler approach of only considering static features. The authors used a feature vector exclusively with binary features which represented mainly system calls, and applied an attack strategy similar to JSMA, but instead only allowed the perturbations to cause extreme values (0 or 1) since the features were binary. After testing the effectiveness of the attack, the authors reached misclassification rates between 63% and 69% based on the proportion of malware samples on the training set, having lower misclassification rates when the malware ratio was higher.

Two defensive techniques were also tested: distillation and adversarial training. When applying distillation, a drop of up to 2% in the accuracy was observed on the normal data, but the misclassification rate dropped drastically up to 38% when under attack. Adversarial training was also proven effective, but largely depended on the proportion of adversarial samples used to train the classifiers.

Hyrum S. Anderson et al. took a different approach at attacking a malware dataset by using deep Q-learning, a reinforcement learning technique [26]. The environment was defined as a feature vector comprised of 2350 categorical features on the metadata of the samples, and the action space was defined as a set of valid modifications to the samples. Rewards were provided when a sample successfully misclassified the target model. The authors attacked a gradient boosted decision tree, and achieved an evasion rate of 16%, which is a very modest result, but serves as a proof of concept, and the results give insights on which features were used to attack the model.

It is presented in table 3.1 an overview of the different works explored in this section.

In the case of intrusion detection, several adversarial attack techniques have been explored against multiple classifiers from different families and structures, all showing effectiveness at deteriorating the performance, with DeepFool, WGAN and Zoo being the most effective, but JSMA the most realistic, since it changes less features. All basic classifiers performed approximately at the same level, with the exception of Decision Trees and Naive Bayes that generally performed the worst. Although many attacking strategies were explored, defenses were only explored in one study (adversarial training and denoising autoencoder), which proved their effectiveness at improving the resilience of the detection system. The variety of datasets is also scarce, with almost all works using exclusively the NSL-KDD dataset.

On malware detection scenarios, there is a greater variety of datasets, although not all are publicly accessible. There were also a great variety of explored attack strategies, with similar results to intrusion scenarios, although in malware scenarios there is more freedom on what features can be modified, making JSMA no more the only realistic attack. Adversarial defenses were also only tested in one study (distillation and adversarial training), which, again, were proven effective with little performance drops on regular data.

Literature Review

Table 3.1: Summary of works on adversarial machine learning applied to intrusion and malware scenarios.

Paper	Attack Techniques	Classifiers	Datasets	Metrics	Results & Conclusions
[39]	JSMA	Decision Tree Random Forest Linear SVM Voting Ensemble of 3 previous techniques Neural Network (MLP)	NSL-KDD	Accuracy F1 AUC	- random forest most resilient (10% accuracy drop) - SVM most affected classifier (27% accuracy drop)
[22]	FGSM JSMA Deepfool C&W	Neural Network	NSL-KDD	Accuracy F1 AUC	- C&W least effective attack (0.8 AUC) - FGSM most effective attack (0.44 AUC)
[18]	WGAN	Neural Network	NSL-KDD	Detection Rate	- detection rates dropped from over 70% to less than 1% for all classifiers - limitation on the range of modifiable features did not improve the performance of the classifiers
[12]	WGAN	CNN	NSL-KDD	True Positive Rate	- TPR reduced from 97.3% to 47.6%
[24]	C&W ZOO GAN	Naive Bayes Random Forest SVM Neural Network	NSL-KDD	F1	- ZOO was the most effective attack with a drop of 70% on F1 - C&W was the least effective attack with a drop of 24% on F1. - JSMA would be the most practical in real scenarios, since it modifies less features, and ZOO the least realistic, since it requires too many queries
[10]	FGSM JSMA C&W ENM	Kitsune NIDS (ensemble of autoencoders)	Mirai botnet	False Negative Rate	- all attacks were achieved 100% FNR under Integrity attack scenario - under availability attacks, FGSM and JSMA only achieved 0% and 4% FNR, with C&W and ENM achieving 100%
[29]	<i>Gradient Based Technique</i>	SVM Neural Network	Contagio	False Negative Rate	- False negative rates went up to 100% for Neural Networks and SVM classifiers - The attacks are effective in a black-box scenario
[36]	GAN	Neural Network	malwr	True Positive Rate	- All classifiers performed with over 90% TPR, with Decision Tree being the best - TPRs dropped below 0.20% for all classifiers
[34]	Modified JSMA	Neural Network	DREBIN	Misclassification rate	- misclassification rate between 63% and 69% when under attack - distillation deteriorates the accuracy by 2% but improves MR to 36% - adversarial training effectiveness largely depends on the proportion of adversarial samples used
[26]	Deep Q-Learning	Gradient Boosted DT	<i>unspecified</i>	False Negative Rate	- evasion rate of 16% was achieved for black box attack

In this section we explored several works which apply Adversarial Machine Learning concepts to Intrusion detection scenarios, and concluded that:

- Adversarial attacks can deteriorate the performance of intrusion and malware classifiers, even if they follow different architectures or are from different families due to the transferability of adversarial attacks [35];
- Various adversarial attack strategies have been explored, with some strategies being more effective than others. FGSM, JSMA and C&W were generally the least effective attacks,

Literature Review

while Deepfool and ZOO were typically the most destructive, although it's hard to find a pattern due to the variety in results achieved in across different studies;

- All classifiers achieved relatively similar results on normal data, but under adversarial attacks, Decision Trees, Naive Bayes and Linear SVMs were the most affected classifiers, while Neural Networks with a different architecture from the attacker, Random Forests and Radial basis function SVMs were the most resilient;
- Adversarial defenses were never explored for intrusion detection scenarios;
- The number of datasets that were used from intrusion detection scenarios was small, with NSL-KDD being mostly used. This is mainly due to the small amount of public labeled intrusion datasets.

To extend the existing literature in this dissertation, we decided to:

- Use multiple classifiers, with at least one for each tribe [\[13\]](#);
- Test multiple attack strategies, with at least one for each distance metric;
- Use a new DDoS dataset that has never been used in literature to the best of the authors knowledge (CICIDS 2017);
- Test the effectiveness of four adversarial defense techniques against multiple attack strategies.

Literature Review

Chapter 4

Proposed Architecture

To answer the questions proposed in section 1.2, an experimental setup was prepared (figure 4.1). The main steps consisted of:

1. Choosing and preparing the Denial of Service datasets;
2. Choosing the classifiers that will perform classification on the labeled datasets;
3. Implementing multiple adversarial attack strategies;
4. Implementing multiple adversarial defense techniques.

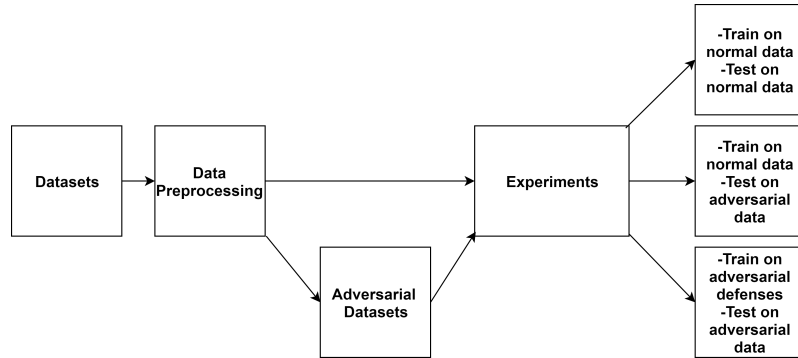


Figure 4.1: Pipeline of the experiments.

4.1 Datasets Description

The first step on the experimental setup was to find Denial of Service datasets. After performing a search on multiple science databases (arXiv, IEEE and ScienceDirect) and on Google for the keywords "*intrusion dataset*" and "*denial of service dataset*", we found nine potential datasets [11], [21].

To perform the experiments, the dataset needed to contain: DoS records; labeled data, to allow the distinction between normal and malicious data, and pre-processed features, to allow the models to operate on higher level data and facilitate the classifications. Among all the datasets that were explored, four met all the requirements: KDD-99 [45], NSL-KDD [51], KYOTO 2006 [66] and CICIDS 2017 [21].

KDD-99 is a dataset derived from the DARPA intrusion dataset, with features extracted. This is the most popular dataset for intrusion detection, being used in over 118 studies [45] and containing over 20 types of intrusions, including DoS attacks.

NSL-KDD was derived from the KDD-99 dataset to facilitate its integration with machine learning algorithms, by removing duplicate records and reducing the size of the dataset [51].

KYOTO 2006+ is a publicly available dataset from Kyoto University’s Honeypots containing multiple types of intrusions, as well as labeled attackers, with over 93 million sessions available.

CICIDS 2017 was generated within an emulated environment over a period of five days in the Canadian Institute for Cybersecurity, containing records of multiple types of intrusions, with DDoS records included, and over 80 features.

To perform our experiments, we decided to use NSL-KDD and CICIDS 2017. We chose to discard KDD-99 due to NSL-KDD being an improved and more used version of it, and we didn’t use KYOTO 2006+ because it contained unbalanced data and the different types of attacks were not individually labeled. In these datasets, each sample represents a connection. The list of features of both datasets are present in tables A.1 and A.2 (appendix A).

4.2 Datasets Preprocessing

Before training the classifiers on the datasets, several preprocessing operations had to be performed.

4.2.1 Samples Selection

The first step was to choose which samples were used from both datasets. The NSL-KDD dataset contained not only samples from DDoS attacks, but also from other types of attacks, so all non-DDoS samples were removed. There were also multiple types of DDoS attacks present, but to simplify the experiments, only two types of labels were considered: normal or malicious.

The NSL-KDD dataset was initially already split into a training and testing set, but to achieve more consistent results, we joined 20% of the training set to the full testing set. In total, the resulting dataset contained 47724 records, of which 34% were malicious (16685 records).

The CICIDS 2017 dataset required no sample selection, already containing only DDoS records with binary labels: normal or malicious.

In total, the CICIDS dataset contains 45146 records, of which 57% were malicious (25655 records).

4.2.2 One-Hot Encoding

The second step to prepare the datasets was to **One-Hot Encode** the categorical features of the datasets. Categorical features are ones that do not contain continuous values, and instead contain specific and pre-determined types of values. One example of a categorical feature is the type of protocol, which can be TCP, UDP or ICMP on the NSL-KDD dataset.

One-Hot encoding consists of creating a different boolean feature for each possible value of the categorical feature, assigning a 1 to the correct feature and 0 to all others. This allows categorical data to be used for classification on most classifiers.

After One-Hot-Encoding the NSL-KDD dataset, 122 features were obtained out of the initial 41. The CICIDS 2017 dataset was not one-hot encoded because it contained no categorical features aside from the port number, which was not one-hot encoded since it would generate thousands of additional features that would hinder the training process of the multiple classifiers.

4.2.3 Min-Max Normalization

The third step was to apply **Min-Max normalization** to all features. This consists of squeezing their values to a certain interval, which was in this case between 0 and 1. This prevents classifiers that use distance as a metric for classification from giving more importance to features with a larger interval of values, and is also a required step to generate adversarial examples, since all the attacking strategies require all features to be within a certain interval.

4.2.4 Feature Selection

The last step was to perform **Feature Selection** to remove misleading data and improve training times. Among many tested approaches, **Recursive Feature Elimination with Linear SVMs** was the technique that offered the highest mean AUC for all classifiers on the original dataset. The optimal number of features for the NSL-KDD dataset was 41 from 122, while for the CICIDS2017 it was 26 from 77. The score of the SVM classifier over the number of features selected can be seen in figure B.1, and the list of selected features for both datasets are present in table B.1 (appendix B).

4.3 Machine Learning Algorithms

To perform classification, six algorithms were used: **Decision Tree**, **Random Forest**, **Naive Bayes**, **Support Vector Machine**, **Neural Network** and **Denoising Autoencoder**.

To perform classification with the Denoising Autoencoder, new layers were added on top of the last layer of the autoencoder, which were trained after the autoencoder was trained, with the original layers frozen.

The choice of algorithms was performed in order to have at least one for each tribe of algorithms (according to Pedro Domingos[13]), with the exception of evolutionary algorithms, which

are not easy to implement for classification tasks. To optimize the hyper-parameters of the algorithms, we performed Cross Validation Grid Search, using 3 folds mainly to optimize performance. The algorithms and hyper-parameters that were optimized can be found in table 4.1.

Table 4.1: Optimized parameters for each algorithm.

Algorithm	Parameters
Decision Tree	Max depth, Criterion
Random Forest	Max depth, Criterion, Number of estimators
Support Vector Machine	Cost, Gamma
Naive Bayes	Laplace Smoothing
Neural Network	Neurons per layer, Dropout rate
Denoising Autoencoder	Neurons per layer, Gaussian Noise Std. Dev.

4.4 Adversarial Example Generation

To generate the adversarial examples, four adversarial attack techniques were used: **FGSM**, **JSMA**, **Deepfool** and **C&W**. The choice of adversarial attack algorithms was done to have at least one attack for each distance metric (JSMA for L_0 , Deepfool and C&W for L_2 and FGSM for L_∞).

C&W was included to test its effectiveness against **Adversarial Training** and **Defensive Distillation**, since it was proven by the authors that this attack was able to beat both defenses [46].

We performed multiple tests for the attack algorithms, and tweaked an hyperparameter controlling the magnitude of the attack in three of the attacks (table 4.2). ϵ and ϑ control the magnitude of the perturbations in FGSM and JSMA, and Overshoot (os) is a termination criterion.

Table 4.2: Tweaked parameters and distance metrics for each attack.

Algorithm	Hyperparameters	Distance Metric
FGSM	ϵ (0.1, 0.2 and 0.3)	L_∞
JSMA	ϑ (0.1, 0.2 and 0.3)	L_0
Deepfool	Overshoot (0.2, 0.5 and 1)	L_2
C&W	<i>None</i>	L_2

When generating adversarial datasets, only DDoS labeled connections were perturbed, since the aim of an attacker is to modify its malicious connections in a way to make them undetectable by the classifiers, making it an **Integrity** attack.

We also didn't allow the attacks to modify one-hot-encoded features, since these attacks could potentially assign values between 0 and 1 to these columns, or even assign positive values to more than one column at the same time, which is impossible.

4.5 Adversarial Defenses

To improve the detection of adversarial examples, we selected four adversarial defense strategies: **Adversarial Training**, **Separate Binary Classifier**, **Defensive Distillation** and **MagNet**. We exclusively selected adversarial defense strategies that were not data dependant, being applicable to any type of data outside of images. We also tried to use **Transferability Block**, but couldn't achieve meaningful results due to difficulties in the implementation, so we didn't include these results.

To implement **Adversarial Training**, we included adversarial examples in the training set from every algorithm different from the one being tested, with the maximum perturbations. For example, when testing adversarial training against FGSM with $\epsilon = 0.1$, we trained the classifier on the original datasets, JSMA with $\vartheta = 0.3$, Deepfool with overshoot = 1 and C&W. This was done to simulate a scenario where an attacker uses an attack for which the classifiers were not specifically trained against. In this scenario, we again used the same classifiers that were used to perform classification on the original datasets (DT, RF, SVM, NB, NN and DAE).

The **Separate Binary Classifier** was implemented by initially training a classifier on the original data, and then training a separate model on the original and adversarial datasets, using two labels: 'normal' or 'adversarial'. In this case, malicious samples from the original dataset are "normal", since the objective of this model is only to distinguish normal and adversarial samples, and not benign and malicious ones. We used the same strategy as in adversarial training, by training the classifiers using adversarial datasets from attacks different from the ones being tested. We tested the effectiveness of the defense both using NN and RF as both classifiers.

The **Defensive Distillation** technique was implemented by training a neural network initially on the normal dataset, and then performing predictions on the same dataset, with a higher temperature to increase the class distribution. We then used these predictions to train a new classifier following the same structure, and tested this classifier on the adversarial datasets. We tested multiple values of temperature, by starting at 10 and finishing at 150, with increments of 10, and arrived at an optimal temperature of 120.

Finally, the **MagNet** defense was implemented by initially training a classifier on the original datasets. We then trained an autoencoder on these datasets, and obtained the average and standard deviation of the loss of all samples during training. During deployment, we rejected every sample whose loss on the autoencoder was above a threshold. In our implementation, we took a different approach when calculating this threshold, by instead using the following formula:

$$threshold = mean(losses) + N * stddev(losses)$$

Here, *losses* refers to all the losses during training phase, *mean(losses)* is the average of all the losses during training, *stddev(losses)* is the standard deviation of all the losses and *N* is a hyperparameter controlling the margin of error accepted. We tested multiple values of *N* starting at 1 and ending at 5, and arrived at an optimal value of 4. After rejecting adversarial samples

above the threshold, we pass the remaining samples through the same autoencoder, and perform classification on the outputs of the autoencoder using the classifier that was initially trained. We tested two types of classifiers for this step: RF and NN.

4.6 Pipeline

In total, three different experiments were performed to answer the questions proposed in chapter 1.

Scenario 1 - Original/Original: on the first experiment, all classifiers were trained on the original datasets, and tested on that same datasets. This serves as the baseline performance of the classifiers on the original datasets. We performed in total 6 tests on both datasets, by doing 10-fold cross validation with the 6 classifiers;

Scenario 2 - Original/Adversarial: on the second experiment, all classifiers were trained on the original datasets, and tested on four adversarial datasets. This experiment validates if the generation of adversaries can deteriorate the performance of classifiers that were not trained against them, and allows an analysis on the behaviour of the different classifiers. We performed in total 60 tests on each dataset, by doing 10-fold cross validation with the 6 classifiers against 10 adversarial datasets (the original datasets were used in the train folds, while the adversarial datasets were used in the test folds);

Scenario 3 - Adversarial Defense/Adversarial: on the third experiment, four adversarial defense strategies were implemented to validate their performance at detecting adversarial examples. In this experiment we performed 110 tests (60 for Adversarial Training, 20 for Separate Binary Classifier, 20 for MagNet and 10 for Defensive Distillation). We again performed 10-fold cross validation, with the training folds and classifiers used depending on the defensive strategy used, and the test fold belonging to the adversarial datasets.

4.7 Technologies

All experiments were performed using **Python 3.7**. To facilitate the implementations of several techniques, some machine learning libraries were used. To implement the Connectionist classifiers, we used **Keras** with the **Tensorflow** backend [1], [43]. All other classifiers were implemented using the **scikit-learn** library [65]. To generate adversarial attacks, the **Cleverhans** library was used, which offers implementations for all the techniques used [20]. The adversarial defense techniques were all manually implemented on top of **Keras** and **scikit-learn**.

Chapter 5

Experimental Results

In this chapter we explore the results of the experiments proposed in the previous chapter. We first test the performance of the classifiers on the original data to serve as a baseline (scenario 1), and then test the same classifiers against adversarial attacks on the Denial of Service records (scenario 2). Finally, we test the effectiveness of adversarial defense strategies (scenario 3).

To evaluate the performance of the classifiers, the ROC-AUC and F1-score metrics were used, since they are independent on the proportions of classes present in the datasets. On the tables showing the results of the experiments, the yellow cells indicate the best performing model in a certain dataset, while the green cells indicate the classifier that performed the best, on average, across all datasets in that experiment.

5.1 Original Data Results

The first experiment serves as a baseline for the performance of the classifiers. We performed 6 tests, one for each classifier, using 10-fold cross validation on each test. The results can be seen in tables 5.1 and 5.2.

Table 5.1: Baseline results on the NSL-KDD dataset (ROC-AUC: left, F1-score: right).

AUC	Original	Ranks	F1	Original	Ranks
dtree	0.9911	1	dtree	0.9893	1
rforest	0.9884	2	rforest	0.9876	2
svm	0.9702	4	svm	0.9678	4
naive	0.9038	6	naive	0.8901	6
nn	0.9783	3	nn	0.9757	3
dae	0.9053	5	dae	0.8906	5

In the baseline experiments, we can see that the performance of the classifiers is very similar across both datasets, and both AUC and F1-score follow the same pattern. In both experiments, the Decision Tree was the best performing algorithm, with the Random Forest achieving very similar results. The classifiers that achieved the worst results were Naive Bayes and Denoising Autoencoder. It is expected that the performance of the Denoising Autoencoder would be one of

Experimental Results

Table 5.2: Baseline results on the CICIDS 2017 dataset (ROC-AUC: left, F1-score: right).

AUC	Original	Ranks	F1	Original	Ranks
dtree	0.9993	1	dtree	0.9994	1
rforest	0.9993	2	rforest	0.9994	2
svm	0.9931	4	svm	0.9946	4
naive	0.8889	6	naive	0.9221	6
nn	0.9921	3	nn	0.9932	3
dae	0.8762	5	dae	0.9139	5

the lowest, since it performs a denoising step before classification which might cause corruption of outlier samples and lead to their misclassification.

We used Cross Validation Grid Search on 3 folds on the entire datasets to optimize the parameters of the classifiers, and the optimal ones that were found for each dataset can be seen in tables C.1 and C.2 (appendix C).

5.2 Adversarial Attacks

The aim of the second experiment was to test the effectiveness of adversarial attacks on Denial of Service datasets, allowing malicious connections to be classified as normal, and to analyze the behaviour of the different types of classifiers. To achieve this, we applied four adversarial attack algorithms: **Fast Gradient Sign Method**, **Jacobian Based Saliency Map Attack**, **Deepfool** and **Carlini & Wagner attack**. After ten adversarial datasets were generated using these attacks, we evaluated the performance of the classifiers trained in experiment 1, by performing 10-fold cross validation, training the classifiers on the same data as in experiment 1, but testing on the adversarial datasets.

The parameters used for the attacks are in table 5.3.

Table 5.3: Parameters used by the attacks (10 attacks in total).

Classifier	Parameters		
FGSM	ϵ : 0.1; 0.2; 0.3		
JSMA	ϑ : 0.1; 0.2; 0.3	gamma: 1	target: Benign Sample
Deepfool	overshoot: 0.2; 0.5; 1	max iterations: 3	
C&W	confidence: 0.5	learning rate: 0.2	max iterations: 100

5.2.1 Approaches Comparison

These attacks were exclusively applied to denial of service records, since the aim is to simulate an attacker that is trying to bypass the classifiers by altering their malicious connections. Because this scenario is a binary classification problem, both **Targeted** and **Untargeted** attacks produce the same results. Multiple perturbation settings were used for three of the attack strategies, to

Experimental Results

evaluate how the magnitude of the attack influences the performance of the classifiers. The results are in tables 5.4-5.7.

Table 5.4: Adversarial attacks on the NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.7286	0.7650	0.8783	0.8695	0.9241	0.8781	0.5920	0.7176	0.7306	0.7068	5.2
rforest	0.6870	0.7741	0.8849	0.9526	0.9548	0.9476	0.5488	0.5779	0.6820	0.6200	4.5
svm	0.8082	0.8700	0.9336	0.9000	0.9349	0.9669	0.6420	0.6723	0.7666	0.9456	2.9
naive	0.7888	0.8914	0.9016	0.9057	0.9056	0.9044	0.7612	0.8231	0.8996	0.9014	2.9
nn	0.7681	0.8474	0.9244	0.9054	0.9372	0.9637	0.7034	0.7239	0.7728	0.8671	3
dae	0.8953	0.9027	0.9058	0.9010	0.9012	0.9072	0.8846	0.9027	0.9047	0.8922	2.5

Table 5.5: Adversarial attacks on the NSL-KDD dataset (F1-score).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.6303	0.6944	0.8608	0.8185	0.9134	0.8159	0.3177	0.6078	0.6331	0.5815	5.2
rforest	0.5396	0.7087	0.8697	0.9497	0.9521	0.9442	0.1831	0.2699	0.5355	0.3895	4.5
svm	0.7627	0.8504	0.9277	0.8882	0.9292	0.9642	0.4492	0.5171	0.6975	0.9412	2.8
naive	0.7395	0.8756	0.8875	0.8922	0.8922	0.8908	0.6971	0.7884	0.8852	0.8873	3
nn	0.6987	0.8197	0.9173	0.8940	0.9322	0.9607	0.5708	0.6203	0.7046	0.8344	2.9
dae	0.8794	0.8884	0.8919	0.8865	0.8873	0.8934	0.8663	0.8880	0.8909	0.8755	2.6

Table 5.6: Adversarial attacks on the CICIDS 2017 dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.5030	0.5006	0.5018	0.5648	0.5646	0.5642	0.5001	0.5018	0.5039	0.5030	3.5
rforest	0.5006	0.4996	0.4996	0.6355	0.6292	0.6550	0.4997	0.5000	0.5032	0.5010	4.2
svm	0.4998	0.4998	0.5000	0.4990	0.5025	0.5015	0.5031	0.5040	0.5056	0.5529	3.8
naive	0.3891	0.3891	0.3891	0.3898	0.3898	0.3898	0.4106	0.4201	0.4303	0.5587	5.7
nn	0.5004	0.4999	0.5265	0.6043	0.6520	0.7926	0.5000	0.5024	0.5050	0.9349	2.7
dae	0.8155	0.8602	0.8284	0.8887	0.8976	0.8766	0.8692	0.8491	0.8840	0.9147	1.1

Table 5.7: Adversarial attacks on the CICIDS 2017 dataset (F1-score).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.0139	0.0047	0.0093	0.2309	0.2304	0.2293	0.0026	0.0094	0.0178	0.0141	4.4
rforest	0.0040	0.0000	0.0001	0.4121	0.3942	0.4574	0.0000	0.0026	0.0141	0.0054	4.75
svm	0.0252	0.0252	0.0260	0.0215	0.0335	0.0307	0.0380	0.0413	0.0473	0.2088	3.7
naive	0.0000	0.0000	0.0000	0.0023	0.0023	0.0023	0.0709	0.1008	0.1318	0.4499	4.45
nn	0.0253	0.0253	0.1168	0.3595	0.4627	0.7422	0.0223	0.0400	0.0383	0.9234	2.7
dae	0.8598	0.8851	0.8878	0.9376	0.8923	0.9098	0.9104	0.8994	0.9201	0.9397	1

Experimental Results

On this second experiment, with the introduction of adversaries in the testing phase, we can observe a generalized decrease of the performance of all classifiers. The classifier’s parameters used in this experiment were the same as in the first experiment.

On the NSL-KDD dataset, the average AUC of all classifiers decreased by **13%**, from **0.956** to **0.833**, and the average F1-score decreased by **19%**, from **0.9505** to **0.7730**. Meanwhile, on the CICIDS 2017 dataset, the average AUC decreased by **40%**, from **0.958** to **0.576**, and the average F1-score decreased by **78%**, from **0.9710** to **0.2132**. The higher decrease in performance on the CICIDS dataset can mainly be attributed to the higher proportion of features that were vulnerable to noise, since, on the NSL-KDD dataset, noise was only added to 10 out of its 41 features, while on the CICIDS dataset, it was added to 25 out of 26 features. On this experiment, the classifier that offered most resilience to adversaries on both datasets was the Denoising Autoencoder, whose mean AUC and F1-score only decreased by 1% on the NSL-KDD dataset, and 1% on the CICIDS dataset, showing very little variations due to its denoising capabilities.

Using the Friedman test with a critical difference of **2.38** for a 95% confidence level, we can observe that the performance of the Denoising Autoencoder is significantly better than the Decision Tree on the NSL-KDD dataset, and significantly better than all classifiers except the Neural Network on the CICIDS dataset. The higher performance of the Denoising Autoencoder over other classifiers can be due to its ability to remove noise from the adversarial data, since it was trained by introducing additive gaussian noise on the training data. Although on average the Denoising Autoencoder had the highest rank in this experiment, we can observe that the Support Vector Machine and Random Forest performed generally well in scenarios with less noise introduced, such as all JSMA attacks and FGSM with $\epsilon=0.1$ on the NSL-KDD dataset.

When analyzing the effectiveness of the attacks, the most effective one on the NSL-KDD dataset was Deepfool, by decreasing AUC by **23%** and the F1-score by **35%**, while the least effective was JSMA, only decreasing both metrics by **5%**. Meanwhile, on the CICIDS 2017 dataset, the most effective attack was FGSM, by decreasing the mean AUC by 46% and F1-score by 85%, and the least effective was C&W, by decreasing the mean AUC by 31% and F1-score by 55%. We can thus conclude that the effectiveness of the attacks largely depends on the dataset used, as well as the proportion of features to which the adversary has access.

5.2.2 Feature Perturbation Analysis

While implementing the different adversarial attack strategies on Denial of Service datasets, we decided to validate the integrity of the adversarial samples by verifying what were the labels of the closest samples, using euclidean distance, in the original dataset. We concluded that, for 99% of the adversarial examples, the closest sample in the original dataset was also malicious, confirming that the integrity of the attack was not substantially affected.

We also decided to analyze what features the attacks were targetting, on the average magnitude and frequency. We didn’t include the results on FGSM, since that attack targets every feature with the same magnitude, and we only included the attacks one hyperparameter setting, since the results were similar across different magnitudes.

Experimental Results

The results are in tables 5.8-5.10.

Table 5.8: Analysis on modified features by JSMA attack with $\vartheta=0.3$.

NSL-KDD (3.17 features modified per sample on average)				
Number of modified samples	duration	num_access_files	dst_host_srv_diff_host_rate	dst_host_srv_count
	9053	8851	4026	3959
CICIDS 2017 (2.02 features modified per sample on average)				
Number of modified samples	Fwd Packet Length Max	Fwd Packet Length Std	Fwd Packets/s	Fwd Packet Length Mean
	22971	14942	13297	1868

Table 5.9: Analysis on modified features by Deepfool attack with Overshoot=1.

NSL-KDD (13.17 features modified per sample on average)				
Number of modified samples	srv_count	count	dst_host_srv_diff_host_rate	duration
	9234	9234	9105	9099
Average Perturbation Magnitude	wrong_fragment	duration	num_access_files	dst_host_srv_diff_host_rate
	0.896	0.754	0.461	0.416
CICIDS 2017 (18.57 features modified per sample on average)				
Number of modified samples	Fwd Packets/s	Fwd Packet Length Std	Fwd Packet Length Max	Init_win_bytes_forward
	25046	25035	25027	24981
Average Perturbation Magnitude	Fwd Packets/s	Fwd Packet Length Std	Fwd Packet Length Max	Fwd Packet Length Mean
	0.1247	0.1237	0.1125	0.0942

Table 5.10: Analysis on modified features by C&W attack.

NSL-KDD (6 features modified per sample on average)				
Number of modified samples	count	dst_host_same_src_port_rate	srv_count	dst_host_diff_srv_rate
	15637	13684	13455	13436
Average Perturbation Magnitude	count	dst_host_same_src_port_rate	duration	srv_count
	0.242	0.086	0.057	0.049
CICIDS 2017 (14.52 features modified per sample on average)				
Number of modified samples	Avg Fwd Segment Size	Fwd Packet Length Max	Fwd Packet Length Mean	Init_win_bytes_forward
	23533	23367	23000	22753
Average Perturbation Magnitude	Fwd Packet Length Std	Fwd Packet Length Max	Fwd Packets/s	Avg Fwd Segment Size
	0.064	0.062	0.052	0.045

For each attack excluding FGSM, we show: the average number of features modified per malicious sample, the four most perturbed features in count, and the four most perturbed features in magnitude.

We can see that, in the NSL-KDD dataset, mostly time related features are targeted, such as duration of the connection (duration) and number of connections to the target host in the past two seconds (count), which are typically the features that distinguish a Denial of Service attack, due to the flooding of data in short periods of time. Meanwhile, in the CICIDS dataset, mostly features relating to the length of the packets are targeted, which can indicate that the length of the packets follows a certain pattern for the Denial of Service attacks used in this dataset, although the number of packets per second is also used, as in NSL-KDD.

Comparing the different attacks, we can see that JSMA distinctly perturbed a very small amount of features per sample on average compared to the other attacks. We can thus conclude

that JSMA is the most realistic attack, since, in a Denial of Service scenario, an attacker has small power to modify a larger amount of features.

We can also see that Deepfool tended to modify more features per sample than both attacks, showing a similar pattern to C&W, since both are L_2 attacks, although with different magnitudes due to hyperparameter settings. We can also see that, although they targeted nearly the same features, they gave different priorities to each according to the number of modified samples.

5.3 Adversarial Defenses

The adversarial attack strategies were proven effective on both datasets, with varying effectiveness based on the technique and hyperparameters used. The third and final experiment was thus performed to test the effectiveness of multiple adversarial defenses against these attacks. The techniques that were tested were: **Adversarial Training**, **Separate Binary Classifier Distillation** and **MagNet**.

5.3.1 Adversarial Training

The first defensive strategy that was proposed in adversarial machine learning literature was adversarial training. This strategy can follow two paths, either by including adversarial samples on the training dataset, or using an adversarial loss function on the classifier, which produces similar results.

Our implementation took the first approach of including adversarial samples in the training set to allow the inclusion of more than one type of adversary on the training set, and extended it by using other classifiers other than Neural Networks: Decision Tree, Random Forest, Support Vector Machines, Naive Bayes and Denoising Autoencoders (the same classifiers used in the second experiment). To defend against each adversarial attacks, we included samples from other attack types on the training set, to prevent the classifiers from being trained against the specific attack, creating more realistic scenarios.

The results obtained are in the tables 5.11-5.14.

Table 5.11: Adversarial training on the NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.9847	0.9865	0.9914	0.9798	0.9808	0.9818	0.9934	0.9931	0.9929	0.9942	2.4
rforest	0.9924	0.9931	0.9935	0.9740	0.9737	0.9781	0.9996	0.9995	0.9995	0.9944	1.9
svm	0.9912	0.9894	0.9882	0.9794	0.9797	0.9804	0.9093	0.9593	0.9736	0.9692	3.3
naive	0.7935	0.6376	0.6114	0.9171	0.6863	0.6114	0.6467	0.6378	0.6333	0.6114	5.9
nn	0.9860	0.9875	0.9893	0.9833	0.9851	0.9846	0.9890	0.9895	0.9902	0.9770	2.4
dae	0.8384	0.8966	0.8902	0.8982	0.8960	0.8988	0.8946	0.8979	0.9002	0.8954	5.1

By introducing adversaries in the training phase using different attack algorithms from the one being tested, the decrease in the average performance of all classifiers was significantly less

Experimental Results

Table 5.12: Adversarial training on the NSL-KDD dataset (F1-score).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.9829	0.9848	0.9898	0.9784	0.9794	0.9804	0.9911	0.9908	0.9906	0.9931	2.4
rforest	0.9921	0.9928	0.9932	0.9732	0.9728	0.9775	0.9995	0.9994	0.9993	0.9942	1.9
svm	0.9895	0.9877	0.9865	0.9775	0.9779	0.9786	0.8984	0.9562	0.9715	0.9667	3.3
naive	0.7048	0.4295	0.3717	0.9076	0.5099	0.3717	0.4572	0.4379	0.4272	0.3717	5.9
nn	0.9839	0.9852	0.9874	0.9815	0.9830	0.9831	0.9871	0.9877	0.9884	0.9755	2.4
dae	0.7912	0.8802	0.8729	0.8813	0.8793	0.8824	0.8782	0.8818	0.8853	0.8785	5.1

Table 5.13: Adversarial training on the CICIDS 2017 dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.8579	0.8868	0.9752	0.5649	0.5648	0.6171	0.9060	0.9679	0.9981	0.9916	3.8
rforest	0.9999	0.9999	0.9999	0.9992	0.9991	0.9910	0.9997	0.9996	0.9996	0.9923	1
svm	0.4981	0.6039	0.8192	0.9809	0.9858	0.9882	0.9883	0.9883	0.9883	0.9872	3.4
naive	0.3990	0.5684	0.6924	0.8435	0.8024	0.8024	0.8219	0.8204	0.8155	0.7950	5.4
nn	0.8134	0.9389	0.9919	0.8401	0.9270	0.9474	0.9951	0.9959	0.9955	0.9896	2.9
dae	0.8282	0.7959	0.7935	0.8535	0.7763	0.8463	0.7916	0.8613	0.8599	0.8271	4.5

Table 5.14: Adversarial training on the CICIDS 2017 dataset (F1-score).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
dtree	0.8069	0.8520	0.9746	0.2309	0.2304	0.3802	0.8946	0.9669	0.9982	0.9917	4
rforest	1.0000	0.9999	1.0000	0.9992	0.9991	0.9907	0.9997	0.9996	0.9996	0.9923	1.1
svm	0.0310	0.3498	0.7844	0.9836	0.9886	0.9910	0.9911	0.9912	0.9912	0.9903	3.5
naive	0.0981	0.5005	0.7016	0.8313	0.7741	0.7741	0.7969	0.7953	0.7897	0.7612	5.4
nn	0.7706	0.9339	0.9926	0.8028	0.9222	0.9452	0.9955	0.9965	0.9959	0.9899	3
dae	0.8877	0.8689	0.8680	0.9001	0.8584	0.8960	0.8652	0.9048	0.9040	0.8865	4

notorious, with a decrease of 4% in AUC (**0.956** to **0.917**) and 8% in F1 (**0.950** to **0.883**) on the NSL-KDD dataset, and a decrease of 10% in AUC (**0.958** to **0.870**) and 14% in F1 (**0.971** to **0.864**) on the CICIDS dataset.

In this defense strategy, the classifier that showed more resilience to adversaries was Random Forest, which, when compared to the best baseline classifier (decision tree), only suffered a decrease of 0.1% in AUC and decrease of 0% in F1 on the NSL-KDD dataset, and a decrease of 0.1% of AUC and F1 on the CICIDS dataset. Contrary to the second experiment, the Denoising Autoencoder suffered minimal changes in the mean AUC, with larger improvements being observed on the Random Forest and Neural Network Classifiers. Using the Friedman test, the Random Forest classifier is significantly better than the Denoising Autoencoder and Naive Bayes on the NSL-KDD dataset, and significantly better than all classifiers except the Neural Network on the CICIDS dataset.

Experimental Results

Conclusions: We can thus conclude that adversarial training is an effective defensive technique, with the random forest implementation being the most effective of the six on both datasets. The performance of Random Forest in this scenario can be attributed to it being an ensemble method, which reduces the error by averaging the outputs of all the independent trees, showing resilience when trained with different types adversarial examples. Although Random Forest performed the best on the CICIDS classifier for all attacks, we can observe that in the NSL-KDD dataset, where less features were perturbed, the classifier that performed the best against JSMA was the Neural Network, showing that Random Forest is specifically effective against attacks with larger number of features perturbed.

We can also see in the tables that, in some cases, the performance of the classifiers increased when comparing to the baseline experiment (scenario 1), showing that, when using adversarial training, some malicious samples that weren't initially detected were correctly identified.

5.3.2 Separate Binary Classifier

The second adversarial defense that was implemented was a Separate Adversarial Classifier, which consists of using two classifiers: one is trained to exclusively identify adversarial examples, and the other performs classification on non-adversarial data. Due to the results we obtained with Random Forest in adversarial training, we decided to expand the original implementation, which uses neural networks, by using random forest classifiers for both steps.

The results for the separate adversarial classifier are present in tables 5.15-5.18.

Table 5.15: Separate Binary Classifier on the NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
bin nn	0.9866	0.9909	0.9900	0.9815	0.9834	0.9839	0.9840	0.9858	0.9830	0.9566	2
bin rforest	0.9981	0.9973	0.9974	0.9837	0.9864	0.9877	0.9978	0.9974	0.9973	0.9981	1

Table 5.16: Separate Binary Classifier on the NSL-KDD dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
bin nn	0.9866	0.9909	0.9900	0.9815	0.9834	0.9839	0.9840	0.9858	0.9830	0.9566	2
bin rforest	0.9981	0.9973	0.9974	0.9837	0.9864	0.9877	0.9978	0.9974	0.9973	0.9981	1

Table 5.17: Separate Binary Classifier on the CICIDS 2017 dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
bin nn	0.9927	0.9948	0.9941	0.7708	0.8574	0.9154	0.9926	0.9923	0.9899	0.9835	1.8
bin rforest	0.9998	1.0000	1.0000	0.8279	0.7798	0.7495	1.0000	1.0000	0.9999	0.9998	1.2

Experimental Results

Table 5.18: Separate Binary Classifier on the CICIDS 2017 dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
bin nn	0.9944	0.9960	0.9955	0.9223	0.9278	0.9184	0.9944	0.9942	0.9914	0.9845	1.7
bin rforest	0.9999	1.0000	1.0000	0.7776	0.7028	0.6642	1.0000	1.0000	1.0000	0.9999	1.3

We observed that our implementation using Random Forest performed on average better than the original implementation using Neural Networks, with its ranks being better than neural network on most cases. When comparing this implementation to the best baseline classifier (decision tree), there was an improvement of 0.01% on AUC and 0.02% on F1 on the NSL-KDD dataset, and a decrease of 6% in AUC and 9% in F1 on the CICIDS dataset. We also observed that this technique reached near 1.0 AUC and F1 on the CICIDS dataset, but responded particularly badly against JSMA attacks, although this was not verified in the NSL-KDD dataset.

Conclusions: We concluded that this strategy followed the same pattern as adversarial training, since random forest implementation was the most effective one and showed improvements when compared to scenario 2. We can see again in the tables that, in some cases, the performance of the classifiers increased when compared to the baseline experiment (scenario 1), showing the effectiveness of including adversarial examples in the training set at detecting malicious samples.

5.3.3 Defensive Distillation

The next adversarial defense strategy that we tested was Defensive Distillation. This technique consists of initially training a classifier on normal data, and then outputting the predictions of the class distributions on a higher temperature, increasing the label distribution. This predictions are then used to train a second neural network with the same structure which will perform the final predictions. This technique was only used with neural networks, since it's not possible to increase the temperature of the output of other models, as they only output a final prediction and not class distributions.

The results for distillation with a temperature of 120 are present in tables 5.19-5.22.

Table 5.19: Distillation on the NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default
distillation	0.8544	0.8511	0.9326	0.9361	0.9343	0.9331	0.8455	0.8604	0.9311	0.9342

Table 5.20: Distillation on the NSL-KDD dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default
distillation	0.8265	0.8204	0.9216	0.9263	0.9248	0.9239	0.8156	0.8323	0.9194	0.9242

Comparing the average performance of distillation against the best baseline classifier (decision tree), we concluded that the decreases in performance were more notorious than the previous

Experimental Results

Table 5.21: Distillation on the CICIDS 2017 dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default
distillation	0.4866	0.4865	0.6953	0.6773	0.7495	0.9255	0.7670	0.8397	0.9435	0.9805

Table 5.22: Distillation on the CICIDS 2017 dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default
distillation	0.0138	0.0089	0.0042	0.5059	0.6453	0.9197	0.7019	0.8066	0.9448	0.9849

techniques, by showing a decrease of 9% in AUC and 10% in F1 on the NSL-KDD dataset, and a decrease of 25% in AUC and 45% in F1 on the CICIDS dataset. Some outlier performances can also be observed, where F1 nearly reached 0 against FGSM on the CICIDS dataset.

Conclusions: We can conclude that this defense strategy showed inferior results across all metrics when compared to Adversarial Training and Separate Binary Classifier, although it showed improvements in the detection of adversaries when compared to the classifiers used in scenario 2. It is important to note that this strategy doesn't require the inclusion of adversarial examples on the training set, although it requires the training of a new classifier. We also observed that this strategy was resilient to the C&W attack, contrary to the results obtained by Papernot et al. [46], where they were able to defeat defensive distillation in image recognition datasets.

5.3.4 MagNet

The fourth and final adversarial defense that we implemented was MagNet. This technique consists of using the loss of an autoencoder during the testing phase to find samples that significantly deviate from the original data distribution. Upon removing samples whose loss deviates significantly, the remaining samples are passed through an autoencoder, and then are input to a model trained to classify normal data. We also decided to test the effectiveness of Random Forest as the last classifier, due to the great performance observed in adversarial training. The results for MagNet are present in tables 5.23-5.26.

Table 5.23: MagNet on the NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
magnet nn	0.9220	0.8937	0.9202	0.9077	0.8950	0.9056	0.9024	0.9229	0.9038	0.9171	1.4
magnet rforest	0.9158	0.9171	0.9016	0.9574	0.9601	0.9616	0.7715	0.7938	0.8315	0.8714	1.6

On average, the original MagNet implementation with neural network offered better performance than the one with random forest, contrary to the adversarial training strategy. When comparing to the best baseline classifier (decision tree), we observed a decrease of 9% in AUC and 10% in F1, and a decrease of 18% in AUC and 21% in F1 on the CICIDS dataset.

Experimental Results

Table 5.24: MagNet on the NSL-KDD dataset (F1).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
magnet nn	0.9060	0.8769	0.9038	0.8901	0.8796	0.8891	0.8850	0.9078	0.8855	0.9010	1.5
magnet rforest	0.9067	0.9082	0.8894	0.9534	0.9567	0.9578	0.7049	0.7397	0.7968	0.8394	1.5

Table 5.25: MagNet on the CICIDS 2017 dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
magnet nn	0.9541	0.8995	0.6732	0.7842	0.7578	0.8415	0.7688	0.8022	0.8631	0.8373	1.3
magnet rforest	0.9628	0.9536	0.7335	0.5003	0.5000	0.5557	0.7070	0.6800	0.6479	0.5041	1.7

Table 5.26: MagNet on the CICIDS 2017 dataset (F1).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
magnet nn	0.9639	0.8971	0.5606	0.7557	0.7073	0.8430	0.7301	0.7759	0.8566	0.8239	1.2
magnet rforest	0.9594	0.9503	0.6281	0.0014	0.0003	0.2005	0.5857	0.5290	0.4544	0.0168	1.8

Conclusions: We concluded that MagNet also showed inferior results across all metrics when compared to Adversarial Training and Separate Binary Classifier, although it also showed improvements in the detection of adversaries when compared to the classifiers used in scenario 2. The inferior performance of Random Forest in this scenario can be due to the smaller variety and amount of data used to train the classifiers, not fully exploiting the ensembling of the Random Forest. This strategy can be easier to deploy than the previous ones, since it doesn't require the inclusion of adversarial examples on the training set, and it doesn't require the training of a new classifier on non-adversarial data.

5.3.5 Approaches Comparison

After individually analyzing the performance of adversarial defenses, we concluded that all offered an improvement relative to the baseline classifiers when exposed to adversarial examples. To compare the different defensive approaches, we decided to include the best ranking classifiers for each defense strategy, as well as the denoising autoencoder strategy without adversarial training, due to its superior effectiveness compared to the other classifiers when detecting adversarial examples in the second experiment. The results can be seen in tables 5.27-5.30.

Taking a look at the rankings, we can observe that the defensive strategies that revolved around including adversarial samples in the training set performed better than the other strategies. Using the **Friedman test**, with a critical difference of 1.928, we can conclude that **adversarial training with random forest** and **Separate Binary Classifier** were significantly better than all other strategies on the NSL-KDD and CICIDS datasets, with the exception of the **Denoising Autoencoder** on the CICIDS dataset.

Experimental Results

Table 5.27: Defense strategies comparison on NSL-KDD dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
advtrain rforest	0.9924	0.9931	0.9935	0.9740	0.9737	0.9781	0.9996	0.9995	0.9995	0.9944	1.7
dae	0.8953	0.9027	0.9058	0.9010	0.9012	0.9072	0.8846	0.9027	0.9047	0.8922	4.2
bin rforest	0.9981	0.9973	0.9974	0.9837	0.9864	0.9877	0.9978	0.9974	0.9973	0.9981	1.3
dist	0.8544	0.8511	0.9326	0.9361	0.9343	0.9331	0.8455	0.8604	0.9311	0.9342	3.8
magnet	0.9220	0.8937	0.9202	0.9077	0.8950	0.9056	0.9024	0.9229	0.9038	0.9171	4

Table 5.28: Defense strategies comparison on NSL-KDD dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
advtrain rforest	0.9921	0.9928	0.9932	0.9732	0.9728	0.9775	0.9995	0.9994	0.9993	0.9942	1.7
dae	0.8794	0.8884	0.8919	0.8865	0.8873	0.8934	0.8663	0.8880	0.8909	0.8755	4.2
bin rforest	0.9974	0.9966	0.9967	0.9828	0.9856	0.9869	0.9971	0.9967	0.9965	0.9974	1.3
dist	0.8265	0.8204	0.9216	0.9263	0.9248	0.9239	0.8156	0.8323	0.9194	0.9242	3.8
magnet	0.9060	0.8769	0.9038	0.8901	0.8796	0.8891	0.8850	0.9078	0.8855	0.9010	4

Table 5.29: Defense strategies comparison on CICIDS dataset (AUC).

AUC	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
advtrain rforest	0.9999	0.9999	0.9999	0.9992	0.9991	0.9910	0.9997	0.9996	0.9996	0.9923	1.6
dae	0.8155	0.8602	0.8284	0.8887	0.8976	0.8766	0.8692	0.8491	0.8840	0.9147	3.2
bin rforest	0.9998	1.0000	1.0000	0.8279	0.7798	0.7495	1.0000	1.0000	0.9999	0.9998	1.9
dist	0.4866	0.4865	0.6953	0.6773	0.7495	0.9255	0.7670	0.8397	0.9435	0.9805	4.1
magnet	0.9541	0.8995	0.6732	0.7842	0.7578	0.8415	0.7688	0.8022	0.8631	0.8373	4.2

Table 5.30: Defense strategies comparison on CICIDS dataset (F1).

F1	FGSM			JSMA			Deepfool			C&W	Mean Ranks
	$\epsilon=0.3$	$\epsilon=0.2$	$\epsilon=0.1$	$\vartheta=0.3$	$\vartheta=0.2$	$\vartheta=0.1$	os=1	os=0.5	os=0.2	default	
advtrain rforest	1.0000	0.9999	1.0000	0.9992	0.9991	0.9907	0.9997	0.9996	0.9996	0.9923	1.6
dae	0.8598	0.8851	0.8878	0.9376	0.8923	0.9098	0.9104	0.8994	0.9201	0.9397	3.2
bin rforest	0.9999	1.0000	1.0000	0.7776	0.7028	0.6642	1.0000	1.0000	1.0000	0.9999	2
dist	0.0138	0.0089	0.0042	0.5059	0.6453	0.9197	0.7019	0.8066	0.9448	0.9849	4.2
magnet	0.9639	0.8971	0.5606	0.7557	0.7073	0.8430	0.7301	0.7759	0.8566	0.8239	4

We can observe that **Adversarial Training** and **Separate Binary Classifier** performed very similarly across both datasets, having a difference of 1% across both metrics in the NSL-KDD dataset, and a difference of 7% on the CICIDS dataset. The larger difference in the CICIDS dataset is mostly due to the ineffectiveness of **Separate Binary Classifier** against JSMA attacks, although its performance had a difference of less than 1% against the other types of attacks.

When comparing the **Denoising Autoencoder**, **Distillation** and **MagNet**, we concluded that their performances were relatively similar across both datasets, and their performance largely depends on the dataset and type of attack, not being generalizable.

We can conclude that, whenever possible, defense strategies that revolve around the inclusion of adversarial samples in the training set should be used, and that the Separate Binary Classifier

Experimental Results

has the advantage of being easier to deploy for this scenario, since it doesn't require retraining of a previously trained model, by instead using a separate model to identify adversarial examples, and keeping the original one to classify normal samples. Meanwhile, when the generation of adversarial examples is not possible for training a classifier, all the three other strategies can be implemented, since all obtained very similar average results, although the Denoising Autoencoder was the most effective against JSMA attacks, which we argued are the most realistic type of attack in an Denial of Service detection scenario.

Experimental Results

Chapter 6

Conclusions and Future Work

Adversarial machine learning is a growing concern with the crescent usage of machine learning on multiple fields, including intrusion and Denial of Service detection.

In this dissertation, we studied the effectiveness of multiple classifiers against adversarial attacks on Denial of Service datasets, by perturbing malicious samples in specific ways to go undetected by classifiers. We then tested the effectiveness of multiple adversarial defense strategies against the previously generated attacks.

With the results obtained from the experiments, we can answer the five questions proposed in chapter 1:

1. **Can the generation of adversarial Denial of Service datasets deteriorate the performance of classifiers?**
 - Yes. A generalized decrease in performance, both in AUC and F1, was observed across both datasets when we generated adversarial datasets using four different attacks: Fast Gradient Sign Method, Jacobian Based Saliency Map Attack, Deepfool and Carlini & Wagner attack. On average, there was a decrease of **13%** of the AUC on the NSL-KDD dataset, and a decrease of **19%** of the AUC on the CICIDS 2017 dataset.
2. **Which classifiers offered higher resilience to adversarial attacks?**
 - When testing the baseline classifiers against adversarial attacks, the one that suffered less decrease in performance was the denoising autoencoder, which only showed a decrease of **1%** of the AUC on both datasets, although support vector machine and random forest also showed resilience when the perturbations had smaller magnitudes.
3. **Which attack strategies are the most effective in each scenario?**
 - The most effective attacks largely depend on the dataset used, and on the proportion of features available to be targeted. On the first dataset, where a smaller proportion of features was targeted, Deepfool was the most effective, while on the second dataset, where a larger proportion of features was targeted, FGSM was the most effective.

4. Can adversarial defense techniques be effective in a Denial-of-Service context?

- Yes. We tested four different adversarial defense strategies: **Adversarial Training**, **Separate Binary Classifier**, **Distillation** and **MagNet**. All showed an improvement of the performance relatively to the average performance observed in the baseline classifiers against adversarial attacks.

5. Which defense techniques are the most effective in each scenario?

- We concluded that strategies that revolved around the inclusion of adversarial samples on the training set (Adversarial Training and Separate Binary Classifier) offered greater performance than the other techniques across all scenarios and datasets, by only decreasing the mean AUC between **0%** and **6%**, while the decrease on the other defenses was between **9%** and **25%** relatively to the best classifier in the baseline experiment (training and testing on the original datasets). Although the results of the two techniques based on training with adversaries were very similar, Separate Binary Classifier can be more convenient, since it doesn't require a new model to be trained to classify regular data, as it uses a separate model to identify adversarial examples.

To extend this dissertation, new directions should be explored to analyze the perturbations generated by the attacks, and replicate their effects on lower level features to allow the reproduction of adversarial Denial of Service attacks in real scenarios. A more thorough analysis of which features are modifiable, while maintaining the integrity of the attack, should also be considered for this scenario. Furthermore, more datasets should be used to achieve more realistic results, and the problem should be explored on a Denial of Service dataset with three classes representing: (1) a normal connection with normal traffic, (2) a normal connection with heavy traffic (which can involuntarily produce Denial of Service) and (3) a malign connection. In the studied datasets, only a normal and malign connection was considered, as no dataset was found for the three-class scenario.

References

- [1] F. Chollet *et al.*, *Keras*, (accessed on 06-2019). [Online]. Available: <https://keras.io>.
- [2] *Cifar-10 dataset*, (accessed on 06-2019). [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [3] *Contagio dataset*, (accessed on 06-2019). [Online]. Available: <http://contagiodump.blogspot.it>.
- [4] *Gtsrb dataset*, (accessed on 06-2019). [Online]. Available: <http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset>.
- [5] *Malwr dataset*, (accessed on 06-2019). [Online]. Available: <https://malwr.com/>.
- [6] *Microsoft kaggle malware dataset*, (accessed on 06-2019). [Online]. Available: <https://www.kaggle.com/c/microsoft-malware-prediction/data>.
- [7] *Mnist handwritten digit dataset*, (accessed on 06-2019). [Online]. Available: <http://yann.lecun.com/exdb/mnist/>.
- [8] Wikipedia, *Marriott International*, (accessed on 06-2019). [Online]. Available: https://en.wikipedia.org/wiki/Marriott_International.
- [9] —, *Yahoo! data breaches*, (accessed on 06-2019). [Online]. Available: https://en.wikipedia.org/wiki/Yahoo!_data_breaches.
- [10] J. Clements *et al.*, “Rallying adversarial techniques against deep learning for network security”, *arXiv e-prints*, 2019. arXiv: [1903.11688](https://arxiv.org/abs/1903.11688).
- [11] M. Ring *et al.*, “A survey of network-based intrusion detection data sets”, *arXiv e-prints*, 2019. arXiv: [1903.02460](https://arxiv.org/abs/1903.02460).
- [12] Q. Yan *et al.*, “Automatically synthesizing dos attack traces using generative adversarial networks”, *International Journal of Machine Learning and Cybernetics*, pp. 1–10, 2019.
- [13] P. Domingos, *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York, NY, USA: Basic Books, Inc., 2018.
- [14] V. Duddu, “A survey of adversarial machine learning in cyber warfare”, *Defence Science Journal*, vol. 68, pp. 356–366, 2018.

REFERENCES

- [15] I. Frazão *et al.*, “Applying machine learning techniques on the detection of cyber-physic attacks”, Master’s thesis, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2018.
- [16] —, “Denial of service attacks: Detecting the frailties of machine learning algorithms in the classification process”, in *Critical Information Infrastructures Security*, 2018, pp. 230–235.
- [17] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, *A taxonomy and survey of intrusion detection system design techniques, network threats and datasets*, 2018. arXiv: [1806.03517](#).
- [18] Z. Lin *et al.*, “IDSGAN: generative adversarial networks for attack generation against intrusion detection”, *arXiv e-prints*, 2018. arXiv: [1809.02077](#).
- [19] Y. Mirsky *et al.*, “Kitsune: An ensemble of autoencoders for online network intrusion detection”, *arXiv e-prints*, 2018. arXiv: [1802.09089](#).
- [20] N. Papernot *et al.*, “Technical report on the cleverhans v2.1.0 adversarial examples library”, *arXiv e-prints*, 2018. arXiv: [1610.00768](#).
- [21] I. Sharafaldin *et al.*, “Toward generating a new intrusion detection dataset and intrusion traffic characterization”, 4th International Conference on Information Systems Security and Privacy, 2018, pp. 108–116.
- [22] Z. Wang, “Deep learning-based intrusion detection with adversaries”, *IEEE Access*, vol. 6, pp. 38 367–38 384, 2018.
- [23] C. Xiao *et al.*, “Generating adversarial examples with adversarial networks”, *arXiv e-prints*, 2018. arXiv: [1801.02610](#).
- [24] K. Yang *et al.*, “Adversarial examples against the deep learning based network intrusion detection systems”, *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*, pp. 559–564, 2018.
- [25] N. Akhtar *et al.*, “Defense against universal adversarial perturbations”, *arXiv e-prints*, 2017. arXiv: [1711.05929](#).
- [26] H. S. Anderson *et al.*, “Evading machine learning malware detection”, *Black Hat*, 2017. [Online]. Available: <https://blackhat.com/>.
- [27] M. Antonakakis *et al.*, “Understanding the mirai botnet”, in *Proceedings of the 26th USENIX Conference on Security Symposium*, 2017, pp. 1093–1110.
- [28] M. Arjovsky *et al.*, “Wasserstein gan”, *arXiv e-prints*, 2017. arXiv: [1701.07875](#).
- [29] B. Biggio *et al.*, “Evasion attacks against machine learning at test time”, *arXiv e-prints*, 2017. arXiv: [1708.06131](#).
- [30] N. Carlini *et al.*, “Towards evaluating the robustness of neural networks”, *arXiv e-prints*, 2017. arXiv: [1608.04644](#).

REFERENCES

- [31] P.-Y. Chen *et al.*, “Ead: Elastic-net attacks to deep neural networks via adversarial examples”, *arXiv e-prints*, 2017. arXiv: [1709.04114](#).
- [32] —, “Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models”, *arXiv e-prints*, 2017. arXiv: [1708.03999](#).
- [33] Z. Gong *et al.*, “Adversarial and Clean Data Are Not Twins”, *arXiv e-prints*, 2017. arXiv: [1704.04960](#).
- [34] K. Grosse *et al.*, “Adversarial examples for malware detection”, in *European Symposium on Research in Computer Security*, 2017, pp. 62–79.
- [35] H. Hosseini *et al.*, “Blocking transferability of adversarial examples in black-box learning systems”, *arXiv e-prints*, 2017. arXiv: [1703.04318](#).
- [36] W. Hu and Y. Tan, “Generating adversarial malware examples for black-box attacks based on gan”, *arXiv e-prints*, 2017. arXiv: [1702.05983](#).
- [37] A. Madry *et al.*, “Towards deep learning models resistant to adversarial attacks”, *arXiv e-prints*, 2017. arXiv: [1706.06083](#).
- [38] D. Meng and H. Chen, “Magnet: A two-pronged defense against adversarial examples”, *arXiv e-prints*, 2017. arXiv: [1705.09064](#).
- [39] M. Rigaki *et al.*, “Adversarial deep learning against intrusion detection classifiers”, Master’s thesis, Information Security’s master dissertation, Luleå University of Technology, 2017.
- [40] S. Sankaranarayanan *et al.*, “Regularizing deep networks using efficient layerwise adversarial training”, *arXiv e-prints*, 2017. arXiv: [1705.07819](#).
- [41] W. Xu *et al.*, “Feature squeezing: Detecting adversarial examples in deep neural networks”, *arXiv e-prints*, 2017. arXiv: [1704.01155](#).
- [42] X. Yuan *et al.*, “Adversarial examples: Attacks and defenses for deep learning”, *arXiv e-prints*, 2017. arXiv: [1712.07107](#).
- [43] M. Abadi *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous systems”, *arXiv e-prints*, 2016. arXiv: [1603.04467v2](#).
- [44] K. N. Mallikarjunan *et al.*, “A survey of distributed denial of service attack”, in *10th International Conference on Intelligent Systems and Control (ISCO)*, 2016, pp. 1–6.
- [45] A. Ozgur and H. Erdem, “A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015”, *PeerJ Preprints*, vol. 4, e1954, 2016.
- [46] Papernot *et al.*, “Distillation as a defense to adversarial perturbations against deep neural networks”, *arXiv e-prints*, 2016. arXiv: [1511.04508](#).
- [47] N. Papernot *et al.*, “Practical black-box attacks against machine learning”, *arXiv e-prints*, 2016. arXiv: [1602.0269](#).
- [48] —, “Towards the science of security and privacy in machine learning”, *arXiv e-prints*, 2016. arXiv: [1611.03814](#).

REFERENCES

- [49] —, “Transferability in machine learning: From phenomena to black-box attacks using adversarial samples”, *arXiv e-prints*, 2016. arXiv: [1605.07277](#).
- [50] T. Salimans *et al.*, “Improved techniques for training gans”, *arXiv e-prints*, 2016. arXiv: [1606.03498](#).
- [51] L. Dhanabal and S. S. Periyasamy, “A study on nsl-kdd dataset for intrusion detection system based on classification algorithms”, vol. 4, pp. 446-452, 2015.
- [52] I. Goodfellow *et al.*, “Explaining and harnessing adversarial examples”, *arXiv e-prints*, 2015. arXiv: [1412.6572](#).
- [53] G. Hinton *et al.*, “Distilling the knowledge in a neural network”, *arXiv e-prints*, 2015. arXiv: [1503.02531](#).
- [54] M. Hossin and S. M.N, “A review on evaluation metrics for data classification evaluations”, *International Journal of Data Mining Knowledge Management Process*, vol. 5, pp. 1–11, 2015.
- [55] S. Moosavi-Dezfooli *et al.*, “Deepfool: A simple and accurate method to fool deep neural networks”, *arXiv e-prints*, 2015. arXiv: [1511.04599](#).
- [56] N. Papernot *et al.*, “The limitations of deep learning in adversarial settings”, *arXiv e-prints*, 2015. arXiv: [1511.07528](#).
- [57] I. J. Goodfellow *et al.*, “Generative adversarial networks”, *arXiv e-prints*, 2014. arXiv: [1406.2661](#).
- [58] J. Vacca, *Network and system security*. Syngress, 2014.
- [59] C. Szegedy *et al.*, “Intriguing properties of neural networks”, *arXiv e-prints*, 2013. arXiv: [1312.6199](#).
- [60] M. Zamani, “Machine learning techniques for intrusion detection”, *arXiv e-prints*, 2013. arXiv: [1312.2177](#).
- [61] S. T. Zargar *et al.*, “A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks”, *IEEE Communications Surveys Tutorials*, vol. 15, pp. 2046–2069, 2013.
- [62] P. Baldi, “Autoencoders, unsupervised learning, and deep architectures”, in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012, pp. 37–49.
- [63] D. Maiorca *et al.*, “A pattern recognition system for malicious pdf files detection”, in *Machine Learning and Data Mining in Pattern Recognition*, 2012, pp. 510–524.
- [64] Huang *et al.*, “Adversarial machine learning”, in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, pp. 43–58.
- [65] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

REFERENCES

- [66] J. Song *et al.*, “Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation”, in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, vol. 2, p. 3, 2011.
- [67] P. Laskov and R. Lippmann, “Machine learning in adversarial environments”, *Machine Learning*, vol. 81, pp. 115–119, 2010.
- [68] C.-F. Tsai *et al.*, “Intrusion detection by machine learning: A review”, *Expert Systems with Applications*, vol. 16, pp. 303–336, 2009.
- [69] P. Vincent *et al.*, “Extracting and composing robust features with denoising autoencoders”, in *Proceedings of the 25th International Conference on Machine Learning*, 2008, pp. 1096–1103.
- [70] J. Demsar, “Statistical comparisons of classifiers over multiple data sets”, *Journal of Machine learning research*, vol. 7, pp. 1–30, 2006.
- [71] R. A. Kemmerer, “Cybersecurity”, in *25th International Conference on Software Engineering*, 2003, pp. 705–715.

REFERENCES

Appendix A

Datasets feature list

Datasets feature list

Table A.1: Complete list of features on the NSL-KDD dataset.

No.	Name	Type	Description	No.	Name	Type	Description
1	Duration	Continuous	Duration of the connection	22	Is guest login	Discrete	1 if the login is a guest, 0 otherwise
2	Protocol Type	Discrete	Type of protocol of the connection	23	Count	Continuous	Num. of connections to the same destination in the past 2 seconds
3	Service	Discrete	Destination Service Used	24	Srv count	Continuous	Num. of connections to the same service in the past 2 seconds
4	Flag	Discrete	Status of the connection	25	Error. rate	Continuous	Prcnt. of connections in Count that activated the s0, s1, s2 or s3 flags
5	Src_bytes	Continuous	Bytes transfered from source to destination	26	Srv error rate	Continuous	Prcnt. of connections in Srv count that activated the s01,s1,s2 or s3 flags
6	Dst_bytes	Continuous	Bytes transfered from destination to source	27	Error rate	Continuous	Prcnt. of connections in Count that activated the REJ flag
7	Land	Discrete	1 if the source and destination have the same ip and port, 0 otherwise	28	Srv error rate	Continuous	Prcnt. of connections in Srv count that activated the REJ flag
8	Wrong Fragment	Continuous	Number of wrong fragments in connection	29	Same srv. rate	Continuous	Prcnt. of connections in Srv count that were made to the same service
9	Urgent	Continuous	Number of packets with urgent bit set to True.	30	Diff srv rate	Continuous	Prcnt. of connections in Srv count that were made to different services
10	Hot	Continuous	Number of 'hot' indicators in the connection, such as accessing an operating system resources.	31	Srv diff host rate	Continuous	Prcnt. of connections in Srv count that were made to different machines
11	Num failed logins	Continuous	Number of failed login attempts.	32	Dst host count	Continuous	Number of connections having same destination Ip address
12	Logged In	Continuous	1 if logged in during connection, 0 otherwise	33	Dst host srv count	Continuous	Number of connections with same port number
13	Num compromised	Continuous	Number of compromised conditions	34	Dst host same srv rate	Continuous	Prcnt. of connections from dst host count to the same service
14	Root shell	Discrete	1 if a rootshell is used, 0 otherwise	35	Dst host diff srv rate	Continuous	Prcnt. of connections from dst host count to different services
15	Su attempted	Discrete	1 if the 'su root' command was used, 0 otherwise	36	Dst host same src port rate	Continuous	Prcnt. of connections from dst host srv count to the same port
16	Num root	Continuous	Number of root operations performed	37	Dst host srv diff host rate	Continuous	Prcnt. of connections from dst host srv count to different machines
17	Num file creations	Continuous	Number of file creations during connection	38	Dst host error rate	Continuous	Prcnt. of connections from dst host count with s0, s1, s2 or s3 flags set to 1
18	Num shells	Continuous	Number of shell prompts	39	Dst host srv error rate	Continuous	Prcnt. of connections from dst host srv count with s0, s1, s2 or s3 flags set to 1
19	Num access files	Continuous	Number of accesses to control files	40	Dst host error rate	Continuous	Prcnt. of connections from dst host count with REJ flag set to 1
20	Num outbound cmds	Continuous	No. of outbound commands in ftp connection	41	Dst host srv error rate	Continuous	Prcnt. of connections from dst host srv count with REJ flag set to 1
21	Is hot login	Discrete	1 if hot login (e.g. admin), 0 otherwise				

Datasets feature list

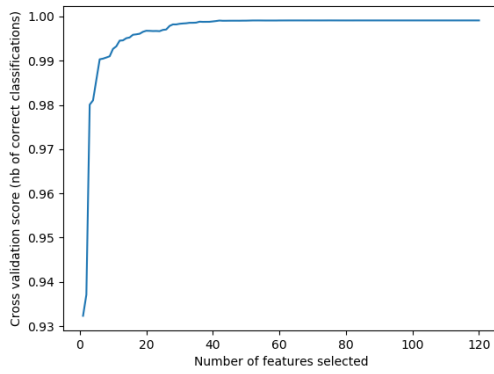
Table A.2: List of features on the CIC-IDS 2017 dataset. Some features were suppressed due to being statistical information about a certain property (identified with *(s)*)

No.	Name	Type	Description	No.	Name	Type	Description
1	Destination Port	Discrete	Port used by the connection	22	RST flag count	Continuous	Number of times a packet in the flow had the reset flag set to 1
2	Flow duration	Continuous	Duration of the flow	23	PSH flag count	Continuous	Number of times a packet in the flow had the push flag set to 1
3	Total Fwd Packets	Continuous	Number of packets in forward direction	24	ACK flag count	Continuous	Number of times a packet in the flow had the acknowledged flag set to 1
4	Total Bwd Packets	Continuous	Number of packets in backward direction	25	URG flag count	Continuous	Number of times a packet in the flow had the urgent flag set to 1
5	Statistics on Length of Fwd packets <i>(s)</i>	Continuous	Statistics on length of packets in forward direction	26	ECE flag count	Continuous	Number of times a packet in the flow had the explicit notif. echo flag set to 1
6	Statistics on Length of Bwd packets <i>(s)</i>	Continuous	Statistics on length of packets in backward direction	27	CWE flag count	Continuous	Number of times a packet in the flow had the congestion wind. reduced flag set to 1
7	Flow bytes/s	Continuous	Bytes per second in the flow	28	Down/Up ratio	Continuous	Download and upload ratio
8	Flow packets/s	Continuous	Packets per second in the flow	29	Average packet size	Continuous	Average size of the packets in the flow
9	Statistics on flow IAT <i>(s)</i>	Continuous	Statistics on the inter-arrival time of the flow	30	Avg Fwd Segment Size	Continuous	Average segment size in the forward direction
10	Statistics on bwd IAT <i>(s)</i>	Continuous	Statistics on the inter-arrival time of the bwd packets	31	Avg Bwd Segment Size	Continuous	Average segment size in the backward direction
11	Statistics on fwd IAT <i>(s)</i>	Continuous	Statistics on the inter-arrival time of the fwd packets	32	Fwd Header Length	Continuous	Average Header length in the forward direction
12	Fwd/Bwd PSH flags	Continuous	Number of packets in the flow with push flag set to 1 in forward and backward direction	33	Statistics on bulk rate <i>(s)</i>	Continuous	Statistics on bulk rates (data transfer mechanism to transfer large files)
13	Bwd URG flags	Continuous	Number of packets in the flow with urgent flag set to 1 in backward direction	34	Subflow Fwd Packets	Continuous	The average number of packets in a subflow in the forward direction
14	Fwd URG flags	Continuous	Number of packets in the flow with urgent flag set to 1 in forward direction	35	Subflow Bwd Packets	Continuous	The average number of packets in a subflow in the backward direction
15	Bwd Header length	Continuous	Header length of packets in backward direction	36	Subflow Fwd Bytes	Continuous	The average number of bytes in a subflow in the forward direction
16	Fwd Header length	Continuous	Header length of packets in forward direction	37	Subflow Bwd Bytes	Continuous	The average number of bytes in a subflow in the backward direction
17	Fwd Packets/s	Continuous	Packets per second in forward direction	38	Init Win bytes fwd	Continuous	Number of bytes sent in the initial window in forward direction
18	Bwd Packets/s	Continuous	Packets per second in backwards direction	39	Init Win bytes bwd	Continuous	Number of bytes sent in the initial window in backward direction
19	Statistics on Packet Length <i>(s)</i>	Continuous	Statistics on the packet lengths	40	Act data pkt fwd	Continuous	Number of packets with at least one byte of tcp data in forward direction
20	FIN flag count	Continuous	Number of times a packet in the flow had the finish flag set to 1	41	Statistics on active <i>(s)</i>	Continuous	Statistics on amount of time a flow was active during a flow
21	SYN flag count	Continuous	Number of times a packet in the flow had the synchronize flag set to 1	42	Statistics on idle <i>(s)</i>	Continuous	Statistics on amount of time a flow was idle during a flow

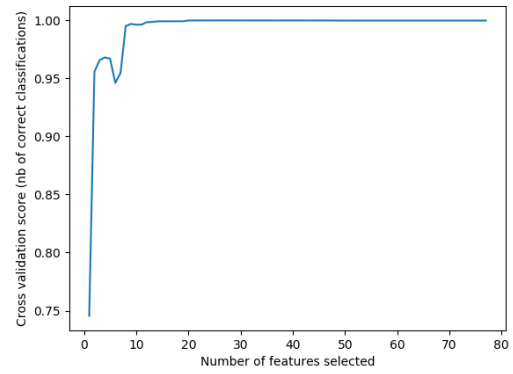
Datasets feature list

Appendix B

Feature selection



(a) NSL-KDD



(b) CICIDS 2017

Figure B.1: Feature selection process using Recursive Feature Elimination on both datasets.

Feature selection

Table B.1: List of features after the selection step (NSL-KDD: left, CICIDS 2017: right)

No.	Feature	No.	Feature	No.	Feature	No.	Feature
1	Duration	22	service-ntp_u	1	Destination Port	22	URG Flag Count
2	protocol_type-icmp	23	service-other	2	Flow Duration	23	Avg Packet Size
3	protocol_type-udp	24	service-pop_2	3	Fwd Packet Length Max	24	Avg Fwd Seg Size
4	service-X11	25	service-pop_3	4	Fwd Packet Length Mean	25	Init Win Bytes Fwd
5	service-Z39_50	26	service-printer	5	Fwd Packet Length Std	26	Min Seg Size Fwd
6	service-auth	27	service-supdup	6	Bwd Packet Length Mean		
7	service-bgp	28	service-tim_i	7	Bwd Packet Length Std		
8	service-courier	29	service-time	8	Flow Bytes/s		
9	service-csnet_ns	30	service-vmnet	9	Flow IAT Max		
10	service-discard	31	logged_in-l	10	Fwd IAT Total		
11	service-domain_u	32	su_attempted-0	11	Fwd IAT Max		
12	service-echo	33	num_shells	12	Bwd IAT Total		
13	service-ecr_i	34	count	13	Fwd Packets/s		
14	service-efs	35	srv_count	14	Bwd Packets/s		
15	service-exec	36	rerror_rate	15	Max Packet Length		
16	service-gopher	37	dst_host_count	16	Packet Length Variance		
17	service-klogin	38	dst_host_srv_count	17	Packet Length Std		
18	service-ldap	39	dst_host_diff_srv_rate	18	Packet Length Mean		
19	service-netbios_ssn	40	dst_host_same_src_port_rate	19	FIN Flag Count		
20	service-nnsp	41	dst_host_srv_error_rate	20	SYN Flag Count		
21	service-nntp			21	ACK Flag Count		

Appendix C

Optimal parameters

Table C.1: Optimal parameters for classifiers on original NSL-KDD dataset.

Classifier	Parameters		
dtree	Criterion: Entropy	Max Depth: 9	
rforest	Criterion: Entropy	Max Depth: 9	N. of Estimators: 200
svm	Cost: 1000	Gamma:1	Kernel: RBF
naive	Laplace Smoothing: 1.0		
nn	Neurons per Layer: 41-51-51-41	Dropout Rate: 0.3	
dae	Neurons per Layer: 41-48-55-48-41	Gaussian Noise Std. dev: 0.5	

Table C.2: Optimal parameters for classifiers on original CICIDS 2017 dataset.

Classifier	Parameters		
dtree	Criterion: Entropy	Max Depth: 9	
rforest	Criterion: Gini	Max Depth: 9	N. of Estimators: 100
svm	Cost: 10000	Gamma:1	Kernel: RBF
naive	Laplace Smoothing: 0.0001		
nn	Neurons per Layer: 26-36-36-26	Dropout Rate: 0.3	
dae	Neurons per Layer: 26-34-41-34-26	Gaussian Noise Std. dev: 0.5	