Protecting network intrusion detection systems from evasion attacks

Thesis presented by Islam Debicha

in fulfilment of the requirements of the joint PhD in computer science from Université Libre de Bruxelles and Royal Military Academy

Supervisor: Dr. Thibault Debatty

Co-supervisor: Prof. Jean-Michel Dricot

Thesis jury :

Prof. Gianluca Bontempi (Université Libre de Bruxelles)

Prof. Jamal El Hachem (University of South Brittany)

Prof. Filip Van Utterbeeck (Royal Military Academy)

Prof. Wim Mees (Royal Military Academy)

Academic year 2022-2023

Title: Protecting network intrusion detection systems from evasion attacks
Author: Islam Debicha
PhD in computer science
Academic year: 2022 – 2023

Abstract

Nowadays, numerous applications incorporate machine learning algorithms due to their prominent achievements. However, many studies in the field of computer vision have shown that machine learning can be fooled by intentionally crafted instances, called adversarial examples. These adversarial examples take advantage of the intrinsic vulnerability of machine learning models. This vulnerability raises many concerns in the cybersecurity field since an increasing number of security systems are powered by machine-learning algorithms.

In this thesis, we explored the effects of adversarial machine learning on cybersecurity systems driven by machine learning models, focusing on intrusion detection systems. To do so, we implement and evaluate evasion attacks in both black-box and white-box settings to generate adversarial network traffic able to fool the intrusion detection system. We also design and test novel evasion attacks and adversarial defenses to improve the robustness of intrusion detection systems.

The experimental results demonstrated that machine learning-based intrusion detection systems are vulnerable to adversarial attacks generated by adding minor specially crafted perturbations to malicious network traffic, allowing the attacker to evade detection and thus successfully perform his initial attacks. Adversarial detection, on the other hand, provides an efficient way to mitigate the effect of adversarial attacks at the expense of increasing model complexity by adding a second line of defense.

Keywords: Intrusion detection system, Machine learning, Adversarial learning, Evasion attack, Adversarial defense

Acknowledgements

I don't know where to start except to thank God the Almighty for giving me the strength and courage to continue to make efforts and to reach the point I have reached today.

First of all, I would like to express my most sincere thanks to my supervisors Dr. Thibault Debatty and Prof. Jean-Michel Dricot for the trust they placed in me, for their availability and their judicious advice throughout this work. I also thank Prof. Tayeb Kenaza for his advice, his help and his sympathy. I also thank the members of the jury, who accepted to honor me by agreeing to examine, judge and evaluate my work.

Finally, I am extremely grateful to all those who have contributed in any way to the realization of this work.

Table of Contents

List of Figures

List of Tables

Introduction

Motivation

With the growth of information communication and in particular the growth of the Internet, the concern to secure this information and related systems has become increasingly important. Every day, massive amounts of sensitive information are created, transferred, stored, or updated around the world. This sensitive information can range from personal emails to banking transactions, from simple holiday photos to military communication. This information has always been the target of malicious entities wanting to steal, modify or delete it. To achieve these goals, hackers and other malicious agents have discovered, exploited, and improved a wide range of cyberattacks.

This new era of cyber security has required a paradigm shift from simple defense mechanisms to sophisticated defense systems. While basic network protections, such as firewalls, may have been sufficient in the past, the increasing complexity of cyberattacks has made them insufficient if used alone. To guard against these complex attacks, Intrusion Detection Systems (IDS) are now the cornerstone of cyber security.

As the sophistication of attacks increases, weaknesses are also discovered and exploited at an increasing rate. Attacks that exploit weaknesses almost as soon as they are discovered are called zero-day attacks. With zero-day hacking attacks breaking records year after year [5], it becomes necessary for IDSs to be able to detect never-before-seen attacks. Machine learning-based IDSs are one of the solutions to this problem. Indeed, numerous research papers have shown that the use of machine learning provides new approaches and detection mechanisms against zero-day attacks in all areas of cybersecurity [6–9].

As always in the ongoing rivalry between cyber-attacks and defenses, as machine learning has become an important part of defense mechanisms, it has also become the target of attacks. The field of adversarial learning studies various methods of attack against machine learning (ML) and different ways of protecting models

against these attacks. These new attacks take advantage of the fact that machine learning models have a discontinuous input-output mapping. This means that humanly imperceptible changes to the input of a model can cause a dramatic change in the classification result. These changes or perturbations in the input are called adversarial examples [10].

Research in the field of computer vision, where this vulnerability was first discovered, has shown that adversarial images designed to fool a specific model can, to some extent, fool other machine learning models [11]. This is known as the transferability property of adversarial attacks. By exploiting this property, an attacker can build a surrogate intrusion detection system, create adversarial traffic for that detector, and then attack another intrusion detection system without even knowing the internal architecture of that detector, leading to a black-box attack.

This change in cyber-target has led to an equivalent change in cyber defenses, as the focus has shifted to protecting IDSs themselves from attacks created to undermine their effectiveness. While adversarial learning features some defenses, more research is needed to refine these techniques. Some defenses have shown promising results in the field of computer vision, but very limited work has been done regarding these approaches in the field of intrusion detection systems.

In addition, a significant amount of research on the impact of adversarial learning in computer vision has been transferred into intrusion detection. Initial results have shown that the classifiers used in IDS are also vulnerable to these algorithms. A typical approach used by researchers is to focus on the theoretical aspect of the problem by setting simplifying assumptions and focusing only on the feature space [12]. However, unlike computer vision where the created perturbations have relatively few constraints, a valid network traffic perturbation must satisfy many domain-specific constraints (both semantic and syntactic). These domain-specific constraints ensure that the added perturbation will generate valid network traffic enabling the transition from feature space to traffic space. Unfortunately, network-specific constraints are often not considered or only to a limited extent. This means that the feasibility of attacks from a realistic point of view is not fully considered.

Goal of this thesis

The goal of this thesis is to investigate the implications of adversarial machine learning on cybersecurity systems powered by machine learning models, specifically intrusion detection systems. Six main contributions are presented in this thesis:

- Since a considerable amount of research in the field of computer vision has been translated for use in intrusion detection without addressing domain restrictions, we present a revised review of the state of the art based on the feasibility of adversarial attacks and defenses. (Chapier 1)
- A novel intrusion detection system based on evidence theory is presented, along with a novel contextual discounting method based on the reliability of the sources. (Chapier 2)
- A study is conducted to determine how adversarial attacks affect deep learning-based intrusion detection systems. In addition, the effectiveness of Adversarial Training as a defense against adversarial attacks for intrusion detection systems is investigated. (Chapier 3)
- In a black-box setting, the transferability of adversarial network traffic between multiple anomaly-based intrusion detection systems is investigated using various machine learning techniques. Furthermore, the effectiveness of the ensemble method and the detection and rejection method as a defensive mechanism for mitigating the effect of adversarial transferability is investigated. (Chapier 4)
- Proposal of a novel adversarial defense in a white-box setting by designing and testing the use of multiple strategically placed transfer learning-based detectors of adversarial attacks. (Chapier 5)
- Investigate the actual feasibility of evasion attacks against network-based intrusion detection systems (NIDS), demonstrating that using our proposed adversarial algorithm and as many constraints as possible in a black-box setting, it is entirely possible to fool these ML-based IDSs. Furthermore, because designing defense mechanisms to protect these ML-based IDSs is critical, a defensive scheme is presented. (Chapier 6)

Publications

This thesis led to the following contributions in recognized international conferences and scientific journals:

Journal Articles

1. **Debicha**, I., Bauwens, R., Debatty, T., Dricot, J. M., Kenaza, T., & Mees, W. (2023). "TAD: Transfer learning-based multi-adversarial detection of evasion attacks against network intrusion detection systems". Future Generation Computer Systems, 138, 185-197, Elsevier.

2. **Debicha**, I., Cochez, B.,Kenaza, T., Debatty, T., Dricot, J. M., & Mees, W. "Review on the Adversarial Evasion Attacks and Defenses for Network Intrusion Detection Systems". (Computer Networks, Elsevier - Under review)

3. **Debicha**, I., Cochez, B.,Kenaza, T., Debatty, T., Dricot, J. M., & Mees, W. "Adv-Bot: Realistic Adversarial Botnet Attacks against Network Intrusion Detection Systems". (Computers & Security, Elsevier - Under review)

Conference Articles

1. **Debicha**, Islam, Thibault Debatty, Jean-Michel Dricot, Wim Mees, and Tayeb Kenaza. "Detect & Reject for Transferability of Black-box Adversarial Attacks Against Network Intrusion Detection Systems." In The Third International Conference On Advances In Cyber Security (ACES 2021). Springer, Singapore.

2. **Debicha**, Islam, Thibault Debatty, Jean-Michel Dricot, and Wim Mees. "Adversarial Training for Deep Learning-based Intrusion Detection Systems." In The Sixteenth International Conference on Systems (ICONS 2021), Porto - Portugal.

3. **Debicha**, Islam, Thibault Debatty, Wim Mees, and Jean-Michel Dricot. "Efficient Intrusion Detection Using Evidence Theory." In The Twelfth International Conference on Evolving Internet (INTERNET 2020), Porto - Portugal.

Chapter 1

Literature review, state of the art, definitions and notations

1.1 Introduction

Today, problems related to the security of ML-based IDS are an active research topic [13, 14]. A significant amount of research on the impact of adversarial learning in computer vision has been transferred into intrusion detection. Initial results have shown that the classifiers used in IDS are also vulnerable to these algorithms. A typical approach used by researchers is to focus on the theoretical aspect of the problem by setting simplifying assumptions and focusing only on the feature space [15–17]. However, unlike computer vision where the created perturbations have relatively few constraints, a valid network traffic perturbation must satisfy many domain-specific constraints (both semantic and syntactic). These domain-specific constraints ensure that the added perturbation will generate valid network traffic enabling the transition from feature space to traffic space. Unfortunately, network-specific constraints are often not considered or only to a limited extent. This means that the feasibility of attacks from a realistic point of view is not fully considered. Some researchers [18–20] have decided to take a different approach to deal with the problem by limiting the need for feature knowledge by directly manipulating the traffic space.

This chapter is therefore a revised review of the state of the art providing a new aspect based on the feasibility of adversarial attacks and defenses. We also provide an update on new contributions that have been produced recently concerning the feasibility of attacks in real settings:(i) we propose a complete analysis, for each selected paper, on the real feasibility of the proposed attacks by demonstrating whether or not the constraints of the domain are respected. In addition, (ii) we propose an analysis of the defenses used in the papers studied to highlight the

strengths and weaknesses of each. Finally,(iii) we identify some realistic aspects that should be considered for future studies of the impact of adversarial attacks on IDSs.

The rest of the chapter is structured as follows. Section 1.2 gives a theoretical reminder introducing the key concepts used in the reviewed papers. Section 1.3 summarizes previous reviews that have been conducted on the subject. Section 1.4 describes the most commonly used state-of-the-art attacks in the literature. Section 1.5 shows the most popular defense mechanisms used in the literature to counter the attacks described in Section 1.4 . Section 1.6 contains a detailed analysis of the realism of the selected papers. Section **??** discusses the actual feasibility of the attacks present in Section 1.6 and the challenges associated with them. Section 1.8 concludes the chapter by providing the key points that have been discussed.

1.2 Background

1.2.1 Anomaly-based IDS

The increasing development of new threats targeting network infrastructures worldwide has pushed researchers to develop new defenses. Due to the huge number of undiscovered attacks, most defense mechanisms are unable to cope with such threats. To mitigate this problem, solutions such as Anomaly-Based IDS (AIDS), which can detect some of these previously unknown attacks through the use of statistics and machine learning algorithms, are gaining popularity. AIDS have many properties, among which we note their ability to be deployed in a network (NIDS) or directly on a host (HIDS). As far as NIDS are concerned, two different types can be found: packet-based and flow-based. The data used by NIDS to collect this information can come from different sources such as network protocols like NetFlow/IPFIX, SNMP, or directly from an agent. NIDS can also use application logs from anti-virus or firewalls. To measure the performance of NIDS, different metrics are used such as true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), false negative rate (FNR), accuracy, precision, F1 score, error rate, area under the curve (AUC). All these metrics are derived from the *confusion matrix* shown in Table 1.1.

	Predicted Class	
Actual Class	Anomaly	Normal
Anomaly	True Positive (TP)	False Negative (FN)
Normal	False Positive (FP)	True Negative (TN)

Table 1.1: IDS confusion matrix

1.2.2 Threat model

Although there are several types of threats, an attacker often seeks to violate one of the following properties: confidentiality, integrity, authenticity, and availability.

In terms of threat modeling, there are two important points to consider, namely the knowledge restriction corresponding to the complexity of the attack and the objective of the attack, corresponding to the capability of the attack.

Knowledge restriction As shown in Figure 1.1, attacks can be conducted in two forms, black box or white box. The white box attack means that the adversary knows everything about the training dataset and the model architecture, in particular all the parameters and meta-parameters which are for example the inputs, the gradients (for DNNs), the tree depth (for decision trees) or the number of neighbors (for K-nearest neighbors) as well as the chosen cost function or the type of optimizer (e.g., ADAM or RMSProp) in case of neural networks. The black box refers to the fact that the attacker knows nothing about the target model, i.e., the architecture of the model and the dataset used. The attacker can only send requests to the targeted model and receive answers in the form of decisions or probabilities (logits). He must, therefore, without knowing any information about the model, approximate a decision boundary similar to that of the target model to be able to craft adversarial samples. Another option for black box attacks is exploiting transferability. An attacker can create a surrogate model, similar in functionality to the targeted model, craft adversarial instances to fool the surrogate model, and then transfer those instances to the targeted model so that it will also be fooled. Black-box attacks are more complicated to perform since less knowledge is available, but also because more computational resources are needed to accommodate this accumulated knowledge (queries).

Attack objective Another relevant property of an attack is its objective. There are two different types of objectives, the untargeted attack, and the targeted attack. A non-targeted attack is easier to perform since all the attacker has to do is trick the machine learning model without any particular considerations. Two possible scenarios can be expected. The first is confidence reduction, which means that the attacker simply wants to decrease the performance of the model while maintaining the overall functionality. The second scenario is misclassification. In this case, the adversary's goal is to trick the model into misclassifying without specific constraints. In a targeted attack, the adversary's goal is to force an ML model to produce the desired output by manipulating the input. This type of objective is therefore more complicated to achieve because it requires manipulating the model in a specific direction, unlike a non-targeted attack that is not limited to a certain target. There are two variants of targeted attacks that can be highlighted. The first is targeted misclassification, which means that an attacker wants to cause misclassification in a certain target class with any input. The other variant is source/target misclassification, which means that an attacker wants to cause misclassification in a certain target class with a certain input. This particular goal is the most difficult to achieve.

1.2.3 Adversarial examples

Adversarial learning refers to the problem of designing attacks against machine learning as well as defenses against these attacks. Depending on the phase in which the attack is carried out, adversarial attacks can be divided into poisoning and evasion attacks. This thesis focuses on evasion attacks. This choice is due to the fact that this review wants to focus on the most realistic aspects of adversarial attacks against NIDS. The problem with poisoning attacks is that they require the ability to directly manipulate the model training data. It is clear that in a realistic scenario, the attacker's knowledge will be limited, and it will be less possible to manipulate the model before its training phase.

The creation of adversarial examples can be expressed as an initial problem

formulation as defined in Eq. 1.1.

$$\begin{aligned} \text{Minimize: } & D(x, x+\delta) \\ \text{Such that: } & C(x+\delta) = t \text{—- constraint 1} \\ & x+\delta \in [0,1]^n \text{—- constraint 2} \end{aligned} \tag{1.1}$$

where we want to minimize the distance between the original element and the adversarial element D(x, x + δ) respecting 2 constraints. The first is that the classification C of x + δ must be classified as the target label t. The second is that x + δ must be a valid element.

According to the work of Szegedy *et al.* [21], adversarial examples exploit the fact that neural networks have "blind spots". The cause of this "blind spot" effect would be due to the models being non-linear and trying to behave linearly as concluded by Goodfellow *et al.* [22]. These adversarial examples have certain properties described below.

L_P norms To compute the distance between the original element x and the perturbed element x_{adv}, an L_P norm (i.e., distance metric) is used such as L_0,L_1, L_2 and L_∞ allowing to define the boundary of adversarial examples. These norms are thus used to minimize the perturbation rate used to generate the adversarial example. The most common norms used by adversarial algorithms are:

L_0: This distance metric counts the number of features of x modified in x_{adv}. This metric only takes into account the number of modified features regardless of the perturbation rate introduced in each feature.

L_1: This norm represents the Manhattan distance between x and x_{adv} as defined in Eq. 1.2.

$$L_1 = |x_1 - x_{1adv}| + ... + |x_n - x_{nadv}| \tag{1.2}$$

L_2: This norm calculates the Euclidean distance or the mean-squared error between x and x_{adv} as shown in Eq. 1.3.

$$L_2 = \sqrt{(x_1 - x_{1adv})^2 + ... + (x_n - x_{nadv})^2} \tag{1.3}$$

L_∞: This norm gives the largest change among all features of x_{adv} compared to

x and it's defined in the following Eq. 1.4.

$$L_\infty = max(|x_1 - x_{1adv}|, ..., |x_n - x_{nadv}|) \tag{1.4}$$

Attack frequency Attack frequency is a property that defines whether the attack is executed in a one-step iteration or requires several. Thus, there are two types of attacks: one-step attacks and iterative attacks. One-step attacks mean that the adversarial examples are generated by an algorithm that executes only once, i.e., it does not iterate multiple times to optimize the adversarial example. Thus, one-step attacks are faster but less optimized. Iterative attacks on the other hand use iterative functions to generate adversarial examples so that it maximizes their efficiency but takes more time.

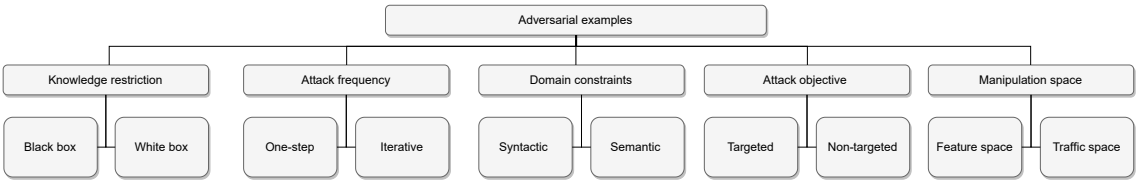


Figure 1.1: Classification of adversarial examples

Domain constraints The feasibility of adversarial attacks is domain-specific and is influenced by several constraints. These constraints can be divided into two main categories: syntactic constraints and semantic constraints. The following syntactic constraints were originally discussed by Merzouk *et al.* [23]. As for the semantic links, the exact definition of these constraints is difficult since they are specific to each domain and even to each type of feature used. However, we draw on the work of Hashemi *et al.* [24], and Teuffeunbach *et al.* [25] in the IDS domain to provide a generalization of three different groups with different semantic links.

Syntactic constraints concern all those related to syntax, e.g., out-of-range values, non-binary values and multiple category membership. Out-of-range values are values that exceed a theoretical maximum value that cannot be exceeded, for example, a float between 0 and 1 or an integer between 0 and 255. Non-binary values are entries that violate the binary nature of a feature and multiple category membership are values that violate the one-hot encoding concept.

Semantic Links represent the links that certain features may have with each

other. These features can be grouped into three distinct groups, each with different semantic properties. The first set includes features that cannot be modified (e.g., IP address, protocol type). The second group includes features that can be directly modified (number of forward packets, size of the forward packet, flow duration, ...). The last group concerns the features that depend on the second group. They must be recalculated based on the latter (number of packets/second or average forward packet size).

This implies that the complexity of generating realistic adversarial examples varies with the different types of data used to represent the domain, such as numerical (continuous or discrete) or categorical data. It also depends on the context in which the model is located (such as network traffic). The NIDS domain is therefore strongly affected by both semantic and syntactic constraints as it uses heterogeneous data types, and its context requires several semantic links most of the time unlike other domains such as computer vision.

Manipulation space An essential property of a realistic adversarial instance is the ability of an attacker to modify its characteristics. In theory, it is possible to directly modify the features of adversarial instances. However, in real-world scenarios, this approach is considered unsuitable for certain domains such as IDSs that analyze network traffic. This is mainly due to the fact that the feature extraction process (i.e., from raw traffic to feature space) is not a fully reversible process, unlike other domains such as computer vision. This means that features can be extracted, and modified but not easily reintroduced into network traffic due to the semantic links between features. Moreover, direct feature modification requires full knowledge of the feature extraction process used by the IDS in order to respect the syntactic or semantic constraints assigned to them. We can therefore deduce that working on the feature space is not very realistic. For this reason, recent studies [18–20, 24] propose to manipulate directly the raw network traffic so that it is not necessary to know the features used, nor to transform the feature values into traffic form. In this way, we can distinguish two manipulation spaces, the feature-based and the traffic-based.

1.3 Related work

Numerous research studies on the real impact of adversarial attacks have already been extensively conducted in the compute vision field, which has also urged researchers to study the issue in the cybersecurity field. Today, the number of papers on this topic is rapidly increasing and the actual impact of these attacks in a real-world scenario seems to be getting clearer. To help the community gain more insight into the topic, we analyze the important aspects of the feasibility of adversarial attacks by comparing the different research and reviews on the topic, especially those related to IDS.

In the review proposed by Reza *et al.* [26], the authors focus on giving a better understanding of adversarial examples in the computer vision domain. They propose an analysis of numerous attacks and defenses dedicated to this domain. Among these attacks, some are more realistic as they are directly applicable to a real-world scenario. However, this review does not provide any information about the implication of these attacks and defenses in the IDS domain. In addition, the review does not address the topic of domain constraints, nor the attack manipulation space.

Vitorino *et al.* [27] took an interesting approach in their paper to analyze, from the point of view of domain constraints, the suitability of adversarial attacks for the IDS domain. They showed that most of the state-of-the-art attacks, initially dedicated to computer vision, were not suitable for the IDS domain as they did not comply with these constraints. However, the paper does not address the manipulation space used, nor the problems related to the respect of semantic and syntactic constraints found in papers dealing with attacks against IDSs. In addition, defenses against adversarial examples in the IDS domain are also not addressed.

The study proposed by McCarthy *et al.* [28] proposes the analysis of several attacks and defenses in different domains of cybersecurity, namely intrusion detection, malware detection, and anomaly detection in industrial systems. They pointed out some constraints related to adversarial algorithms. Our review further elaborates on the manipulation space property, as well as a discussion of semantic and syntactic constraints that are not discussed in detail in their paper. In addition, our work surveys more recent papers.

The paper by Apruzzese *et al.* [12] provides interesting insights into the manip-

Knowledge restriction	Attack	Advantages	Disadvantages	Norm	Attack frequency	Attack objective	Suitable for IDS
White box	L-BFGS	Efficient in generating adversarial instances	highly computationally demanding	L_2	I	T	/
	FGSM	Calculation time efficiency	All features, including non-modifiable ones, are perturbed.	L_2 L_∞	O	T & NT	/
	PGD/BIM	More efficient than FGSM due to its iterative nature	When compared to FGSM, it is more computationally expensive	L_2 L_∞	I	T & NT	/
	JSMA	Successfully deceive a model by changing a few input features	More computationally demanding compared to FGSM	L_0	I	T	Could
	DeepFool	Produce significantly smaller perturbations than FGSM	Perturbations are sub-optimal More time consuming than FGSM and JSMA	L_2	I	NT	/
	C&W	Empirically shown to be more effective than other attacks Successfully circumvented many adversarial defenses	More computationally intensive than previous attacks	L_0 L_2 L_∞	I	T & NT	/
	EAD	Produce highly transferable instances Successfully bypassed defensive distillation	More time-consuming than other attacks such as FGSM	L_1	I	T & NT	/
Black box	Substitute Model	Successfully defeats gradient masking-based defenses Feasible against non-differentiable models	Not as effective as white box attacks	/	I	T & NT	yes
	ZOO	Its performance is comparable to that of C&W	Empirically slower than Substitute Model attack Needs a significant amount of queries to the target classifier	L_2	I	T & NT	/
	Boundary	Knowledge of the victim's model is not required Deliver comparable results to white box attacks	Require a substantial number of queries to find high quality adversarial examples	L_2	I	T & NT	/
	OPT	Requires fewer queries compared to ZOO and Boundary	Being a query-based attack, it can be easily detected	L_2 L_∞	I	T & NT	/
	GAN/WGAN	Can create samples that differ from those used in training	This attack can be computationally heavy and highly unstable	/	I	T & NT	yes

Table 1.2: List of well-known state-of-the-art evasion attacks (I: iterative, O: one-time, T: targeted, NT: non-targeted)

ulation space used in defining attacks as problem or feature-based. This work also provides an in-depth analysis of the different learning phases of the model by articulating the feasibility at all levels of the machine-learning pipeline. Our work differs by providing an analysis of more recent work on the topic and an explanation of each paper based on the domain constraints analysis. In addition, their work does not include an overview of possible defenses.

Martins *et al.* [29] provides a comprehensive overview of adversarial attacks against IDS and malware classifiers. They also describe the state of the art of defenses. However, this review does not include a discussion of the feasibility aspect of adversarial attacks and defenses. In our contribution, an analysis of the realistic aspect of the state-of-the-art defenses and attacks is introduced with an explanation of their feasibility.

1.4 Adversarial strategies

In this section, we present state-of-the-art adversarial attacks, classified into white-box and black-box algorithms. A list of these attacks can be found in Table 1.2.

1.4.1 White-Box algorithms

Limited-memory Broyden Fletcher Goldfarb Shanno (L-BFGS) The idea of this iterative attack is to produce an instance x_{adv} similar to the initial instance x under the distance L_2 but have x_{adv} classified as another target class using the L-BFGS box constraint. For this, Szegedy *et al.* [21] explain that it's possible to express the initial problem as a constrained minimization problem to generate targeted adversarial examples as illustrated in Eq. 1.5.

$$\begin{aligned} \text{Minimize: } & ||x - x_{adv}||_2 \\ \text{Such that: } & C(x_{adv}) = t \\ & x_{adv} \in [0,1]^n \end{aligned} \tag{1.5}$$

Since this problem is difficult to solve, they adapt it into an easier-to-handle variant, as shown in Eq. 1.6.

$$\begin{aligned} \text{Minimize: } & c.|x - x_{adv}| + J(x_{adv}, t) \\ \text{Such that: } & x_{adv} \in [0,1]^n \end{aligned} \tag{1.6}$$

where x is the input element, x_{adv} is the adversarial element, c is a positive constant, J is the loss function and t is the target label.

On the other hand, while L-BFGS is an effective attack, it can be time-consuming due to the use of the linear search method to find an optimal c.

Fast gradient sign method (FGSM) This one-step algorithm was developed by Goodfellow *et al.* in their 2014 paper [22]. The idea of FGSM is to generate perturbation using gradient ascent to maximize the loss function. FGSM can be used as a targeted or untargeted attack and originally runs under the L_∞ norm but is easily adaptable for the L_2 norm. FGSM is a very fast algorithm for generating adversarial instances even if the adversarial samples are not optimized because it does not minimize the generated perturbation. This algorithm is very efficient, in most cases, at creating adversarial perturbations in a time-efficient manner. It can

be defined by the following Eq. 1.7.

$$x_{adv} = x + \epsilon * sign(\nabla_x J(x, y)) \tag{1.7}$$

where ϵ is the variable allowing control of the amount of perturbation, y is the desired label, and the input x. The main disadvantage of this attack from a network traffic perspective is that all features are modified, making it less practical in real-life scenarios.

Basic Iterative Method/Projected Gradient Descent (BIM/PGD) This is an improvement of FGSM where the algorithm iteratively increases the amount of perturbation to cause misclassification. It is more efficient than the classical FGSM in terms of misclassification, but on the other hand, this attack takes more time to create adversarial examples. PGD is an algorithm proposed by Aleksander Madry *et al.* [30] and BIM is proposed by Alexey Kurakin *et al.* [31]. Both attacks are quite similar as they use, at each iteration, a projection function to project the adversarial examples into the $\epsilon - ball$ which can be L_2 or L_∞, as shown in Eq. 1.8.

$$x^t_{adv} = Proj[x^{t-1} + \epsilon * sign(\nabla_x J(x^{t-1}, y))] \tag{1.8}$$

where $x_{0adv} = 0$ and $Proj$ is the projection function.

The main difference between the BIM and PGD versions of the attack concerns the initialization of the attack. Indeed, BIM sets the value of the original point as the initialization point while PGD starts the attack at a random point using the L_∞ norm. Moreover, at each restart, a new random point is chosen. Since the results of these two attacks are generally quite similar, it is common to use only one of them when testing.

DeepFool This attack proposed by Moosavi-Dezfooli *et al.* [32] works as an untargeted attack and iteratively generates small perturbations to fool the classification. This algorithm uses the L_2 norm to generate these perturbations. To do so, this attack determines the nearest hyperplane for an input element and projects it beyond this hyperplane. This method is primarily based on the assumption that the model is completely linear. However, in most high-dimensional models, as in many

deep neural networks, this is rarely the case. To overcome this problem, a linear approximation is first performed. The main problem with this attack is the inability to introduce domain-specific constraints and the significantly longer time required to generate adversarial instances compared to the FGSM.

Jacobian-based Saliency Map Attack (JSMA) JSMA is an iterative and targeted algorithm proposed by Nicolas Papernot *et al.* [33] that uses a *saliency map* to tell which feature has the greatest impact on classification. This saliency map is based on a *jacobian matrix* which is a matrix containing the first-order partial derivatives as defined in Eq. 1.9

$$J_F(x) = \frac{\partial F(x)}{\partial x} = [\frac{\partial F_j(x)}{\partial x_i}]i \times j \tag{1.9}$$

This jacobian matrix, therefore, allows us to obtain the direction of sensitivity and, therefore know what input element influences the most desired output. This algorithm, based on the L_0 norm, has the advantage that it can generate adversarial samples using fewer features. It is therefore an interesting option for practical attacks against IDS.

Carlini and Wagner (C&W) Carlini and Wagner [34] proposed an optimization algorithm to generate adversarial examples under the L_0, L_2, and L_∞ norms. This attack is different from L-BFGS because it uses a different loss function to escape box constraints. They redefine the initial problem of adversarial examples previously defined in Eq. 1.1. This redefinition is given in Eq. 1.10

$$\begin{aligned} \text{Minimize: } & D(x, x+\delta) + c.f(x+\delta) \\ \text{Such that: } & x + \delta \in [0,1]^n \text{—— constraint 2} \end{aligned} \tag{1.10}$$

This attack is one of the most successful since it was able to break several defenses such as defensive distillation (see section 1.5.1). This attack can be used in a targeted and non-targeted version. Nevertheless, even if C&W is very efficient, it should also be noted that this algorithm takes significantly more time to generate adversarial instances.

Elastic-Net Attacks to Deep Neural Networks (EAD) This iterative algorithm proposed by Pin-Yu *et al.* [35] introduces the use of the L_1 norm to generate perturbations to create adversarial examples. The authors transformed the problem into an elastic network regularized optimization problem. The elastic network regularization takes advantage of Lasso (using the L_1 norm) and Ridge (using the L_2 norm) regularization. EAD uses an iterative attack under L_2 using a L_1 regularizer. The original Elastic-Net regularization defined in Eq. 1.11 is redefined for EAD as shown in Eq. 1.12.

$$\text{Minimize}_{z \in Z} \; f(z) + \lambda_1 ||z||_1 + \lambda_2 ||z||_2^2 \tag{1.11}$$

$$\begin{aligned} \text{Minimize}_x \; & c.f(x,t) + \beta ||x - x_0||_1 + ||x - x_0||_2^2 \\ & \text{Such that: } x \in [0,1]^n \end{aligned} \tag{1.12}$$

The experimental results [35] show that the attack is as effective as other state-of-the-art attacks. It is important to note that the results of this attack showed that it was the most effective in terms of transferability, which makes it interesting both for attackers using *substitute models* (black-box attack) and also for defenders using the adversarial training defense. In addition, the authors showed that this attack, like C&W, can break the defensive distillation defense. However, due to its optimization problem, it takes more time to execute than FGSM.

1.4.2 Black-Box algorithms

Zeroth-Order Optimization (ZOO) It's a black-box and score-based algorithm inspired by the C&W attack. As the name suggests, instead of using First-Order Optimization, it employs Zeroth-Order Optimization. It uses the logit values thanks to a zeroth order oracle to estimate the gradients. To estimate the gradients and Hessian, the authors [36] use the symmetric quotient difference.

To avoid detection by other defenses, Oracle queries must be reduced. To this end, ZOO employs three techniques, importance sampling, hierarchical attacks, and attack space reduction. The results of the experiments suggest that this attack is effective against ML models in black-box settings.

It is important to note that while this technique has comparable performance to C&W and yields a better attack success rate than substitute model attack, it is much more resource intensive than white-box algorithms, and even than the substitute model attack.

Boundary This iterative targeted/non-targeted decision-based attack created by Wieland Brendel *et al.* [37] is notably effective as it does not require gradient information and succeeds in defeating many existing defenses like defensive distillation and gradient masking defenses. Moreover, it is more realistic with respect to the other attacks as it doesn't rely on probabilities, but rather on the decision, which is what Machine Learning APIs typically provide. According to the authors, despite being a black-box attack, the Boundary attack produces a similar misclassification efficiency as other white-box attacks such as FGSM, C&W, and DeepFool.

Boundary uses a relatively simple and flexible algorithm. This attack uses a simple rejection sampling algorithm to track the decision boundary from the adversarial classification region to the non-adversarial region. The main drawback of this attack is that it uses an excessive number of iterations to find adversarial examples due to its brute-force nature.

OPT The OPT attack, proposed by Minhao Cheng *et al.* [38], is an iterative decision-based black-box attack that can be targeted or untargeted. Being a decision-based attack means that it just needs the decisions rather than logits or probabilities. This optimization-based attack uses the Randomized Gradient-Free (RGF) method to estimate the gradient at each iteration rather than using the zeroth-order coordinate descent method, which provides lower performance. The RGF method is defined in Eq. 1.13 to estimate the gradient:

$$\hat{g} = \frac{g(\theta + \beta u) - g(\theta)}{\beta}.u \tag{1.13}$$

where g is the search direction, $g(\theta)$ is the distance from x_0 to the nearest adversarial example along the direction θ, $\beta > 0$ is a parameter and u is a random Gaussian vector.

This attack uses the L_2 and L_∞ norms to determine the perturbation rate to be

applied and the binary search to evaluate the objective function. The results showed that the OPT attack was more efficient in terms of the number of queries required than the boundary attack, a similar attack in that it also uses only the model decision to be able to generate adversarial perturbations. In terms of performance, OPT was shown to be as efficient as many other state-of-the-art algorithms.

Despite this, it requires performing a large number of queries, which can be detected by the victim's model if defense mechanisms are in place.

Substitute Model attack This method, used to perform adversarial attacks in a black box setting, was designed by Papernot *et al.* [11]. It allows extracting the architecture of the model, the decision boundaries, and its functionalities. The goal is to try to mimic the original model using several queries to obtain $y = F^s(x)$, i.e., the given prediction of the original model must be equal to the prediction of the copied model. Once the model has similar behavior, state-of-the-art white-box attacks are used to generate adversarial examples. Using the transferability property, which is intrinsic to machine learning models, the original model can be fooled. This type of attack is less effective than white-box attacks but since it does not rely on gradient information, this attack is indeed feasible against non-differentiable models. In addition, it has been shown that the Substitution Model attack also defeats defenses based on gradient masking and defensive distillation.

(Wasserstein) Generative Adversarial Network (GAN/WGAN) GAN is an algorithmic architecture created by Goodfellow *et al.* [39] in 2014 and used to generate synthetic instances that resemble the original real instances. GAN uses two neural networks called *discriminator* and *generator*, both of which play a zero-sum adversarial game. The generator will try to create fake instances using a random normal distribution to fool the discriminator. The discriminator's goal is not to be fooled and to try to identify the false instances by learning from real instances contained in a dataset. As it goes along, the generator will try to learn to create more real instances.

WGAN [40] uses a different method than GAN to compute the probability distance between the two distributions. Instead of using the Jensen-Shannon divergence [41], which is based on the Kullback–Leibler divergence [42], it uses the

Defenses	Advantages	Disadvantages	Type of defense	Efficiency	Applicable for IDS
Adversarial Training	Effective against all attacks Easy to implement	Not effective against attacks different from those used during the training phase and there is a trade-off between robustness and accuracy	Proactive	Could	Yes
Adversarial Detection	Keeps the accuracy of the model	Some detection methods are proven ineffective	Reactive	Could	Yes
Obfuscated Gradients	Effective against gradient-based attacks	Different method to bypass this defense and not effective against transferability	Proactive	/	Yes
Defensive Distillation	Distilled model is less sensitive to small perturbations	Shown to be ineffective	Proactive	/	Yes
Feature Squeezing	Good performance for image classification domain	Not suitable for tabular data	Proactive	/	/
Ensemble Defense	Combines several defense methods	Not efficient if the defenses used are broken	Reactive	/	Could
Feature Removal	Decreases the attack surface	Decreases the general performance of classification	Proactive	Yes	Yes
Adversarial Query Detection	Don't modify the performance of the model	Applicable only for black-box attacks	Reactive	/	Yes

Table 1.3: List of well-known state-of-the-art evasion defenses

Earth-Mover's distance [43]. The main advantages of WGAN are that it solves the vanishing gradient and mode collapse problems through more stable training. However, even if this mitigates the stability problem, the attack remains unpredictable and therefore unstable.

1.5 Defense strategies

The following defenses are designed to mitigate the effect of adversarial examples, such as those generated by the adversarial attacks discussed in Section 1.4. One could divide defenses into two different types: **proactive**, where the idea is to prevent the model to be fooled by adversarial examples, and **reactive**, where the defense tries to detect adversarial examples during attacks. A list of the well-known defenses is in Table 1.3.

1.5.1 Proactive defenses

Adversarial Training The purpose of this defense proposed by Ian J. Goodfellow *et al.* [22] is to strengthen the model against adversarial attacks by taking them into account during the learning phase. This defense can be seen in two ways. One can either provide the adversarial examples directly to the model with the training data or incorporate them into the loss function of the model which acts as a regularizer. This kind of defense is easy to implement and can be used very well in the IDS domain.

A variant, called Ensemble Adversarial Training [44] allows to improve robustness against attacks. The effectiveness of this defense can be improved by taking into account adversarial examples generated with different algorithms rather than using only one. If the model is trained with adversarial examples generated based on some adversarial attacks, an attacker could use other attacks to fool the classifier by exploiting the lack of generalization.

Although the models become more robust to adversarial examples, they are not completely immune because some adversarial examples may go undetected. Moreover, this defense is limited by a trade-off between robustness and accuracy, as the more, the model is trained with adversarial examples, the more its overall performance decreases. Athalye *et al.* [45] also showed that, if the model is trained with adversarial examples generated using the L_∞ norm, the model is less robust as compared to training based on other norms (L_0, L_1 and L_2)

Obfuscated Gradients This technique is based on a gradient masking method so as to disrupt the descent of the gradient and, in this way, prevent gradient-based attacks from being able to successfully exploit the gradient by trying to make the model non-differentiable.

Gradient masking was broken by several attacks. One of them is in a white-box setting by using a random step and then switching to a gradient-based algorithm like FGSM for example. It was also broken in the black-box setting via transferability, which ensures the effectiveness of adversarial examples against other models than the one on which they were generated. Papernot *et al.* [46] show in particular that black box attacks are more effective than white box attacks when gradient masking defense is used. Furthermore, Athaye *et al.* [45] have shown how to bypass three types of obfuscated gradients, namely: shattering gradients, stochastic gradients, and vanishing/exploding gradients.

It can be noted that this defense could be used in the IDS domain but may not be as effective since, in theory, an attacker has limited knowledge of the defender's model, which limits the possibility of using white box attacks directly.

Defensive Distillation Initially used to reduce the dimensionality of DNN, a defense based on distillation is proposed by Papernot *et al.* [33] defensive distillation

aims to smooth the decision surface of the model. The distillation method uses two neural network models. An initial network, taking as input the training data and the corresponding labels. The model provides predictions in a probability vector and transfers this knowledge to the second network, called the distilled network. This network, therefore, takes the same training data as the initial network, but the corresponding labels are taken from the probability vector of the initial network, then new predictions are made. The authors showed that defensive distillation is less sensitive to small perturbations.

This defense could be used in the IDS domain, however, Carlini and Wagner [47] have shown that this defense is broken by their attack. Therefore, it is not wise to use defenses that have already been shown to be broken when protecting a model.

Feature Squeezing This defense technique, proposed by Weilin Xu *et al.* [48], compresses the features of the instances and classifies them. Then a comparison is made between this classification and the classification of the original samples. If the results are different, then the instance is considered adversarial. Of the compression methods used by the author, such as bit depth compression, median smoothing, or non-local means, none consistently gives the best results, all need to be evaluated collectively as performance differs depending on the dataset used.

This defense is not suitable for the IDS use case because network traffic is often represented in tabular form and these compression techniques result in significant information loss for the underlying data.

Ensemble Defense This technique is expected to be effective by assuming that several different defense techniques improve the robustness of the model. Such a defense can be either proactive or reactive, or a mixture of both, in case the different defense mechanisms used are both reactive and proactive. It can therefore be used in the IDS domain and is potentially effective against various types of attacks.

This technique is ineffective because an attacker can exploit any of the defenses' flaws to bypass them all. In addition, Warren He *et al.* [49] demonstrated that employing multiple weak defenses does not result in a stronger defense.

Feature Removal This defense consists of identifying the most vulnerable features and removing them from the data used to train the model. These vulnerabilities often come from the complexity of the model with high dimensions. This defense will therefore reduce the complexity of learning the model and remove some dimensions that are too vulnerable to escape the attack. The removal of features will reduce the attack surface by reducing the possible vectors that can be perturbed.

However, the work of Apruzzese *et al.* [50] shows that feature removal decreases the performance of an IDS. It results in a loss of precision that increases the number of false positives.

1.5.2 Reactive defenses

Adversarial Detection This attack mechanism uses different methods to detect adversarial examples, based on statistical tools such as principal component analysis (PCA), distributions, and normalization.

One example of this method is the defense proposed by Feinman *et al.* [51] which is based on two techniques: density estimates and Bayesian uncertainty estimates. The general idea of the first method is to check whether the density estimates of the last hidden layer for an input instance are significantly different from those associated with the training set containing the benign examples and, if so, the instance will be considered adversarial. Density estimates are made in the feature space of the last hidden layer because it is considered more linear than that of the input. Bayesian uncertainty estimates can be used to overcome situations where density estimates cannot detect adversarial examples. This method allows detection in low-confidence regions in the input space.

The main advantage of this reactive defense is that it does not change the initial accuracy of the model. It could also be used in the IDS domain because it has no particular restrictions. However, it has two main problems: The first one is that it provides many false positives, which makes it less effective. The second problem is that most of these detection methods have been proven to be broken by Carlini and Wagner [52]. Tianyu *et al.* [53] proposed a more robust alternative to detect adversarial examples using kernel density estimates and the reverse cross-entropy training procedure.

Adversarial Query Detection Introduced in the Titi-taka framework proposed by Zhang *et al.* [54], this defense consists of detecting the number of abnormal queries that signal that an attack is being conducted. This defense reduces the number of possible queries sent to the model, making it more difficult to exploit for attacks using a large number of queries, while maintaining the initial accuracy. The main drawback is that this defense is only effective against black-box attacks that use large numbers of queries.

1.6 Adversarial attacks against IDS

In recent years, many researchers have been interested in the implications of adversarial learning in the context of IDS by proposing several studies. We have selected several of them for their contributory aspect, and more precisely their contribution in terms of feasibility, a very important topic for IDS. These contributions are all listed in Table 1.4.

In 2017, Maria Rigaki *et al.* [55] demonstrated in their work that adversarial examples can be generated by adversarial algorithms to fool an IDS using a DNN trained on the NSL-KDD dataset. They showed that not all algorithms are suitable for fooling an IDS and pointed out the fact that FGSM is incompatible with this goal, but JSMA may be suitable. Furthermore, they showed that adversarial examples are capable of transferring to several machine learning models such as Decision Trees, Random Forests, Linear SVM, and Ensemble Voting. In addition, they showed that creating a feature-based adversarial instance requires knowing the mapping between features and network traffic and how the data is preprocessed because, unlike images, features extracted from network traffic are highly correlated.

Zilong *et al.* [56] proposed to study the effectiveness of adversarial examples with a WGAN using the NSL-KDD dataset. The performances of several classifiers were studied, namely decision tree, Random Forest, SVM, MLP, Naive Bayes, logistic regression, and KNN. The results showed that adversarial examples can fool all trained classifiers. They also showed that the attack remains effective even when using a limited feature space.

Warzyński and Kołaczek [57] proposed a study on using FGSM to generate adver-

Paper	Year	Adversarial attack	Target ML model	Datasets	Metrics	Defense used	Network type	Manipulation space	Domain constraints
[55]	2017	FGSM JSMA	Decision Tree, Random Forest Linear SVM, Voting Ensemble DNN	NSL-KDD	Accuracy F1-score AUC	/	Classical	Features	/
[56]	2018	WGAN	Decision Tree, Random Forest SVM, DNN, Naive Bayes Logistic Regression, KNN	NSL-KDD	Detection Rate	/	Classical	Features	/
[57]	2018	FGSM	DNN	NSL-KDD	Accuracy FPR, FNR Precision	/	Classical	Features	/
[15]	2018	FGSM, JSMA DeepFool C&W	DNN	NSL-KDD	Accuracy, F1-score FPR, Precision Recall, AUC	/	Classical	Features	/
[16]	2018	WGAN ZOO Substitute Model	Random Forest SVM DNN Naive Bayes	NSL-KDD	Accuracy F1-score, FPR Precision, Recall	/	Classical	Features	/
[58]	2018	Manual perturbations	Random Forest	CTU-13	Accuracy F1-score Precision, Recall	/	Classical	Features	/
[17]	2019	FGSM, JSMA DeepFool C&W	Decision Tree, Random Forest Naive Bayes, SVM, DNN Denoising Autoencoder	NSL-KDD CICIDS-2017	AUC	Adversarial Training	Classical	Features	/
[59]	2019	FGSM, JSMA C&W EAD	KitNET (Kitsune)	Kitsune	Success Rate AUC	/	Classical IoT	Features	/
[60]	2019	FGSM BIM PGD	DNN	BOT-IoT	Accuracy	Feature Normalization	IoT	Features	/
[50]	2019	Manual perturbations modifying up to 4 features	Decision Tree, Random Forest, SVM DNN, Naive Bayes, Linear Regression, KNN ExtraTrees, AdaBoostBagging, Gradient Boosting, SGD Linear Classifier	CTU-13 CICIDS-2017 CICIDS-2018 UNB-CA Botnet	F1-score Precision, Recall Attack Severity	Feature Removal	Classical	Features	/
[24]	2019	Manual perturbations for packet-based IDS A proposed optimization attack for flow-based IDS	KitNET (Kitsune) DAGMM BiGAN	CICIDS-2017	TPR FPR	/	Classical	Traffic	Yes
[25]	2019	Manual perturbations based on modification of the payload size, packet rate and bidirectional traffic	Random Forest SVM, KNN Logistic Regression	CICIDS-2017	Accuracy, F1-score Success Rate TPR, FPR	/	Classical	Traffic	Yes
[61]	2020	Adapted JSMA (AJSMA) Histogram Sketch Generation (HSG)	Decision Tree, SVM, DNN Logistic Regression, KNN	NSL-KDD UNSW-NB15	Success Rate	/	Classical	Features	/
[20]	2020	GAN OPT	Random Forest, OCSVM , DNN, Stacking Model Naive Bayes	Simulation of real ICS environment. Real traffic captured on the implemented infrastructure	F1-score Precision Recall Success Rate	Adversarial Training	ICS	Traffic	Yes
[19]	2020	Adversarial Pad (AdvPad) Adversarial Payload (AdvPay) Adversarial Burst (AdvBurst)	CNN	ISCXVPN2016	F1-score Precision Recall	/	Classical	Traffic	Yes
[18]	2021	Adversarial Features Generation with GAN + Malicious Traffic Mutation with PSO	Decision Tree, SVM KitNET, DNN Logistic Regression Isolation Forest	CICIDS-2017	(evasive metrics) MER, DER, PDR, MMR (performance metrics) F1-score, Precision, Recall	Adversarial Training Feature Removal Adversarial Feature Reduction	Classical IoT	Traffic	Yes
[14]	2022	NES, Boundary, Pointwise HopSkipJumpAttack Opt-Attack	MLP, CNN C-LSTM Stacking Model	CICIDS-2017 CICIDS-2018	Accuracy, F1-score Precision, Recall Success Rate	Tiki-Taka (Adversarial Training, Adversarial Query Detection and Ensemble Voting)	Classical	Features	/
[23]	2022	FGSM, BIM JSMA, DeepFool, C&W	MLP	NSL-KDD	Detection Rate Lp norms mean and max	/	Classical	Features	/
[13]	2022	FGSM, PGD DeepFool, C&W	KitNET (Kitsune) DNN	CICIDS-2017 NSL-KDD	Detection Rate Precision, Recall, F-score	Adversarial detection based on Transfer learning	Classical	Features	/

Table 1.4: List of recent evasion attacks against anomaly-based IDS

sarial examples to fool a neural network-based classifier trained with the NSL-KDD dataset. The results showed that this attack is effective. It may be noted that this study does not address the feasibility of adversarial attacks in a realistic scenario and is limited to the study of a particular dataset and a particular attack.

Zheng Wang [15] provides an in-depth analysis of the NSL-KDD dataset by investigating feature importance when generating adversarial examples against a multilayer perceptron (MLP) based IDS. He showed that the feasibility of adversarial attacks against IDSs is different from that of image classifiers by illustrating that not all adversarial algorithms are suitable for creating adversarial attacks against IDSs. Among these algorithms, JSMA seems to be the most suitable as it does not modify all features to create adversarial examples but only those it considers most important. This is particularly relevant since adversaries are usually limited in their ability to manipulate features due to restricted access or the complexity of manipulating them all at once. Feature importance shows that some features are more vulnerable than others in that they are more often selected by the algorithm during adversarial examples generation.

Yang *et al.* [16] proposed a more realistic approach to the problem by using three black-box attacks that assume no knowledge of the target model information. These three attacks are WGAN, ZOO, and Substitute Model. To analyze their performance, they took 5 classifiers, namely Random Forest, SVM, MLP, and Naive Bayes trained with the NSL-KDD dataset. The results showed that ZOO was the most efficient, the Substitute Model was the least efficient while WGAN provided good performance but was unstable due to its intrinsic properties. However, the paper does not discuss the feasibility of these attacks in real-world settings besides the black-box perspective.

Instead of using state-of-the-art attacks, Apruzzese *et al.* [58] proposed an attack that iteratively produces manually defined perturbations. They studied the performance of this attack on a Random Forest classifier trained on the CTU-13 dataset which is based on a collection of Botnet attacks. This attack follows a fairly simple strategy, it clusters specific features and applies an iterative perturbation. The features are not chosen trivially, they are the ones that are easiest to manipulate, namely time, packet size, and the number of packets. The results showed that

changing only a few features can lead to a decrease in classifier performance. From a feasibility point of view, this work is interesting because the modified features are at most four, and chosen in advance, which can be adjusted to modify only those features we have access to.

To generate adversarial examples, Martins *et al.* [17] chose to work on the NSL-KDD dataset and a more realistic and recent dataset, namely CIC-IDS 2017. They also used adversarial training, first described in the image classification literature, to see if it could be applied to the IDS domain. The results showed that among the attacks used, JSMA is the least effective but disrupts the fewest features. They also showed that Adversarial Training improves the overall robustness of all classifiers, namely Decision Tree, Random Forest, Naive Bayes, SVM, DNN, and Denoising Autoencoder. Nevertheless, we can note that the feasibility of adversarial attacks has not been addressed in this work, except for the use of a more realistic dataset.

Joseph *et al.* [59] used Kitsune's classifier called KitNET, and its dataset, to evaluate its robustness to adversarial examples generated based on four attacks (FGSM, JSMA, C&W, EAD) using different norms (L_0, L_1, L_2, L_∞). In a white box setting, the results showed that the classifier was vulnerable to all four attacks.

The study provided by Olakunle *et al.* [60] analyzed the impact of adversarial examples generated with FGSM, BIM, and PGD against two DNNs trained with the Bot-IoT dataset containing network attacks such as DOS or DDOS. The results showed that these adversarial attacks performed well in the IoT domain. In addition, they proposed feature normalization as a defense mechanism. The results showed that this defense was not effective as it increased the accuracy of the classifier however it also made the model more vulnerable to adversarial examples.

According to Apruzzese *et al.* [50] on evaluating the effectiveness of adversarial attacks against botnet detectors, their results show that it is possible to fool this type of NIDS detector based on machine learning algorithms. They found that all the machine learning algorithms studied in this paper, namely Random Forest, Decision Trees, AdaBoost, Multi-Layer Perceptron, K-Nearest Neighbor, Gradient Boosting, Linear Regression, Support Vector Machines, Naive Bayes, ExtraTrees, Bagging, and Stochastic Gradient Descent Linear Classifier are susceptible to be fooled. In this study, the experiments are conducted from a more realistic perspective

by taking into account some important domain constraints and assuming the gray box parameters, and using known realistic datasets containing botnet attacks to train their models. These datasets are as follows: CTU-13, IDS2017, CIC-IDS2018, and UNB-CA Botnet. To generate adversarial examples, the authors manually add small perturbations to a maximum of 4 features of each malicious instance keeping a realistic perspective. These features are duration, sent bytes, received bytes, and exchanged packets. Furthermore, they showed that using the defense called "*feature removal*" (see Section 1.5.1) does not guarantee robust protection for botnet detectors.

Hashemi *et al.* [24] propose a bottom-up approach by first analyzing the characteristics of the IDS datasets to understand their domain constraints. Once these constraints were identified, they showed that it is possible to fool different IDS models (Kitsune, DAGMM, and BiGAN) trained on the Kitsune and CICIDS2017 datasets, respectively, under these domain constraints for both packet-based and flow-based IDSs. To approach the problem more realistically, they proposed two algorithms for each of these network traffic types. For packets, the algorithm is divided into three parts: one function that generates delays between packets, another one that splits packets to have more packets, and the last one used to generate new packets. For flows, the algorithm uses a system of groups, only one of which can be modified and on which the attacker can apply perturbations. Recent work by Teuffenbach *et al.* [25] building on this work, also uses this grouping method to modify only relevant features. However, these groupings are slightly different from those proposed by Hashemi. Their results also showed that their method, involving domain constraints directly in their optimization problem, was effective in fooling the models (DNN, DBN, and AE) trained on CIC-IDS2017 and NSL-KDD.

In their paper, Aiken *et al.* [62] investigated the effectiveness of a novel adversarial example generation method focused on a SYN Flood DDoS attack. The results showed that the proposed algorithm is effective in fooling the classifiers (Random Forest, SVM, Logistic Regression, and KNN) trained on the SYN Flood attack present in the CIC-IDS 2017 dataset. The accuracy of the model fell to 0% using the proposed algorithm. However, some characteristics of the instances are manipulated when they are not supposed to be since they are not easily modified in reality, such

as the traffic from the victim.

Sheatsley *et al.* [61] study showed that, even when several NIDS-related domain constraints are considered, limiting the number of features that can be modified, it is possible to create realistic adversarial examples capable of fooling attack detectors using the AJSMA (Adapted JSMA) and HSG (Histogram Sketch Generation) adversarial algorithms. The experiments were conducted on the NSL-KDD and UNSW-NB15 datasets. They showed that domains with more restrictive constraints, such as NIDS, are no more robust than those with fewer constraints, such as image recognition. They also showed that these attacks were effective because of their transferability.

Jiming Chen *et al.* [20] studied the impact of adversarial examples on the domain of Industrial Control Systems (ICS). They took a more realistic approach by limiting their knowledge of the models used by the defender. They reproduced an ICS system to create a realistic environment and trained their MLP model directly on the extracted traffic. They then used two attack algorithms, GAN and OPT, to produce adversarial instances while taking into account domain constraints such as constraints related to the protocols used during the attacks. Their results showed that the ICS domain was also vulnerable to adversarial examples. In this study, several models were tested, namely, Random Forest, OCSVM (One-Class SVM), DNN, Stacking Model, and Naive Bayes. They proposed to use adversarial training and they found that this defense improved the robustness of the models studied.

In their paper, Sadeghzadeh *et al.* [19] proposed a problem-based approach as opposed to a feature-based approach [63]. The authors used three new attacks to manipulate network traffic called Adversarial Pad (AdvPad) which adds the perturbation to the packet, Adversarial Payload (AdvPay) which adds perturbation to the payload and Adversarial Burst (AdvBurst) which adds newly crafted packets. They propose to manipulate traffic concerning different types of services such as VoIP, mail protocols, file transfer, or P2P present in the ISCXVPN2016 dataset. The results showed that the classifier used (CNN) decreased its robustness due to the use of the three attacks.

Han *et al.* [18] proposed a novel attack using a GAN and a Particle Swarm Optimization (PSO) technique to directly manipulate the traffic under a black-

Dataset	Year of publication	Description	Reliability
NSL-KDD	2009	Due to the problems related to attacks inside the KDDCUP'99 dataset, this dataset has been proposed with more up-to-date attacks.	It is old and not adapted to train classifiers today because it contains old attacks not used today
CTU-13	2013	It's a dataset that contains different botnet and benign network traffic provided by different datasets.	This dataset, almost 10 years old, is less interesting than some more recent and semantically correct ones.
UNB-CA Botnet	2014	This dataset was developed to provide a focus on the realism, generality, and semantics of botnet attacks.	Since it is synthetically constructed and is an aggregation of several datasets with Botnet attacks (real or not), it has some bias, although it provides a good training ground given its many attack instances.
UNSW-NB15	2015	This synthetic dataset, created by the Cyber Range Lab of UNSW Canberra, provides several types of network attacks such as DoS, Exploits, Reconnaissance or Backdoors. Its diversity of attacks makes it an interesting data source.	from a reliability point of view, it contains biases due to its synthetic profile (as for most public datasets) and is relatively old, thus less interesting than more recent ones with similar attacks.
ISCXVPN2016	2016	It's a realistic dataset that provides different types of traffic such as VoIP, file transfers, P2P, and email protocol usage.	It is reliable because all the traffic generated is real. It also provides traffic from different services with the use of VPNs in some cases.
CIC-IDS2017	2017	Implement different network attacks like Bruteforce, DoS, Web, Botnet, Infiltration and DDOS against machines in a simulated enterprise architecture.	This dataset respect the 11 minimum requirements to have a reliable dataset [64]. Semantic logic is respected, and the list of attacks is relatively useful for classifiers today. Some semantics flaws in attacks are solved in an update done by [65]
CIC-IDS2018	2018	Implement different network attacks like Bruteforce, DoS, Web, Botnet, Infiltration, and DDOS against machines in an AWS network.	More recent than CIC-IDS2017 and follows the same requirements.
Kitsune	2018	The authors of Kitsune used a dataset specifically for the development of their IDS. This dataset contains different types of attacks against a surveillance system. These include reconnaissance attacks (OS Scan, Fuzzing), MitM (ARP cache poisoning), DoS (SYN DoS), and Botnet (Mirai).	This dataset is based on an implementation that mixes traffic from a classical network and an IoT network. The attacks are recent and therefore relevant.
BOT-IoT	2019	Designed by the Cyber Range Lab of UNSW Canberra. It contains different botnet attacks like DOS, DDOS, and Reconnaissance	This dataset is more recent than previous ones and provide good reliability in term updated attacks.

Table 1.5: List of the most popular datasets for training state-of-the-art IDS

box assumption. This approach is more realistic because the feature space is not easily reversible in the IDS domain, which means that the change in feature value cannot be transferred directly into the network traffic due to the numerous domain constraints. The attack process is divided into two parts, the GAN is used to generate adversarial features first, and then the PSO is used to add mutations to the malicious traffic. To evaluate the effectiveness of their attack, the authors used three new metrics to test the evasion effectiveness, namely the Detection Evasion Rate (DER), the Malicious Traffic Evasion Rate (MER), and the Probability Drop Rate (PDR) of the malicious traffic. They also proposed a new metric to provide an interpretability indicator called Malicious features Mimicry Rate (MMR) which provides a measure of how far the adversarial features are from the malicious features during mutation. The results showed that the attack was able to fool packet-based IDSs trained with the Kitsune dataset, as well as flow-based IDSs trained with the CIC-IDS2017 dataset. However, the effectiveness of the attack varies depending on the knowledge of the extracted features. If the substitute model does not know any of the features extracted by the extractor, it will still be able to generate adversarial traffic capable of fooling the classifiers by randomly choosing less effective features. Despite the black-box assumption, the authors assumed that the extractor used by

the IDS is known, which is not always the case in real settings.

Zhang *et al.* [14] proposed a new version of their original paper [54]. Their goal was to evaluate their framework's performance in terms of improving IDS protection against evasion attacks subject to limited defense knowledge. In particular, they used multiple decision-based black-box algorithms to provide a more realistic representation of the problem. The results showed that the classifiers, using MLP, CNN, and C-LSTM, trained on the CIC-IDS2018 dataset were all vulnerable to the used black-box algorithms, namely NES, Boundary, Pointwise, HopSkipJumpAttack, and Opt-Attack. To improve the robustness of the classifiers, the authors propose to use their Tiki-Taka framework combining several defenses, Adversarial Training, Ensemble Voting, and Adversarial Query Detection. Thus, the authors combined two proactive defenses and a reactive one respectively. The results indicate that this combination of defenses is effective against the attacks studied on the CIC-IDS2017 and CIC-IDS-2018 datasets.

In their paper, Mohamed Amine Merzouk *et al.* [23] provide an in-depth analysis of the feasibility of state-of-the-art attacks against an IDS trained with NSL-KDD. The adversarial algorithms studied are FGSM, BIM, DeepFool, C&W, and JSMA. They showed that these adversarial algorithms produce invalid Adversarial Examples (AEs) if applied directly without taking domain constraints. For example, some of the generated values were negative or out of bounds, exceeding their feasible limit. These AEs must meet certain criteria to be valid. In particular, they show four constraints, namely out-of-range values, non-binary values, membership in multiple categories, and semantic links.

Debicha *et al.* [13] proposed an adversarial detector design based on transfer learning. They evaluated the effectiveness of using multiple strategically placed adversarial detectors versus a single adversarial detector for intrusion detection systems. Their experiments were conducted on two IDS architectures: a serial architecture and a Kitsune-inspired parallel architecture. They chose four evasion attacks to generate adversarial traffic, namely FGSM, PGD, DeepFool, and C&W. Although the attacks are feature-based and not traffic-based, the author has taken into account the domain constraints to make them more feasible. Their defense is based on the implementation of multiple adversarial detectors, each receiving a subset of

the information passed by the IDS and using a suitable fusion rule to combine their respective decisions. Using this defense, they were able to improve the detection rate over adversarial training.

1.7 Discussion

This literature review has demonstrated that the feasibility of existing evasion attacks on intrusion detection systems in real network settings is restricted due to certain limitations. The main limitations are related to the feature-space manipulation and the assumptions about the attacker's knowledge of the IDS. Most of the reviewed papers do not consider the realistic aspect or only in a limited way. As a result, future studies should take into account the following aspects: first, the adversarial traffic generated should be valid, and problem-space projection should be performed to ensure that the reverse feature-mapping is feasible. Second, the attacker's knowledge should be limited, and assumptions of full knowledge of the IDS should be avoided. Finally, black box attacks can be easily detected by simple defenses, and querying the IDS like an oracle is not feasible. Therefore, future studies should focus on developing new evasion attacks that take into account these limitations and provide more realistic scenarios for evaluating the effectiveness of intrusion detection systems.

1.8 Summary

Current research on the impact of using evasion attacks to bypass machine learning-based NIDS has shown that a slight perturbation can allow the attacker to circumvent detection. This of course raises a security concern, as the use of machine learning models is becoming more prevalent in the cybersecurity field. However, while it is theoretically possible to exploit these models, their exploitation is a bit different in a real-world setting. In this chapter, we have reviewed the most recent attacks based on two possible adversarial strategies, namely the white-box and the black-box settings. We then explored some popular defense mechanisms. For each of the attacks and defenses, we elaborated on their suitability in the IDS domain. Finally, a set of related research papers are highlighted to clarify the feasibility of

adversarial attacks in a more realistic context in the cybersecurity domain, specifically in the IDS domain. Concerning feasibility, we have provided several criticisms regarding recently published work by identifying their manipulation space. Thus, future research should focus on manipulating the traffic space by limiting the attacker's knowledge. We believe that this review has highlighted various points that may have been overlooked in some previous research, and that it will allow future research in this area to better address the various realistic constraints.

Intrusion Detection Systems are now an essential component of computer and network security. Despite massive research efforts in the field, dealing with source reliability remains an open issue. More information on how to address this issue can be found in the following chapter.

Chapter 2

Efficient Intrusion Detection Using Evidence Theory

2.1 Introduction

As computer network usage grows rapidly along with the significant increase in the number of applications running on it, the importance of network security is increasing. As dedicated tools designed to identify anomalies and attacks on the network, Intrusion Detection Systems (IDS) are becoming more valuable. Detection techniques based on anomalies and misuse have long been the principal subject of research in the field of intrusion detection [66].

Misuse-based IDSs operate quite similarly to most antivirus systems. Maintaining a signature database that could identify specific types of attacks and checking all incoming traffic against these signatures. Overall, this approach performs well, although it does not work properly when dealing with new attacks, or those that were specifically crafted to mismatch existing signatures.

On the other hand, anomaly-based IDSs operate generally on a baseline of normal activities and network traffic. This allows them to assess the current state of network traffic against this baseline so that abnormal patterns can be identified. While such an approach could be quite effective in detecting new attacks or those that have been intentionally crafted to evade IDSs, it can also result in a higher number of false positives compared to misuse-based IDSs.

Dempster-Shafer Theory (DST), also known as evidence theory [67] is one of the most versatile mathematical frameworks, extending Bayesian theory by (i) providing each source with the ability to integrate information at various scales of detail, thus addressing uncertainty; and (ii) offering a robust decision-making tool to make a consensus-based decision. This theory was later widely applied in several domains [68] [69] [70]. Regardless of this popularity, mass function generation and source

reliability estimation remain an ongoing challenge.

Probabilistic frameworks for mass generation take advantage of the extensive research literature of the traditional probabilistic classifiers. These approaches usually represent the information associated with each attribute through Probability Density Functions (PDF), typically Gaussian [71] [72]. Such densities are then transformed into beliefs that can subsequently be merged to form a joint decision. One can attribute masses to the compound hypotheses by subtracting the mass values related to the simple hypotheses involved [71] or by mixing the distributions associated with these hypotheses [72]. It should be noted that for most applications, Gaussian densities have been widely assumed due to their simplicity. Nevertheless, in the case where this assumption fails, the decision-making performance may be influenced considerably. More sophisticated approaches can be used to surmount this limitation by transforming data attributes into an equivalent normal space [73].

This work offers a more effective way to overcome this disadvantage by constructing PDFs that are better suited to the original data histograms instead of projecting them into a new Gaussian-like space. On a more explicit level, a kernel smoothing estimation [74] is used on the training data to derive an approximate PDF for each data attribute and each simple hypothesis. These PDFs may be of any shape. Notably, they might be non-Gaussian. During the classification phase, a given datum is associated with a set of masses that are generated in an elaborated way from the aforementioned densities. Using the proposed contextual discounting method, mass functions are then weakened differently depending on the ability of each source to discriminate between classes. Mass functions of the different sources are then merged to have a consensual mass function using a suitable fusion rule. A final decision is then deduced using the so-called "pignistic transform" [75].

The rest of this work is organized as follows: Section 2.2 recalls the theoretical tools used in the proposed approach. Section 2.3 describes the NSL-KDD dataset. A description of Boosted PR-DS architecture is introduced in Section 2.4. Section 2.5 discusses the experimental results by comparing them with those of some previous studies using the NSL-KDD dataset. Final remarks and further suggestions for improvement are given in Section 2.6.

2.2 Related background

We succinctly outline some fundamentals of Dempster-Shafer theory, Parzen-Rosenblatt density estimation and contextual discounting.

2.2.1 Dempster-Shafer theory

Suppose that $\Omega = \{\omega_1, ..., \omega_K\}$, and $\mathcal{P}(\Omega) = \{A_1, ..., A_Q\}$ is its power set, where $Q = 2^K$. A defined mass function M ranging from $\mathcal{P}(\Omega)$ to $[0, 1]$ is named a "basic belief assignment" (bba) if $M(\emptyset) = 0$ and $\sum_{A \in \mathcal{P}(\Omega)} M(A) = 1$. A bba M therefore defines a "plausibility" function Pl ranging from $\mathcal{P}(\Omega)$ to $[0, 1]$ by $Pl(A) = \sum_{A \cap B \neq \emptyset} M(B)$, and a "credibility" function Cr ranging from $\mathcal{P}(\Omega)$ to $[0, 1]$ by $Cr(A) = \sum_{B \subset A} M(B)$. In addition, the two functions mentioned above are bound by $Pl(A) + Cr(A^c) = 1$. Moreover, a probability function p could be regarded as a particular case wherein $Pl = Cr = p$.

In case where two bbas M_1 and M_2 denote two elements of evidence, we can combine them together using the "Dempster-Shafer fusion" (DS fusion), which results in $M = M_1 \oplus M_2$ that is defined by:

$$M(A) = (M_1 \oplus M_2)(A) \propto \sum_{B_1 \cap B_2 = A} M_1(B_1) M_2(B_2) \tag{2.1}$$

Lastly, through Smets' technique [75], an evidential bba M can be converted into a probabilistic one, whereby every belief mass $M(A)$ is evenly distributed over all elements of A, resulting in the so-called "pignistic probability", Bet , given by:

$$Bet(\omega_i) = \sum_{\omega_i \in A \subseteq \Omega} \frac{M(A)}{|A|} \tag{2.2}$$

where $|A|$ is the number of elements of Ω in A.

It is worth mentioning that there are various evidential fusion rules in the literature that deal differently with the issue of conflicting sources [76] [77] [78].

2.2.2 Parzen-Rosenblatt density estimation

As a statistical tool, the Parzen-Rosenblatt window technique [79] [80], otherwise known as kernel density estimation, is a way to smooth data by making population

inferences based on a finite sample. This technique can be perceived as a non-parametric method to construct the PDF f, of an unknown form, linked to a random variable X. Suppose $(x_1, x_2, ..., x_N)$ an example of the realizations of such a random variable. The challenge is to estimate the f values at multiple points of interest. The smoothing of the kernel can then be seen as a generalization of the histogram smoothing where a window, of a predetermined shape, centered at every point is utilized to approximate the value of density at the given point. This is done by using the following estimator:

$$\hat{f}_h(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right) \tag{2.3}$$

where $K(\cdot)$ is the kernel - a zero-mean non-negative function that integrates to one - and $h > 0$ is a smoothing parameter known as "kernel width". Furthermore, it is possible to use a variety of kernel functions like Uniform (Box), Gaussian (Normal), Triangle, Epanechnikov [81], Quartic (Biweight), Tricube [82], Triweight, Logistic, Quadratic [83], and others.

2.2.3 Discounting methods

Such methods can be used to estimate the weakening coefficients assigned to a source in order to correct its decision. These adjustments differ depending on whether it is a classic or contextual weakening.

Classical discounting

The weakening of mass functions makes it possible to model sources' reliability by introducing a coefficient α^s where for each source s we have:

$$\begin{cases} m'^s(A) = \alpha^s.m^s(A) & \forall A \in 2^\Omega, A \neq \Omega \\ m'^s(\Omega) = (1 - \alpha^s) + \alpha^s.m(\Omega) \end{cases} \tag{2.4}$$

α^s is the weakening coefficient of the s^{th} source. Among the classical weakening methods, we find [84] and [70].

Contextual discounting

The idea behind the contextual weakening is that the reliability of a source can vary depending on the truth of the object to be recognized (the context). For example, a sensor responsible for recognizing flying targets may be more or less able to discern certain types of aircraft. The method we propose belongs to this category and is described below.

Weakening using F-score In this method, we evaluate the ability of each attribute (source) to classify elements belonging to different hypotheses -simple or composite-. This is done by considering each attribute separately to classify a new element. Using a cross-validation process, a confusion matrix is obtained. From this matrix, the "F-score" performance is calculated for all the hypotheses. These measures will be used as weakening coefficients and the equation 2.5 is applied to weaken the mass function of each source s.

$$\begin{cases} m'^s(A) = \alpha_A^s m^s(A) & A \in \left\{2^\Omega/\Omega\right\} \\ \\ m'^s(\Omega) = m^s(\Omega) + \sum\limits_{A \in \{2^\Omega/\Omega\}} (1-\alpha_A^s)\, m^s(A) \end{cases} \tag{2.5}$$

α_A^s is the weakening coefficient of hypothesis A for the s^{th} source.

2.3 NSL-KDD dataset description

In addition to the fact that attack patterns are constantly evolving and changing, the challenge in building a robust Network Intrusion Detection System (NIDS) is that a real-time pattern of network data consisting of both intrusions and normal traffic is out of reach. This is why many recent works are still using the NSL-KDD dataset to evaluate the performance of their approaches [85] [86].

One of the most frequently used datasets for intrusion detection tests is the NSL-KDD dataset which was released in 2009 [87]. In addition to addressing efficiently redundant records' issue in the KDDCUP'99 dataset, NSL-KDD is designed by reducing the number of records in the training and test sets in a sophisticated manner to prevent the classifier from biasing towards frequent records.

There are three datasets within NSL-KDD. One for training which is KDDTrain+ and two for testing with an increasing difficulty respectively KDDTest+ and KDDTest-21, all of which having normal records as well as four distinct types of attack records, as shown in Table 2.1. KDDTest-21 which is a subset of the KDDTest+ is designed to be a more challenging dataset by removing the often correctly classified records. For more details about how KDDTest-21 was conceived, the reader may refer to [87].

Table 2.1: Different classes of the NSL-KDD dataset.

	Normal	**Dos**	**Probe**	**R2L**	**U2R**
KDDTrain+	67343	45927	11656	995	52
KDDTest+	9711	7458	2421	2754	200
KDDTest-21	2152	4342	2402	2754	200

Each record has 41 attributes and a class label as well. These attributes are divided into basic features, content features, and traffic features. Attacks in the dataset are grouped into four categories based on their characteristics: DoS (denial of service attacks), Probe (Probing attacks), R2L (root-to-local attacks) and U2R (user-to-root attacks). Some specific types of attacks are included in the test set but are not included in the training set. This makes it possible to provide a more realistic testing ground.

2.4 Boosted PR-DS

This section describes the theoretical basis of the proposed intrusion detection scheme called Boosted Parzen-Rosenblatt Dempster-Shafer (Boosted PR-DS). To do this, suppose we have a sample of N pre-tagged multiattribute data $(Z_1, ..., Z_N)$ where each datum $Z_n = (X_n, Y_n)$ with $X_n \in \Omega = \{\omega_1, ..., \omega_K\}$ being the tag, and $Y_n = (Y_n^1, ..., Y_n^P) \in \mathbb{R}^P$ being the P-attribute observation. The challenge is then to determine the tag of any new observation $Y_{n'}$.

As shown in Figure 2.1, we begin by briefly outlining the training process carried out on the pre-tagged data sample $(Z_1, ..., Z_N)$. Next, we illustrate the way our approach assigns a new observation $Y_{n'}$ to one of the K classes (tags).

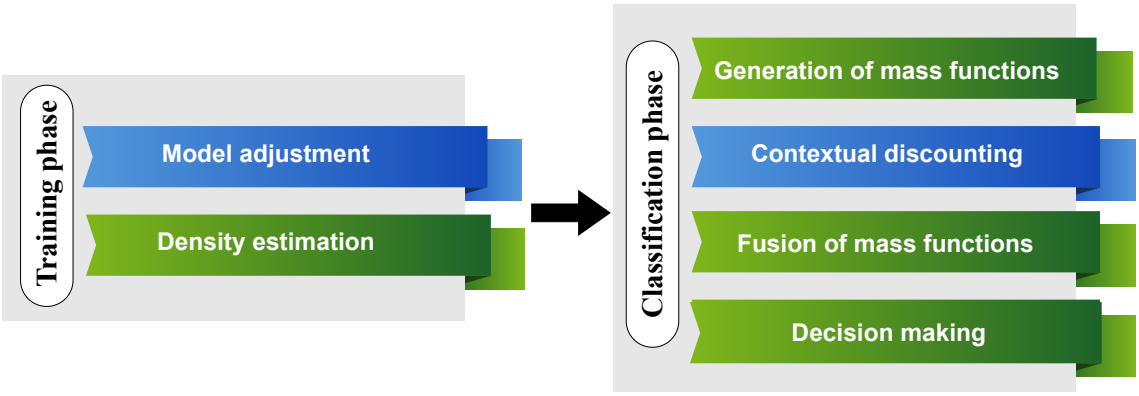


Figure 2.1: Proposed Boosted PR-DS framework

2.4.1 Training phase

Consider the pre-tagged multi-attribute data above $(Z_1, ..., Z_N)$. Under our Boosted PR-DS scheme, the training phase involves two steps. The first is model adjustment which consists of determining the optimal kernel and fusion rule for the data along with the computing of weakening coefficients for each hypothesis. The second step is density estimation where the previously chosen kernel is used to estimate the Probability Density Functions (PDF) of each class for all attributes.

Model adjustment

In the first step, while changing kernels and fusion rules, basic PR-DS is used in a cross-validation process on the training data. The kernel and fusion rule giving the highest accuracy are then selected. To compute the weakening coefficients, we propose to use the F-score measures obtained from classifying each attribute (taken alone) as explained in paragraph Section 2.2.3.

Density estimation

In this step, we use the kernel chosen during the previous step to estimate densities using the Parzen-Rossenblatt method as described in Section 2.2.2 instead of considering that they follow a normal distribution as in the classical case. We thus obtain, for each class $\omega_k \in \Omega$ and for each attribute p $(1 < p < P)$, a Parzen-Rosenblatt density $\hat{f}_k^p$.

Eventually, in addition to the estimated densities, the trained model includes the weakening coefficients and the best-fit fusion rule.

2.4.2 Classification phase

Given a new observation $Y_{n'}$, a mass function M^p for each attribute is constructed based on the estimated densities. The proposed contextual discounting mechanism is then applied using the previously calculated weakening coefficients. Subsequently, the weakened mass functions are combined to obtain a consensual report M. The final decision is made using the so-called Pignistic Transform. In what follows, we describe these different steps.

Generation of mass function

In order to determine the mass $\mathcal{M}^p$ assigned to the attribute p, we will consider the rank function δ_p which is defined from $\{1, .., K\}$ to Ω so that $\delta_p(k)$ is the $k-$ranked element of Ω in terms of $\hat{f}^p$, i.e. $\hat{f}^p_{\delta_p(1)}(Y^p_{n'}) \leq \hat{f}^p_{\delta_p(2)}(Y^p_{n'}) \leq ... \leq \hat{f}^p_{\delta_p(K)}(Y^p_{n'})$. Then, $\mathcal{M}^p$ is determined as follows:

$$\begin{cases} \mathcal{M}^p(\Omega) \propto \hat{f}^p_{\delta_p(1)}(Y^p_{n'}) \\ \\ \mathcal{M}^p(\{\omega_{\delta_p(k)}, ..., \omega_{\delta_p(K)}\}) \propto \hat{f}^p_{\delta_p(k)}(Y^p_{n'}) - \hat{f}^p_{\delta_p(k-1)}(Y^p_{n'}) \end{cases} \tag{2.6}$$

Contextual discounting

To fine-tune the ultimate mass assigned to the p attribute, a weakening process based on the proposed contextual discounting mechanism mentioned in paragraph 2.2.3 is applied.

Fusion of mass functions

Mass Functions assigned to different attributes are then merged into a single consensus mass $M = \bigoplus_{p=1}^{P} M^p$ using the fusion rule selected on the training phase.

Decision making

The final decision is made based on the Pignistic transformation of M:

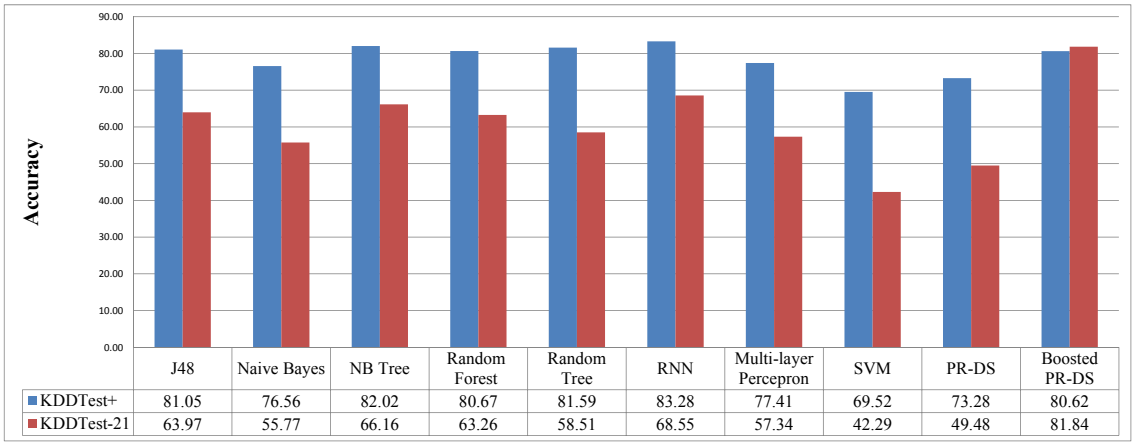
$$\hat{X}_{n'} = \arg\max_{\omega_k} \sum_{A \ni \omega_k} \frac{M(A)}{|A|} \tag{2.7}$$

It is worth noting that the novelty of Boosted PR-DS with respect to those using similar architectures is based on the steps of model adjustment, generation of mass function, and contextual discounting.

2.5 Experimental results

To assess the performance of the proposed boosted PR-DS method, experimental tests are conducted on the NSL-KDD dataset containing two test sets of increasing difficulty, KDDTest+ and KDDTest-21 respectively, as described in Section 2.3.

A comparative analysis is made with regard to nine methods: J48 decision tree learning [88], Naive Bayes [89], NBTree [90], Random Forest [91], Random Tree [92], Multi-layer Perceptron [93], Support Vector Machine (SVM) [94], and Recurrent Neural Networks (RNN) [86], Parzen-Rosenblatt Dempster-Shafer (PR-DS) [95].



	J48	Naive Bayes	NB Tree	Random Forest	Random Tree	RNN	Multi-layer Percepron	SVM	PR-DS	Boosted PR-DS
KDDTest+	81.05	76.56	82.02	80.67	81.59	83.28	77.41	69.52	73.28	80.62
KDDTest-21	63.97	55.77	66.16	63.26	58.51	68.55	57.34	42.29	49.48	81.84

Figure 2.2: Performance of Boosted PR-DS and the other models on KDDTest+ and KDDTest-21.

While giving a comparable accuracy on the KDDTest+ dataset, Boosted PR-DS outperforms the other state-of-the-art methods on the KDDTest-21 testing set as shown in Figure 2.2. This is mainly due to taking the estimated reliability into account by using the contextual discounting mechanism along with adjusting the model by selecting the most suitable kernel and fusion rule for a given training dataset.

To demonstrate the effect of kernel selection, we assess our approach on the KDDTest-21 dataset by changing the kernel each time, while maintaining the other parameters. Figure 2.3 shows that three kernels at least are getting better results

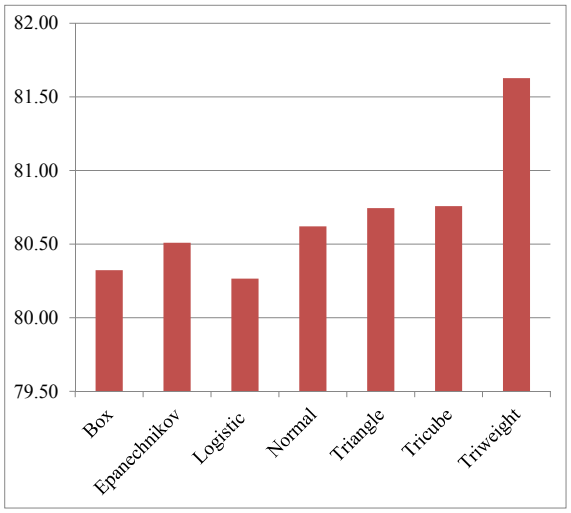


Figure 2.3: Boosted PR-DS on the KDDTest-21 dataset using different kernels

than the Normal kernel which confirms the relevance of choosing an adapted kernel to suitably constructing our densities instead of using the normality assumption.

2.6 Summary

As a conclusion, we can consider Boosted PR-DS as a combination of multiple classifiers where each attribute (source) is a classifier. By using contextual discounting, one may prioritize the decision of an individual classifier regarding those classes in which its accuracy was high in the training phase and be doubtful regarding those classes it did not classify well. Furthermore, Boosted PR-DS choose a suitable fusion rule to take advantage of each individual classifier's knowledge to achieve a consensus decision. Experimental results validate the interest of this approach with respect to other state-of-the-art intrusion detection models. As a possible future direction, it would be interesting to consider handling conflicting sources with a more sophisticated fusion rule.

Nowadays, Deep Neural Networks (DNN) report state-of-the-art results in many machine learning areas, including intrusion detection. Nevertheless, recent studies in computer vision have shown that DNNs can be vulnerable to adversarial attacks that are capable of deceiving them into misclassification by injecting specially crafted data. In security-critical areas, such attacks can cause serious damage; therefore, in the following chapter , we examine the effect of adversarial attacks on deep learning-based intrusion detection.

Chapter 3

Adversarial Training for Deep Learning-based Intrusion Detection Systems

3.1 Introduction

Several research studies have examined the use of different Machine Learning (ML) techniques to improve the accuracy of anomaly-based IDS [96] [97]. Nevertheless, the lack of transferability and the dependence of traditional machine learning on domain knowledge (feature engineering) have been among the main reasons for substituting them with DNNs which not only solved these problems but also yielded, in most cases, the highest accuracies, making them the state-of-the-art in the field of anomaly-based intrusion detection [15].

Despite their popularity, DNNs have proven to be vulnerable to adversarial attacks in computer vision where, by introducing imperceptible changes in an image, an adversary can mislead the classifier. When applied to machine learning-based security products, these attacks can lead to a critical security breach. Although a considerable amount of studies has been conducted on adversarial attacks in computer vision, there are very few studies on this issue in intrusion detection. Therefore, the contribution of this work is double: (1) we study the effect of adversarial attacks on deep learning-based intrusion detection systems. For that, three adversarial attacks are tested: Fast Gradient Sign Method (FGSM) [22], Basic Iterative Method (BIM) [31] and Projected Gradient Descent (PGD) [30] showing that adversarial attacks are able, given enough strength, to mislead the IDS significantly. In addition, (2) this is the first study, to the best of our knowledge, to examine the effectiveness of adversarial training as a defense against adversarial attacks for intrusion detection systems.

In what follows, we briefly recall the concept of DNNs in Section 3.2. The experimental approach is explained in Section 3.3. Results and discussions are presented

in Section 3.4. Concluding remarks and suggestions for possible follow-up work are given in Section 3.5.

3.2 Background

3.2.1 Deep Neural Network (DNN)

DNN refers to a machine learning algorithm made up of multiple interconnected layers where each layer is composed of several nodes - called neurons. Within each neuron, an activation function operates as a basic computing unit. The activation function input on a neuron is the parameter-weighted output of the immediately preceding layer, whilst each layer's output is at the same time the next layer's input.

Frequently described as an end-to-end machine learning process, DNN is capable of learning complex patterns based on limited prior knowledge of input data representation. As a result, deep learning models are widely applied to address large-scale data problems that are frequently inadequately handled by traditional machine learning algorithms. DNN layers fall into three categories: the input layer, the output layer, and, in between, the hidden layer. For large-scale input data, it may be necessary to use several hidden layers so as to learn the subjacent correlation.

DNN can be seen as a function $f(\cdot)$, $f \in F$: $\mathbb{R}^n \rightarrow \mathbb{R}^m$. let Θ be the DNN parameters. Training the model involves finding the optimal parameters Θ where the loss function J (e.g., cross-entropy) is minimal.

For the classification task, the outputs of the last layer in DNN are called logits. The softmax function is added after the last layer in order to transform these logits into a probability distribution, i.e., $0 \leq y_i \leq 1$ and $y_1 + . + y_m = 1$ where y_i is interpreted as the probability that input x has class i. The label with the highest probability C(x) = argmax y_i is assigned as the class of the input x.

Let $Z(x) = z$ be the output of all layers excluding Softmax, thus the full DNN is $F(x) = \text{softmax}(Z(x)) = y$. At the neuron level, the input is first linearly transformed using weights $\tilde{\theta}$ and baises $\hat{\theta}$, and then subjected to a non-linear activation

function σ (e.g., ReLU). The DNN model is a chain function :

$$F = softmax \circ F_n \circ F_{n-1} \circ \cdots \circ F_1 \tag{3.1}$$

Where:

$$F_i(x) = \sigma(\tilde{\theta}_i.x + \hat{\theta}_i) \tag{3.2}$$

3.3 Experimental Approach

In this section, a state-of-the-art intrusion detection system based on deep learning is built to study the effectiveness of adversarial attacks. We focus on untargeted "white box" type evasion attacks, i.e., the attacker has prior knowledge of the internal architecture of the DNN used for detection and carries out his attacks during the prediction process in order to lead the system into misdetection. Subsequently, adversarial training [22] [30] is thoroughly assessed as a defense against adversarial attacks by mixing adversarial samples with clean training data during the training process to enhance the robustness of the DNN against these attacks.

3.3.1 Dataset description

This dataset covers several attacks organized into four classes according to their nature: denial of service (DoS) attacks, probe attacks (Probe), root-to-local (R2L) attacks, and user-to-root (U2R) attacks. The records in the NSL-KDD dataset have 41 features in addition to a class label. These features are grouped into three categories: basic features, content features, and traffic features. For the experimental part, we use KDDTrain+, which contains 125973 records, as follows: 80% of the records are training data and 20% are test data. Table 3.1 provides a summary of the data.

Table 3.1: Different classes of the dataset.

	Normal	**DoS**	**Probe**	**R2L**	**U2R**
Training data	53875	36742	9325	796	42
Test data	13468	9185	2331	199	10

3.3.2 Preprocessing

The preprocessing of the NSL-KDD dataset involves two steps: numericalization and standardization. Neural networks are unable to handle categorical values directly. Numericalization is the process of transforming these categorical values into numerical values. The features that contain categorical values in this dataset are "protocol_type", "service" and "flag". Standardization is an important step to prevent the neural network from malfunctioning because of large differences between features' ranges. That is why we transform each feature into standard normal distribution. In this work, we focus on binary classification; therefore we qualify all attack records as "anomaly" and normal traffic as "normal". We use one-hot encoding to transform the class labels into numerical values.

3.3.3 Building deep learning-based IDS

In order to detect intrusions, a deep binary neural network with two hidden layers, each containing 512 hidden units, is implemented using TensorFlow [1]. Rectified Linear Unit (ReLU) is used as an activation function within each hidden unit so as to introduce non-linearity in these neurons' output. Following each hidden layer, a dropout layer with a dropout rate of 0.2 is employed to prevent Neural Networks from over-fitting. ADAM is set as an optimization algorithm and "categorical_crossentropy" as a loss function to be minimized. softmax layer is added at the end to convert the logits into a normalized probability distribution. the class with the highest probability is considered as the predicted class.

3.3.4 Generating adversarial samples

We use Adversarial Robustness Toolbox (ART) [98] to implement adversarial attacks as well as the adversarial training. ART is an open-source python library for ML security developed by the International Business Machines corporation (IBM).

The generation of adversarial samples can be explained in a simple way. One can consider it as the inverse process of gradient descent where, given a fixed input data x and its label y, the goal is to find the model parameters θ that minimize the

[1] https://www.tensorflow.org/

loss function J. Now, to generate an adversarial sample x', we proceed inversely, given fixed model parameters θ, we differentiate the loss function J with respect to the input data x in order to find a sample x' - close to x - that maximizes the loss function J. FGSM [22] uses a specific factor ϵ to control the magnitude of the introduced perturbation where $\|x' - x\| < \epsilon$. The ϵ factor can be considered as the attack strength or the upper limit of the distortion amount. The adversarial sample x' is then generated as follows:

$$x' = x + \epsilon * sign[\nabla J_x(x, y, \theta)] \tag{3.3}$$

BIM [31] is another attack and is basically an iterative extension of the FGSM applying the attack repeatedly. Similar to BIM, another iterative version of the FGSM is PGD [30]. However, unlike BIM, the PGD is relaunched at each iteration of the attack from many points on the ϵ-norm ball around the original input.

3.3.5 Adversarial training

The idea behind adversarial training is to inject adversarial examples with their correct labels into the training data so that the model learns how to handle them. To do this, we use the PGD attack to generate adversarial samples before mixing them with the training data set. Here, we want to study two parameters of this defense: first, the effect of attack strength ϵ used to generate adversarial samples for the training, let's call it $\epsilon_{defense}$ to avoid confusion with the strength of adversarial attack ϵ_{attack} in the attack phase. Second, the proportion of adversarial training samples compared to clean training samples in the training data.

3.4 Experimental results

In this section, we first evaluate the effect of adversarial attacks on a deep learning-based intrusion detection system. then, in the second part, we examine the effectiveness of adversarial training as a means of making the system more robust against these attacks. we conclude this section by discussing and analyzing the results obtained.

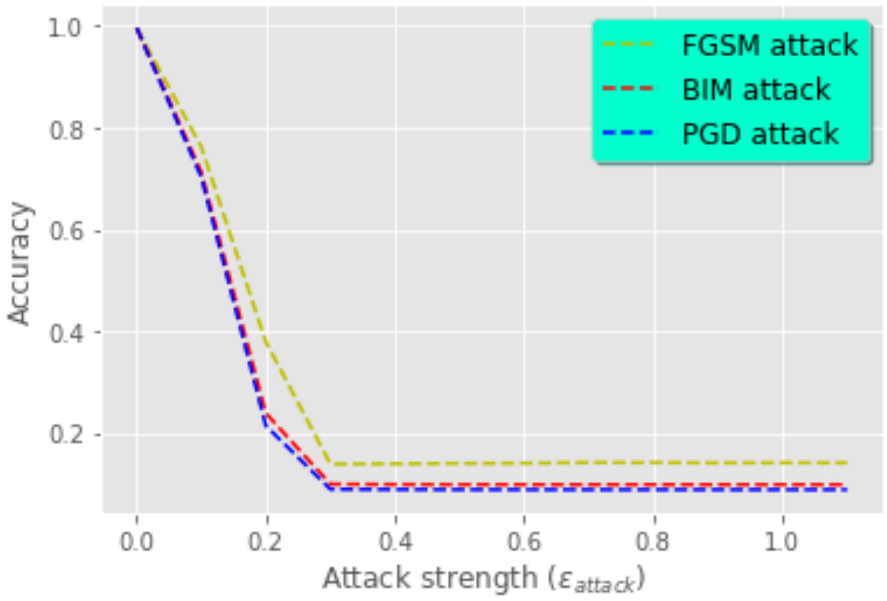


Figure 3.1: Effect of adversarial attacks on deep learning-based intrusion detection system.

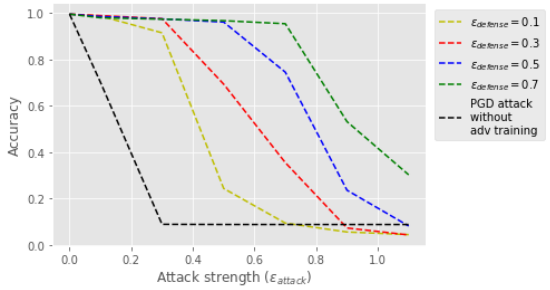
3.4.1 Effect of adversarial attacks on deep learning-based IDS

After training our DNN model, we test its accuracy (the proportion of correct predictions among the total number of cases examined) on unmodified test data. The model gives an accuracy of 99.61%, we then proceed to generate adversarial test data using FGSM, BIM, and PGD respectively. For each attack, the experiment is repeated, intensifying the attack by increasing ϵ value each time. Figure 3.1 shows that all three attacks deteriorate significantly the performance of the intrusion detection system. The FGSM attack lowers the accuracy of the system from 99.61% to 14.13%, while the BIM and PGD attacks decrease it further to 8.85%.

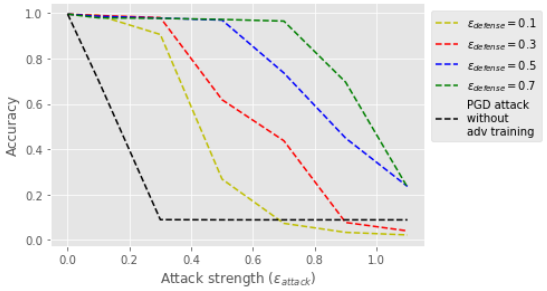
This demonstrates that, with sufficient distortion, adversarial attacks are able to defeat intrusion detection systems based on DNNs and lead them into misdetection.

3.4.2 Adversarial training effect

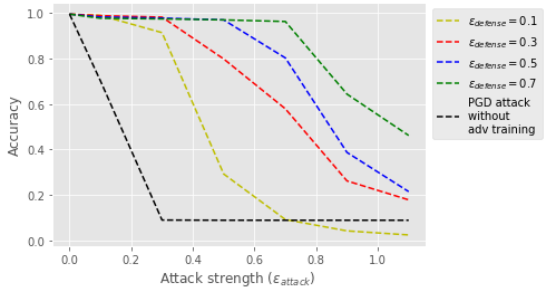
As mentioned in Section 3.3.5, we examine two parameters of adversarial training: 1) $\epsilon_{defense}$ which represents the attack force used to generate adversarial training samples that are mixed with clean training samples. 2) the percentage of adversarial training samples, compared to clean training samples, in the training data. Note that all the adversarial examples used are generated via the PGD attack.



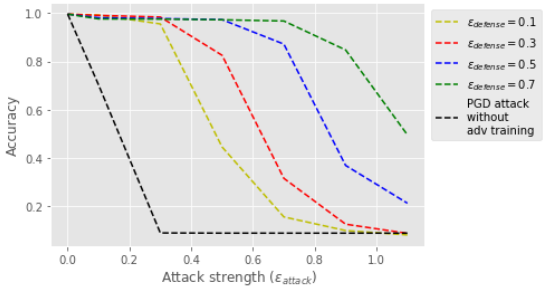
(a) Percentage of adversarial training samples in the training data = 30%



(b) Percentage of adversarial training samples in the training data = 50%



(c) Percentage of adversarial training samples in the training data = 70%



(d) Percentage of adversarial training samples in the training data = 90%

Figure 3.2: Effect of adversarial training on the robustness of deep learning-based intrusion detection system

We begin by setting the percentage of adversarial training samples in the training data to 30%, giving a fixed value of $\epsilon_{defense}$. After training the model with this mixed training data, we apply PGD attack by increasing the value of ϵ_{attack} each time. We repeat the experiment by increasing the value of $\epsilon_{defense}$ used for adversarial training as shown in Figure 3.2a. The whole process is repeated by setting the percentage of adversarial training samples to 30%, 50%, 70%, and 90% respectively.

Figure 3.2 illustrates that compared to using only clean training data, adversarial training improves the robustness of the intrusion detection system against adversarial attacks. Although with sufficient attack force, the accuracy of the detector decreases considerably. We also note that increasing strength of the adversarial examples $\epsilon_{defense}$ used for the training helps to improve the robustness of the detector to some extent, making it more difficult for the attacker to create adversarial samples with a small distortion that can mislead the intrusion detection system. The same cannot be said for the impact of the percentage of adversarial training examples on the robustness of the intrusion detection system because while for $\epsilon_{defense} = 0.7$, higher percentages improved the robustness of the detector against adversarial at-

tacks as shown in Figure 3.3, this improvement is not observed for the other values of $\epsilon_{defense}$. Thus, it is safe to say that the percentage of adversarial training examples doesn't have a direct link to the robustness of the intrusion detection system using adversarial training. This could be explained by the fact that the added dropout layers are designed to reduce overfitting effect on DNN, so as long as the model is fed with enough adversarial samples in the training phase, its performance won't change much by adding data with similar information.

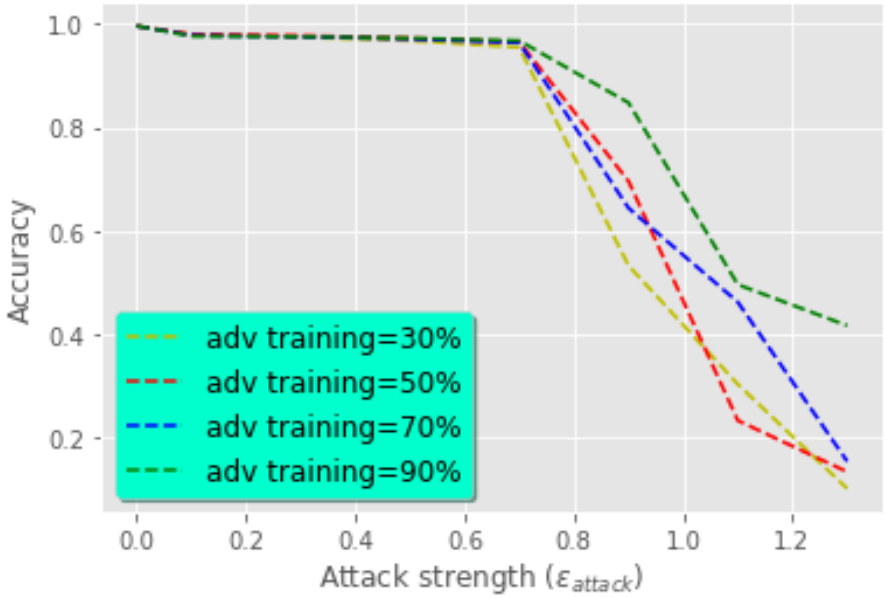


Figure 3.3: Effect of the percentage of adversarial training samples in the training data, $\epsilon_{defense} = 0.7$.

Another important aspect is the effect of adversarial training on the performance of the intrusion detection system when tested on clean test data. While results of the previous experiments indicate that adversarial training increases the robustness of deep learning-based intrusion detection systems, Figure 3.4 shows that adversarial training slightly decreases the accuracy of the detector when tested on clean test data. This indicates that there is a trade-off between robustness and accuracy. The decrease in accuracy of the intrusion detection system on clean test data could be explained by the fact that as the model is trained with adversarial samples, its decision boundary would change in comparison to clean data training.

From a practical point of view, given malicious network traffic, such as HTTP traffic that wants to connect to bad URLs, such as command and control servers, the attacker can use adversarial generation techniques to transform this malicious

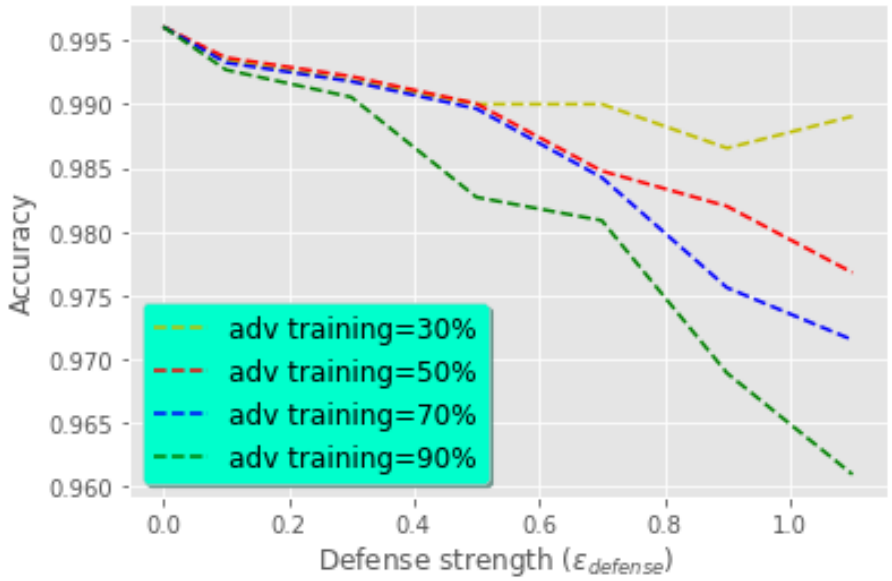


Figure 3.4: Effect of adversarial training on the performance of the intrusion detection system on clean test data.

network traffic into normal traffic for the intrusion detection system while maintaining its maliciousness, for example by adding small amounts of specially crafted data to the network traffic as padding. This allows the attacker to mislead the intrusion detection system. Adversarial training, on the other hand, is a defensive technique. It seeks to make the attacker's task more difficult by making small distortions insufficient to bypass the intrusion detection system.

3.5 Summary

In conclusion, adversarial attacks are a real threat to intrusion detection systems based on deep learning. By generating samples using adversarial attacks, an attacker can lead the system to misdetection and, given sufficient attack strength, the performance of the intrusion detection system can deteriorate significantly. As a defense against such attacks, the adversarial training was examined in depth. The results show that this method can improve to some extent the robustness of deep learning-based intrusion detection systems. However, it comes with a trade-off of slightly decreasing detector accuracy on unattacked network traffic. An interesting future work would be to propose new defense mechanisms against adversarial attacks by exploring uncertainty handling techniques.

The use of Machine Learning techniques in anomaly-based intrusion detection systems has seen much success. However, recent studies have shown that Machine learning in general and deep learning specifically are vulnerable to adversarial attacks where the attacker attempts to fool models by supplying deceptive input. Research in computer vision, where this vulnerability was first discovered, has shown that adversarial images designed to fool a specific model can deceive other machine learning models. In the following chapter, we investigate the transferability of adversarial network traffic against multiple machine learning-based intrusion detection systems.

Chapter 4

Detect & Reject for Transferability of Black-box Adversarial Attacks Against Network Intrusion Detection Systems

4.1 Introduction

Research in the field of computer vision, where this vulnerability was first discovered, has shown that adversarial images designed to fool a specific model can, to some extent, fool other machine learning models [11]. This is known as the transferability property of adversarial attacks. By exploiting this property, an attacker can build a surrogate intrusion detection system, create adversarial traffic for that detector, and then attack another intrusion detection system without even knowing the internal architecture of that detector, leading to a black-box attack.

To avoid this kind of vulnerability, we are conducting this research and the following are our contributions in this work:

- To the best of our knowledge, this is the first study to examine the transferability of adversarial network traffic between multiple anomaly-based intrusion detection systems with different machine learning techniques in black-box settings.

- In addition, we construct an ensemble intrusion detection system to examine its robustness against the transferability property of adversarial attacks compared to single detectors.

- Finally, we investigate the effectiveness of the Detect & Reject method as a defensive mechanism to mitigate the effect of the transferability property of adversarial network traffic against machine learning-based intrusion detection systems.

4.2 Proposed Approach

Previous work has shown that the accuracy of a DNN-based IDS can be significantly reduced when exposed to adversarial attacks [15, 99, 100]. In this section, we construct a DNN-based IDS and five other ML-based IDSs to examine whether the same adversarial instances designed for a DNN-based IDS can be transferred to other ML-based IDSs, which are trained on a different data set, without knowing anything about their internal architectures. We also build an ensemble intrusion detection by clustering the five ML-based IDSs to study whether having multiple classifiers voting prediction can be a defense against the transferability property of adversarial attacks. Finally, we implement the Detect & Reject method as a defense mechanism for the intrusion detection system and evaluate its robustness to transferable adversarial examples.

4.2.1 Dataset partitioning

The NSL-KDD dataset contains 41 network traffic characteristics, covering three aspects: basic characteristics, content characteristics, and traffic characteristics. Many attacks are covered in this dataset; they can be further classified into four families of attacks: denial-of-service (DoS) attacks, probe attacks (Probe), root-to-local (R2L) attacks, and user-to-root (U2R) attacks. We use KDDTrain+ in our experiments by dividing it into 80% and 20% for training and test data respectively. The training data is divided into two almost equal parts A and B to train the DNN-based IDS separately from other ML-based IDSs so as to examine the transferability property. The data used in the experimental part are summarized in Table 4.1 and their partitioning is illustrated in Figure 4.1 .

Table 4.1: Summary of the network traffic dataset.

	Normal	**DoS**	**Probe**	**R2L**	**U2R**
Training data A	26938	18371	4663	398	21
Training data B	26937	18371	4662	398	21
Test data	13468	9185	2331	199	10

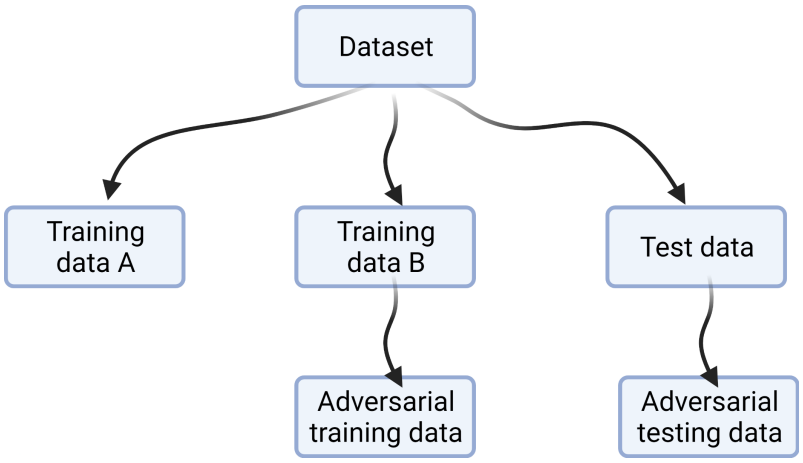


Figure 4.1: The partitioning of the dataset for training and testing of IDS

4.2.2 Preprocessing

The network traffic included in this dataset is heterogeneous and contains both numerical and categorical values. Many machine learning algorithms do not support categorical values, hence the need for the numericalization step that transforms these categorical inputs into numerical values. In the case of the NSL-KDD dataset, the categorical features are "flag", "protocol type" and "service". Another important aspect is feature scaling, which consists of converting all features to the same scale to ensure that all features contribute equally to the result and also to help the gradient-based ML algorithms converge faster to the minima. We restrict our study to a binary classification where we consider any type of attack as "intrusion" and the rest as "normal" traffic.

4.2.3 Building Anomaly-based Intrusion Detection Systems

TensorFlow is used to build the DNN-based IDS, it consists of two hidden layers with 512 units each. As an activation function, we use Rectified Linear Unit (ReLU) to increase the non-linearity. To prevent overfitting, a dropout layer with a 20% dropout rate is placed after each hidden layer. ADAM and categorical cross-entropy are used as an optimization algorithm and loss function respectively. In the end, the logits are converted to probabilities using a softmax layer. The final prediction is assigned to the highest probability class.

We acknowledge the use of Scikit-learn [101] to build five ML-based IDSs. The default settings were maintained. These five ML algorithms were selected due to

their popularity in the ML community: Support Vector Machines (SVM), Decision Tree (DT), Logistic Regression (LR), Random Forest (RF), and Linear Discriminant Analysis (LDA). We also construct an ensemble IDS by grouping these five ML algorithms where the final prediction is made using the majority voting rule.

4.2.4 Transferability of Adversarial Attack in Black-box Settings

In order to test the transferability property of adversarial attacks, we build a DNN-based IDS where we generate adversarial network traffic records in a white-box setting and then test them against five different ML-based IDSs. Note that the five ML-based IDSs are trained on a different dataset (Training data B) and the adversary records were generated without assuming any knowledge of the internal architecture of these five ML-based IDSs, which means that we are working under a black-box setting assumption.

Two adversarial attacks were implemented to generate adversarial network traffic records: FGSM and PDG. For this, we use Adversarial Robustness Toolbox (ART) [98]. The experiments are repeated by increasing the attack strength ϵ to investigate the amount of perturbation required for the adversarial attack to be transferred from the DNN-based IDS to the other five ML-based IDSs in black-box settings.

4.2.5 Defenses against the Transferability of Adversarial Attacks

Since the ensemble technique is known to increase accuracy over a single classifier [102], we want to examine whether it can also increase its robustness against the transferability of adversarial attacks in a black-box setting. To do so, we construct an ensemble IDS based on the previous five ML-based IDSs and use the majority voting rule to obtain the final decision.

The second defense we consider is the Detect & Reject method [103], which involves training our IDSs to detect not only "abnormal" and "normal" traffic, but also a third class called "adversarial". Thus, whenever the IDS decides that a network traffic record is adversarial, it is rejected. We implement this method

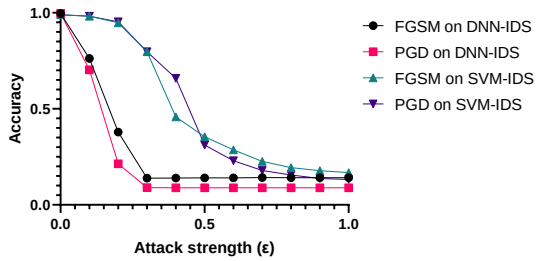
on the five ML-based IDSs and examine their robustness to the adversarial attack transferability property.

4.3 Experimental Results

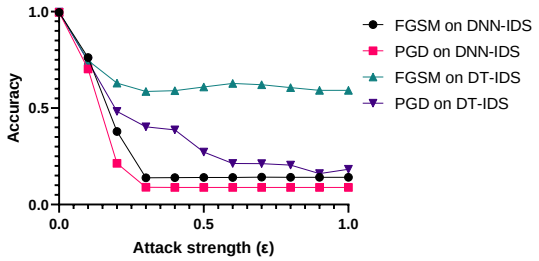
In this section, we present the results of experimenting our appraoch. Subsection (A) illustrates the effect of the transferability property of adversarial attacks on the five ML-based IDSs. In subsection (B), we examine the robustness of the ensemble IDS against these attacks in black-box settings. Subsection (C) illustrates the robustness improvement of all IDSs after adding the detection and rejection mechanism to the five ML-based IDSs.

4.3.1 Transferability of Adversarial Attacks in Black-box Settings

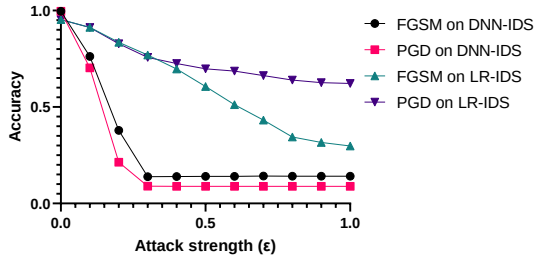
In this study, we use two adversarial attacks: FGSM and PGD to generate adversarial network traffic records from "Test data". These adversarial records are specifically designed to fool DNN-based IDSs since both attacks have access to the internal architecture of DNNs. As mentioned earlier, we train the DNN-based IDS using "Training data A", while the other 5 ML-based IDSs are trained using "Training data B". Adversarial traffic records are used at test time to attempt to mislead the IDSs. As shown in Figure 4.2, increasing the attack strength (ϵ) further degrades the accuracy of the DNN-based IDS. When testing these adversarial network traffic records on the five ML-based IDSs, we find that their accuracy decreases, even though the attacks do not have access to their internal architectures. We also note that although the accuracy of the ML-based IDSs did not deteriorate as much as the DNN-based IDSs, some models were more vulnerable than others. This may be due to their differentiability property, i.e., they are composed of differentiable elements, since the decision tree and the random forest, whose accuracies were least affected, are non-differentiable models that are not amenable to gradient descent due to their Boolean nature, unlike SVM or logistic regression for example.



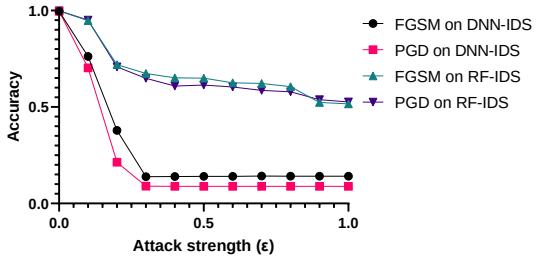
(a) Transferability from DNN-based IDS to SVM-based IDS



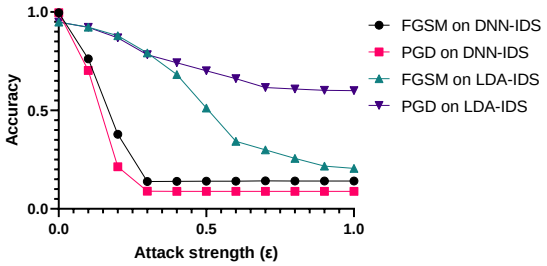
(b) Transferability from DNN-based IDS to DT-based IDS



(c) Transferability from DNN-based IDS to LR-based IDS



(d) Transferability from DNN-based IDS to RF-based IDS



(e) Transferability from DNN-based IDS to LDA-based IDS

Figure 4.2: Transferability of adversarial attacks against ML-based intrusion detection systems in black-box settings

4.3.2 Ensemble Intrusion Detection System Robustness

Since the ensemble technique is known to improve accuracy over a single model, we investigate whether it could also improve robustness. To this end, we construct an ensemble IDS based on the five ML-based IDSs using the majority voting rule. The same setup as in the previous experiment is maintained, which means that the 5 ML models used to build the ensemble model are trained using the "Training data B". The adversarial traffic records are generated from the "Test data" using the FGSM and PGD attacks. These adversarial records are designed to fool DNN-based IDS since both attacks can only access the internal architecture of the DNN model. As shown in Figure 4.3, the ensemble IDS is not able to resist the transferability

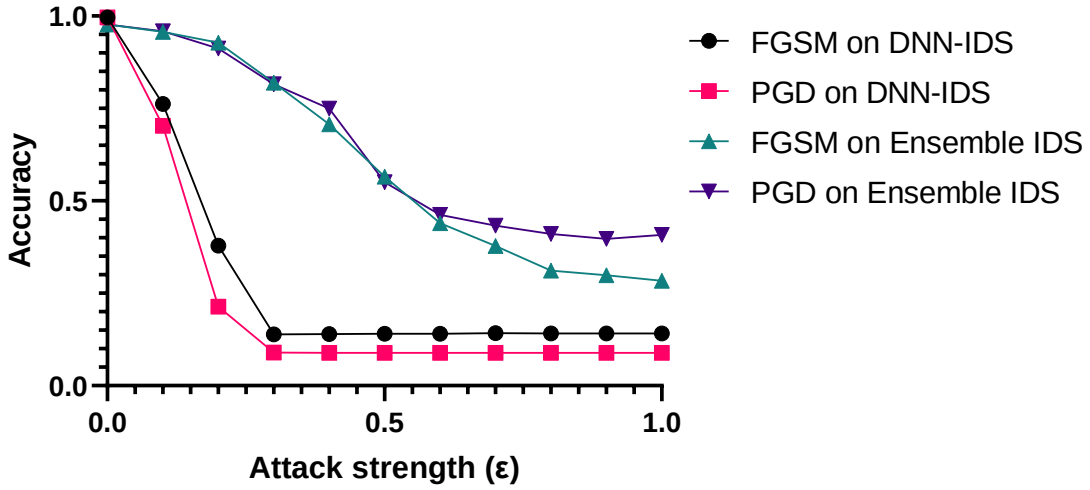
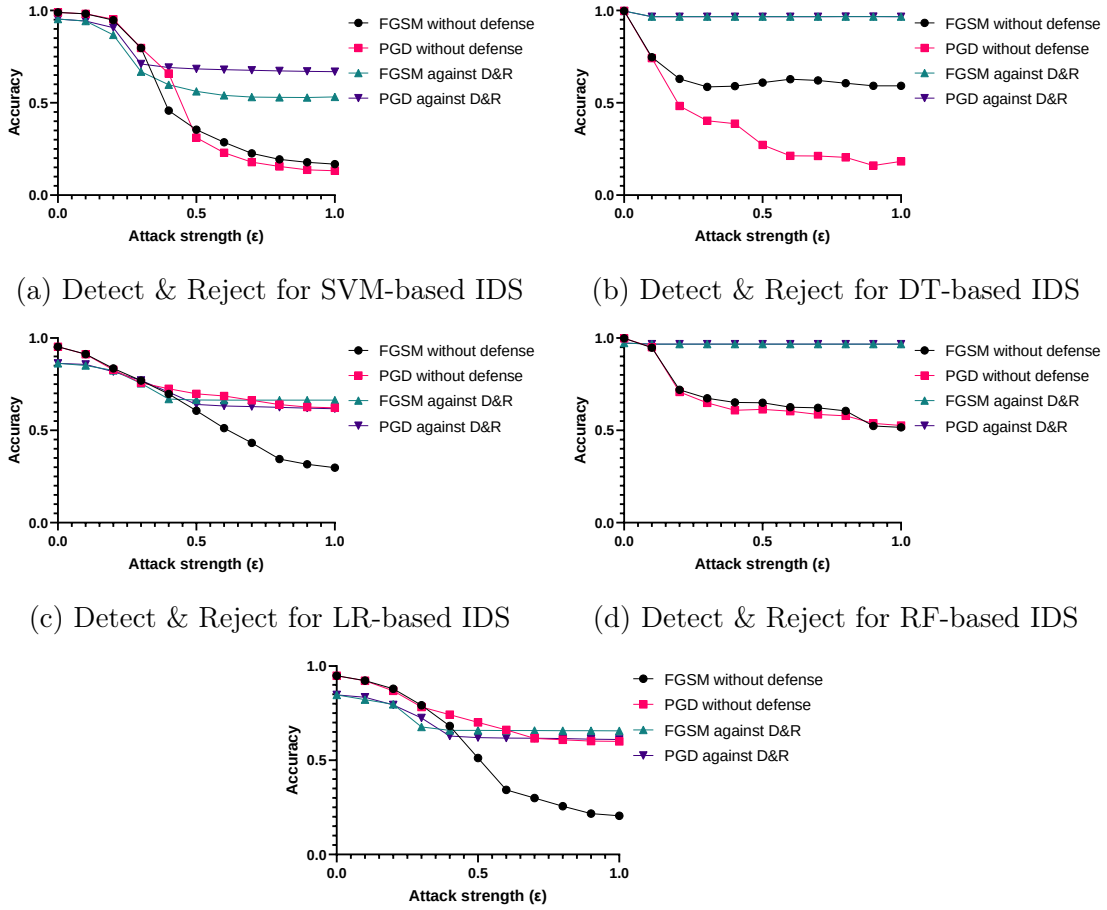


Figure 4.3: Adversarial attack transferability from DNN-based IDS to Ensemble IDS

property of adversarial attacks, even though no information about the ensemble IDS was used to generate these adversarial records. This shows the ease of an evasion attack against an intrusion detection system without even knowing its internal architecture, simply by building a surrogate IDS (DNN-based IDS in our case) and generating adversarial network traffic for this surrogate model.

4.3.3 Detect & Reject for Adversarial Network Traffic

In order to limit the effect of adversarial attacks in a black-box context, we implement the Detect & Reject method in each of the five ML-based IDSs. This method consists of re-training the model to detect not only "abnormal" and "normal" traffic but also "adversarial" traffic. PGD is used against "Training data B" to generate "Adversarial data". After that, the ML model uses a combination of "Training data B" and "Adversarial data" during the training phase to learn to distinguish the three classes. During the prediction phase, any network traffic record recognised as "adversarial" will be rejected. As shown in Figure 4.4, all five ML-based IDSs have improved their robustness against adversarial attacks. Decision Tree and Random Forrest, which is an ensemble version of Decision Tree, have the highest detection rates of adversarial network traffic compared to the other IDSs.



(a) Detect & Reject for SVM-based IDS

(b) Detect & Reject for DT-based IDS

(c) Detect & Reject for LR-based IDS

(d) Detect & Reject for RF-based IDS

(e) Detect & Reject for LDA-based IDS

Figure 4.4: Detect & Reject as a defense against the transferability property of adversarial network traffic

4.4 Summary

From an intrusion detection system perspective, adversarial attacks are a serious threat, as a small intentional perturbation of network traffic can mislead the system. To generate these adversarial records, the attacker must have access to the internal architecture of the machine learning model. However, by exploiting the transferability property of adversarial attacks, he can mislead other intrusion detection systems without having any knowledge about them. Ensemble IDSs, although known to improve model accuracy, are vulnerable to these attacks and thus cannot improve model robustness. On the other hand, Detect & Reject has shown through our experiments to be a suitable built-in defense for intrusion detection systems against adversarial attacks. An interesting future work would be to design more effective defenses to limit the effect of adversarial attacks against intrusion detection

systems.

Nowadays, intrusion detection systems based on deep learning deliver state-of-the-art performance. However, recent research has shown that specially crafted perturbations, called adversarial examples, are capable of significantly reducing the performance of these intrusion detection systems. The objective of the following chapter is to design an efficient transfer learning-based adversarial detector and then to assess the effectiveness of using multiple strategically placed adversarial detectors compared to a single adversarial detector for intrusion detection systems.

Chapter 5

TAD: Transfer Learning-based Multi Adversarial Detection of Evasion Attacks against Network Intrusion Detection Systems

5.1 Introduction

The change in cyber-target has led to an equivalent change in cyber defenses, as the focus has shifted to protecting IDSs themselves from attacks created to undermine their effectiveness. While adversarial learning features some defenses, it is clear that more research is needed to refine these techniques. Adversarial detection has shown promising results in the field of computer vision, but very limited work has been done regarding this method in the field of intrusion detection systems.

The main objective of this work is to design and study the use of multiple strategically placed transfer learning-based detectors of adversarial attacks. The use of multiple detectors could lead to significantly better detection rates against adversarial attacks, as the information is distributed among them and the logical links between these pieces of information can be discovered independently. In our experimental work, the results show that the use of multiple adversarial detectors is more advantageous in parallel IDS than in serial IDS. This is mainly due to the fact that the detectors are more diversified in the parallel architecture. Our four main contributions are the following: (1) Implementation of two IDS models based on deep learning (one in serial form and the other in parallel form) and evaluation of the effect of four adversarial attacks on their performance. (2) Proposal of a new adversarial detection scheme based on transfer learning to allow a better information flow between the IDS and the adversarial detectors. (3) Assessment of

the performance of the proposed approach when exposed to evasion attacks that were not seen in the training phase to simulate the zero-day attack scenario. (4) Performance evaluation of the use of multiple adversarial detectors versus a single detector in both serial and parallel IDS designs.

The remainder of the work is organized as follows. Preliminaries and related works are presented in Section 5.2. We present our evaluation methodology in Section 5.3. We present our performance evaluation in Section 5.4. We conclude in Section 5.5.

5.2 Preliminaries

This section first presents a brief description of the concepts of adversarial learning, as well as the metrics used to evaluate the performance of adversarial attacks. This section also presents the transfer learning techniques used in our detection mechanism. Furthermore, when considering multiple detectors, it is necessary to find a way to combine their respective evaluations into a final decision. Thus, this section presents the three fusion rules used in this work.

5.2.1 Adversarial Learning

Adversarial learning is the name given to the problem of devising attacks against machine learning as well as the defenses against those attacks [10]. There exists a large number of attacks against machine learning models that are often classified with regard to their respective goals [104]. Depending on the phase in which the attack is conducted, adversarial attacks can be divided into either poisoning or evasion attacks.

One of the main differences between poisoning and evasion attacks is the timing of the attack. In the case of the poisoning attack, the adversary targets the IDS during the training phase, whereas for the evasion attack, the adversary launches its attack on the IDS during the testing phase (i.e., only after the IDS has been trained and deployed). Clearly, the poisoning attack is more difficult to execute in a real-world setting, as it requires the attacker to re-train the IDS with its poisoned data. Evasion attacks, on the other hand, allow the attack to be performed without

modifying the IDS.

Since evasion attacks are more practical and therefore more widely used against IDSs, our work will focus on evasion attacks and defenses against them. In particular, this work will only consider evasion attacks against Deep Neural Network-based IDS (DNN-IDS). Although DNNs seem to be the most promising research area in terms of IDS performance [105], these models also seem to be the most vulnerable to adversarial attacks [10].

Assumption of adversary knowledge

When attacking a system in a controlled environment, the first thing to decide is the adversary's knowledge. Indeed, we distinguish between an attacker with the full knowledge of the to-be-attacked model as well as any defenses in place (white-box) and an attacker with no prior knowledge of the attacked system (black-box).

In the case of IDSs, knowledge of the initial classifier is plausible because most IDSs use state-of-the-art models and learning methods. Knowledge of the defenses, on the other hand, is less plausible because there is no clear consensus on the best defense mechanism to implement in this case. In this work, we assume a kind of gray-box situation where the attacker has full knowledge of the IDS but no prior knowledge of additional defenses that are employed to defeat it.

Metrics

When performing adversarial attacks, several metrics can be considered to assess the results. Indeed, one of the commonly used metrics is the success rate of the attack: the difference between the performance of the model before and after the attack. The other commonly considered metric is the strength of the attack. Some attacks allow a strength metric to be defined, which increases the success rate but also potentially the detectability of the attack. The strength often refers to the maximum perturbation that can be applied to the initial sample. In this work, we evaluate the model performance using the standard machine learning metrics: Precision, Recall, and F1-score to evaluate the performance of the IDSs and Detection Rate (Recall) to assess the performance of the adversarial detectors. These metrics are calculated

as follows:

$$\text{Precision} = \frac{tp}{tp + fp} \tag{5.1}$$

$$\text{Recall} = \frac{tp}{tp + fn} \tag{5.2}$$

$$\text{F1-score} = 2 \cdot \frac{Precision * Recall}{Precision + Recall} \tag{5.3}$$

5.2.2 Transfer Learning

Many real-world deep learning configurations involve the need to learn new tasks without compromising the performance of existing tasks. For instance, an object recognition robot may be shipped with a default range of abilities, but additional object models that are specific to the given environment need to be considered. In order to achieve this, new tasks should ideally be learned by sharing the parameters of old tasks without degrading the performance of the latter [1].

Given θ_s a set of shared parameters for a DNN and θ_o task-specific parameters for previously learned tasks, three typical approaches exist for learning new task-specific parameters, θ_n, while leveraging previously learned θ_s as shown in Fig. 5.1:

- Feature Extraction (FE): when learning θ_n, we keep θ_s and θ_o unaltered, and the outputs of the shared layers are used as features for the new task learning.
- Fine-Tuning (FT): in this case, θ_s and θ_n are tuned for the new task, whereas θ_o is unchanged.
- Duplication + Fine-Tuning (DFT): it is possible to mirror the original deep neural network and fine-tune it for each new task to build a dedicated neural network.

5.2.3 Fusion Rules

In order to combine the individual decisions of each detector involved in our proposed design, we will use three fusion rules: majority voting, Simple Bayes averaging, and Dempster-Shafer combination [106].

The majority vote rule is a simple fusion rule and will serve primarily as a reference for the other two rules in our work. With this rule, the final decision will

simply be the individual decision made by the majority of the detectors. The main limitation of this fusion rule is that it processes all models equally, meaning that all models contribute equally to the prediction. This poses a problem if certain models perform well in some situations and poorly in others.

When performing a Bayesian average, the activation score for each class will be summed over all the detectors and then divided by the number of detectors. The highest resulting average will be chosen as the final classification by the system. Using this method helps to account for the confidence of each detector in its classification.

The final fusion rule used in this work is the Dempster-Shafer rule of combination [107]. This rule combines evidence elements from different sources (e.g., detectors) to achieve a degree of belief for each hypothesis (classification decision in our case).

let $\Omega = \{\omega_1, ..., \omega_K\}$, and $\mathcal{P}(\Omega) = \{A_1, ..., A_Q\}$ is its power set, where $Q = 2^K$. A mass function $M{:}\mathcal{P}(\Omega) \rightarrow [0, 1]$ is a basic belief assignment (bba) if $M(\emptyset) = 0$ and $\sum_{A \in \mathcal{P}(\Omega)} M(A) = 1$.

In case where two bbas M_1 and M_2 denote two elements of evidence (e.g., information from two detectors), we can combine them together using the Dempster-Shafer fusion rule, which results in $M = M_1 \oplus M_2$ that is defined by Eq. 5.4.

$$M(A) = (M_1 \oplus M_2)(A) \propto \sum_{B_1 \cap B_2 = A} M_1(B_1) M_2(B_2) \tag{5.4}$$

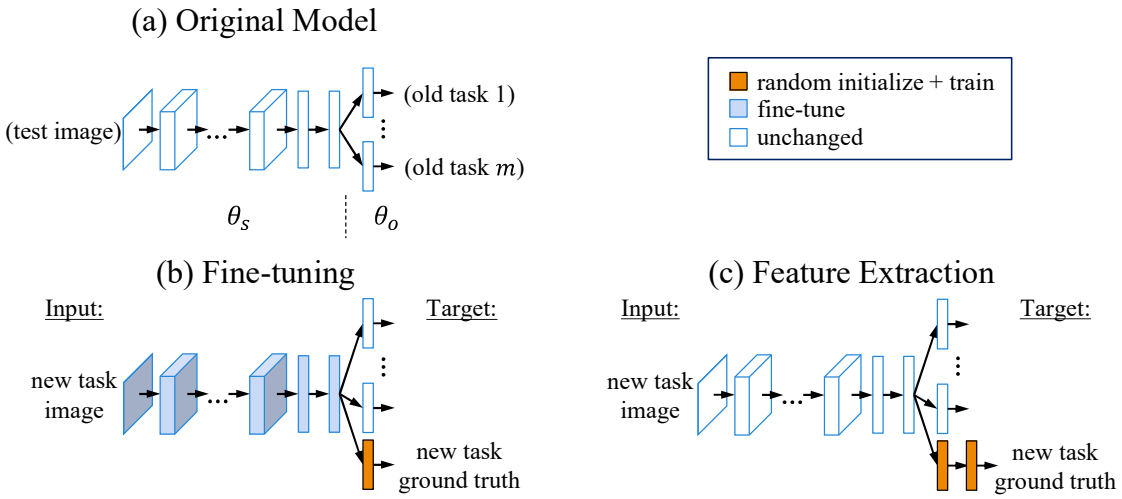


Figure 5.1: An illustration of two transfer learning techniques:Fine-Tuning and Feature Extraction, from [1]

5.3 Evaluation methodology

Anomaly detection is one of the key defenses proposed against adversarial learning in intrusion detection systems. Indeed, to perform any variant of evasion attacks on a trained model, the original features used by the model are modified to maximize the classification error. These modifications can be detected in a variety of ways, including using another neural network trained specifically to recognize the modified input packets. These new neural networks are called adversarial detectors (AdD).

This work aims to investigate the effectiveness of multiple adversarial detectors working together to detect adversarial perturbations during the inference phase. Each of these detectors is placed under a sub-network to receive different levels of information from the DNN-IDS. By combining their respective decisions, we could obtain a final classification affirming the legitimacy of each sample. The hypothesis here is that due to the added level of granularity, multiple detectors could help detect a wider range of perturbations and thus a wider range of evasion attacks. In addition, the use of transfer learning techniques would allow adversarial detectors to learn some important patterns from the IDS (since the two tasks are similar) while augmenting their knowledge based on the adversarial patterns.

To do this, we build two DNN-IDS in both serial (DNN-IDS-serial) and parallel (DNN-IDS-parallel) architectures, perform multiple attacks against these two IDSs, and then design our adversarial detectors by following several transfer learning techniques to finally combine their decisions as shown in Figures 5.3 and 5.5. This defense technique is then compared to adversarial learning.

5.3.1 Network traffic datasets

The experimental evaluation considered in our work is performed on two public network traffic datasets, NSL-KDD [87] and CIC-IDS2017 [108]. NSL-KDD is chosen in this work because of its frequent use in the literature as a benchmark to compare different intrusion detection methods. CIC-IDS2017, on the other hand, is chosen because of its recency, which ensures that it can be considered a good representation of large modern network environments. Each dataset was randomly sampled to include over 50,000 records, with equal representation of both the "normal" and

"attack" classes in terms of network records.

5.3.2 Adversarial examples generation

To craft adversarial samples, we use four different evasion attacks against each model. For this, the set of adversarial attacks that we use in the experiments is Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD), Carlini & Wagner (CW), and DeepFool (DF).

To train and evaluate the adversarial detectors, we create multiple training and testing datasets based on the network traffic dataset as shown in Fig. 5.2. We create an attack-specific dataset for each of the adversarial attacks, comprising the original dataset re-labeled as clean and of the same dataset altered by the chosen attack and labeled as adversarial. This results in balanced datasets between non-altered and altered samples. The final dataset that is created is a balanced version of the attacks. Indeed, while the balanced dataset still contained the original dataset, the altered part is equally shared between the four adversarial attacks.

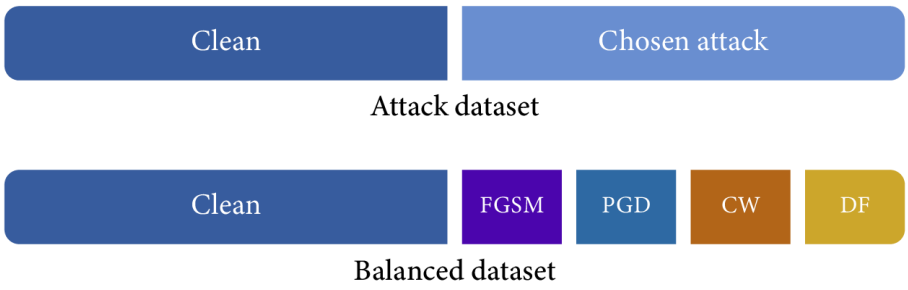


Figure 5.2: Balanced and attack-specific datasets composition

It bears mentioning that our attacks are applied directly to feature vectors (feature-space attacks) as opposed to raw traffic (problem-space attacks) [63]. Therefore, adversarial perturbations may not resemble truly realistic adversarial traffic [12, 18]. However, in an attempt to preserve the functionality of our adversarial samples, certain semantic and syntactic constraints are taken into consideration [23]. A set of features are kept unmodifiable, such as protocol type, service, or flags. We also limit the maximum perturbation rate to 10% to ensure that the generated adversarial samples are close to the original samples.

5.3.3 Serial DNN-IDS design

The first step in this work is to design the DNN-IDS-serial. Its objective will be to classify benign and malicious traffic. Several design choices have to be tested before settling on an optimal design of the DNN: the number of hidden layers, the number of neurons per hidden layer, and the activation function.

The first design choice that was tested was the number of hidden layers to use. Indeed, while it was known that several hidden layers would be used in this DNN, the exact number of hidden layers is a tested parameter. We decided to test between two and four hidden layers. To test each configuration, we trained ten models of each configuration for fifty epochs each. During the training phase, we evaluated each model on our test dataset every ten epochs. This allowed us to keep only the best-performing model and avoid overfitting. We then averaged the performance obtained by each configuration. We found that using three ReLu-activated hidden layers was sufficient to achieve state-of-the-art performance on this dataset and that adding additional layers did not improve performance further.

Based on these results, we decided to use three ReLu-activated hidden layers. We also varied the number of neurons in each hidden layer between forty and two hundred and fifty-six. This variation in the number of neurons resulted in almost no change in overall performance. It was therefore decided that each hidden layer would have two hundred and fifty-six neurons.

5.3.4 Adversarial detectors design for DNN-IDS-serial

The choices made in this stage regarding our adversarial detectors revolved around the design of the detectors and their architecture. Regardless of these choices, we decided to give each detector the information of all layers placed before it in the DNN-IDS-serial. This means that $AdD_{n\in[1,N]}$ will be composed by the $[1, N]$ layers of DNN-IDS-serial in addition to the chosen detector design.

The design of the detectors refers to the number of layers as well as the number of neurons on each layer. We decided to test either one or two ReLu-activated layers, and either forty or two hundred and fifty-six neurons, as we did for our DNN-IDS-serial design. The architecture itself refers to the transfer learning technique used to train our adversarial detectors. The tested architectures are the following:

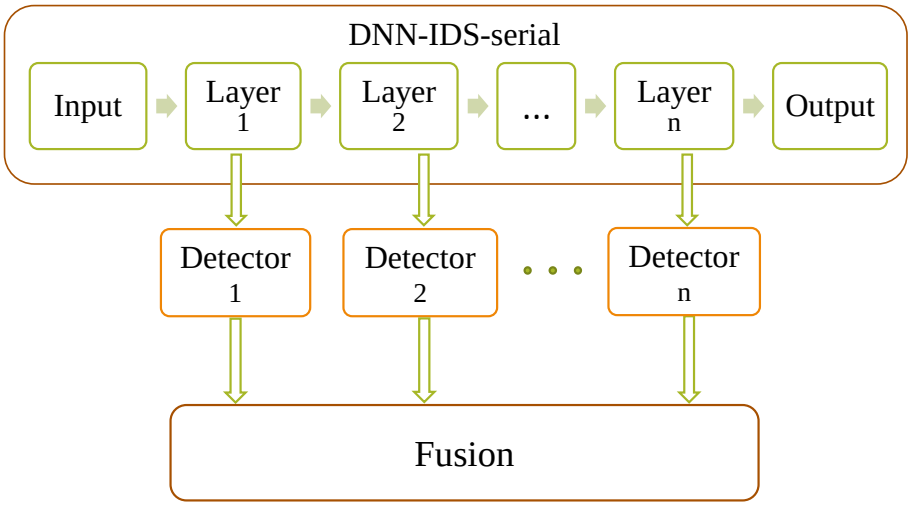


Figure 5.3: An illustration of the adversarial detectors in a serial DNN-IDS design

- Features Extraction: For $AdD_{n\in[1,N]}$, layers $[1, N]$ of the DNN-IDS-serial are shared with the adversarial detector. Those layers are made untrainable by freezing the weights and biases. To that set of layers are added our detector's layers. Those new layers are then trained to detect the attacks. Theoretically, this method allows faster training of the detectors. Indeed, since the shared layers are already trained on a similar task, the provided weights and biases should help to find optimal values for the detector's parameters faster than starting with random values.

- Fine-Tuning: For $AdD_{n\in[1,N]}$, layers $[1, N]$ of the DNN-IDS-serial are shared with the adversarial detector. To that set of layers are added our detector's layers. The ensemble is then trained to detect the adversarial attacks. Since this method changes the DNN-IDS-serial's weights and biases, there is a non-zero risk of changing the DNN-IDS-serial's performance. Since both tasks are quite similar, this change might be beneficial for the original detection performance. But it is also possible that we observe a drop in IDS performance after training our adversarial detectors.

- Duplication + Fine-Tuning: For $AdD_{n\in[1,N]}$, layers $[1, N]$ of the DNN-IDS-serial are duplicated and their copy is passed to the adversarial detector. To those layers are added our detector's layers. We then train the whole set of layers to detect adversarial attacks. This method benefits from the advantages of a Fine-Tuning architecture without risking a performance change for the

DNN-IDS-serial. Indeed, since the layers are duplicated and not shared, the weights and biases of the original DNN-IDS-serial remain unchanged. This architecture would be a suitable solution if the performance change observed while using the Fine-Tuning architecture is detrimental to our DNN-IDS-serial.

To test the design and architecture of our adversarial detectors, we trained them for thirty epochs each and evaluated them in the same way as we evaluated the initial DNN-IDS-serial. The whole experiment was repeated 3 times to average out the results. FGSM attack was used to craft adversarial samples from the NSL-KDD dataset.

Architecture	Layers	Neurons	AdD1	AdD2	AdD3	Mean	IDS F1-score change
DFT (this paper)	2	256	97,12%	**97,50%**	**97,16%**	**97,26%**	No change
		40	97,00%	96,79%	97,05%	96,94%	
	1	256	**97,75%**	97,12%	95,56%	96,81%	
		40	97,01%	97,00%	97,09%	97,03%	
FT	2	256	96,80%	95,93%	96,14%	96,29%	-1,04%
		40	96,31%	96,69%	96,72%	96,57%	-0,57%
	1	256	95,62%	96,52%	96,26%	96,13%	-0,69%
		40	97,20%	97,36%	96,13%	96,90%	-4,80%
FE	2	256	83,56%	82,02%	85,08%	83,55%	No change
		40	84,98%	83,49%	85,38%	84,62%	
	1	256	84,24%	84,84%	85,73%	84,94%	
		40	86,42%	85,37%	86,48%	86,09%	

Table 5.1: Comparaison of the detection rate of the adversarial detectors (AdD) in different design choices. For DFT and FE architectures, the IDS is not re-trained, hence no change in its performance. Note that FE is the architecture used in [3]

As shown in Table 5.1, we can see that the architectures allowing the re-training of the initial layers performed better at detecting the evasion attacks. This result is attributed to the fact that both tasks (detecting intrusion and evasion attacks) are quite similar. This means that the architecture allowing to fine-tune the DNN-IDS-serial benefits from its training and can capitalize on it. Unfortunately, an accuracy drop was observed in the DNN-IDS-serial when using the Fine-Tuning architecture. We notice also that the design choices of the adversarial detectors are of little influence on the detection rate. This is not surprising as it was already observed with the design of DNN-IDS-serial. We decided to use the Duplication +

Fine-Tuning architecture with a design of 2 ReLu-activated layers of two-hundred and fifty-six neurons each for our adversarial detectors.

5.3.5 Parallel DNN-IDS design

In their original paper, the authors present Kitsune as a new IDS generation using an ensemble idea to split the learning process between multiple sub-models each training faster and using less computational and memory resources during the training process. To split the training process, the feature vector extracted from the training samples are first clustered together using their correlation to each other. Once clusters are decided, each cluster of features will be passed to one of the sub-models comprising the ensemble layer of Kitsune. The ensemble layer is comprised of a certain number of auto-encoders each taking one cluster of features and outputting the root-mean-square error (RMSE) between the original feature cluster and the one returned by the auto-encoder. Those RMSE are then used as input to another auto-encoder, the output layer, to compute the final RMSE which is then compared to a fixed threshold to perform a classification between benign and malicious samples. The Kitsune feature mapping, as well as ensemble & output layers are shown in Fig. 5.4. The reason we chose this architecture is to add a deep inspection layer to better detect adversarial perturbations by increasing the granularity.

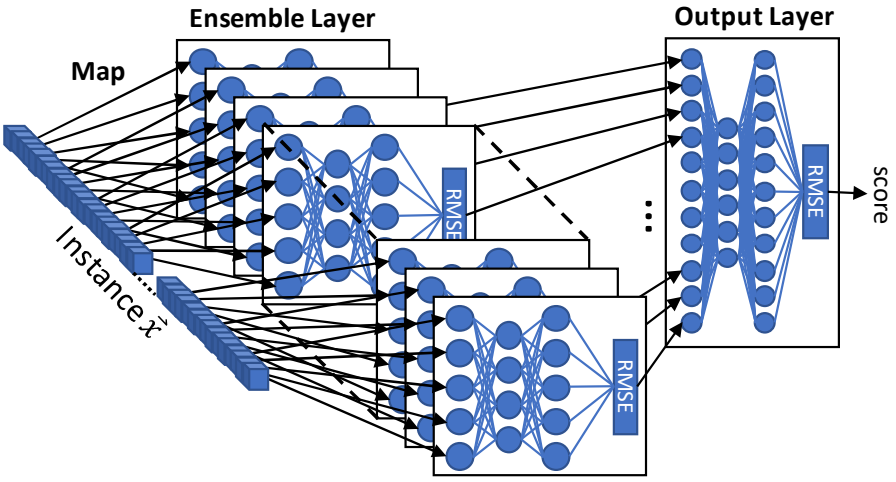


Figure 5.4: An illustration of Kitsune's anomaly detection algorithm, from [2]

Feature clustering

As in the original Kitsune implementation, the network traffic features were split into multiple clusters. The number of clusters K obtained during this step is bound by an arbitrarily fixed parameter (K is fixed to nine (09) for NSL-KDD and five (05) for CIC-IDS2017). Since the features present in our datasets are not the same as the ones used in the original Kitsune paper, the relationship between them is not as direct, and using the original Kitsune clustering method yielded unusable feature clusters. We decided to propose two clustering methods.

The first method, referred to as the distribution method, first performs the same clustering as the original Kitsune version. With highly uncorrelated features, the original Kitsune feature map may contain clusters containing only one feature. As passing only one feature to any neural network from the ensemble layer is not intended, the features belonging to those clusters are distributed between the other smaller clusters. This method will always yield clusters comprising a minimum of two features as well as a number of clusters inferior or equal to K.

The second method, referred to as the cut method, will first compute the number of features in each cluster by dividing the total amount of features by the number of clusters given by K. The features will then be ordered with regard to their correlation before being distributed between the clusters. This method will always yield K clusters of approximately the same size (plus or minus one feature).

By training the DNN-IDS-parallel with the two clustering methods, we observed that the distribution method performed better than the cut method. Therefore, the distribution method was chosen as the reference method throughout the rest of the work.

In the original Kitsune implementation, the ensemble and output layers were composed of auto-encoders. we decided to replace those with deep neural networks similar to the DNN-IDS-serial used in the first part of this work. Each model in our ensemble and output layers is composed of three hidden ReLu layers of two hundred and fifty-six neurons and a bi-neuronal SoftMax layer. Each model of the ensemble layer was then trained for thirty epochs with a performance evaluation every ten epochs, keeping the best-performing model as the final trained model. From the predictions of those trained models for each instance, a new dataset was created.

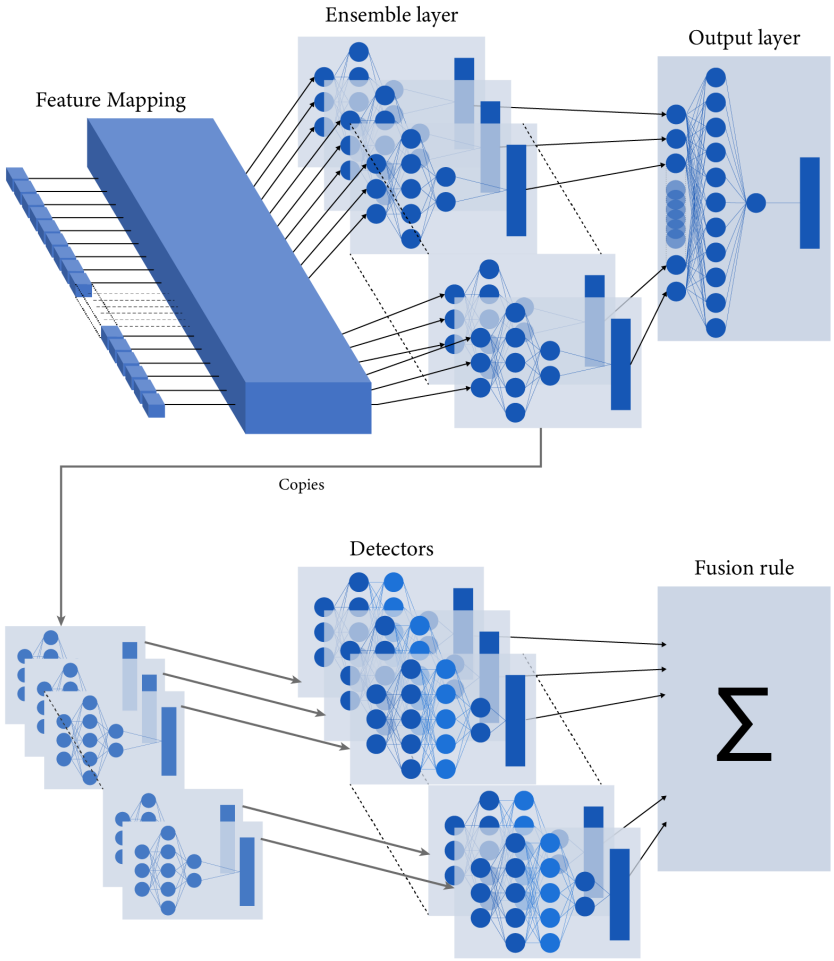


Figure 5.5: An illustration of the adversarial detectors in a parallel DNN-IDS design

This new dataset was then used to train the model composing the output layer in the same manner as the models of the ensemble layer. Using this configuration, the final model achieved state-of-the-art performance on our tested datasets.

5.3.6 Adversarial detectors design for DNN-IDS-parallel

For the adversarial detector design, we decided to follow the same choices as for the DNN-IDS-serial. In Section 5.3.4, we compared three architectures: Fine-tuning, Features extraction, and Duplication + Fine-tuning. The obtained results showed that being able to re-train any part of the DNN-IDS re-used or shared by the adversarial detectors led to a better detection rate. Since we also observed that using a Fine-Tuning architecture without duplication also led to a drop in accuracy for the DNN-IDS, we decided to use the Duplication + Fine-Tuning architecture. The

multi adversarial detectors architecture in a parallel DNN-IDS design is illustrated in Fig. 5.5. Each adversarial detector is composed of:

- A copy of the input layer of the corresponding ensemble layer model.
- A copy of the hidden ReLu layers of the corresponding ensemble layer model, as well as their weights and biases.
- Two additional two hundred and fifty-six neurons hidden ReLu layers, randomly initialized.
- A mono-neuronal sigmoid output layer

The obtained array of detectors will then be trained in the same way as the ensemble layers: thirty epochs with an evaluation every ten epochs and keeping the best-evaluated detector.

5.4 Performance evaluation

This section presents the performance evaluation of the experimental settings. We first investigate the effect of the four adversarial attacks on the performance of the two DNN-IDSs. The adversarial detectors are then assessed by a cross-detection test. The final step is to evaluate the effectiveness of the three fusion rules. The results are then compared to another adversarial defense technique (adversarial training).

5.4.1 Serial DNN-IDS

Adversarial attacks effect on DNN-IDS-serial

In order to evaluate the impact of the four adversarial attacks, namely FGSM, PGD, CW, and DF on the performance of DNN-IDS-serial, we test the model with the four attack-specific datasets crafted as mentioned in Section 5.3.2.

The results presented in Fig. 5.6 confirm the effect of adversarial attacks on DNN-IDS-serial. Indeed, we notice a significant drop in the IDS performance once the samples are altered by any of the adversarial attacks. For example, using PGD on the NSL-KDD dataset resulted in a decrease in F1-score from 83% to 38%, while

using FGSM on the CIC-IDS2017 dataset resulted in a decrease in F1-score from 98% to 48%. These results are compatible with those found in the literature [23,109].

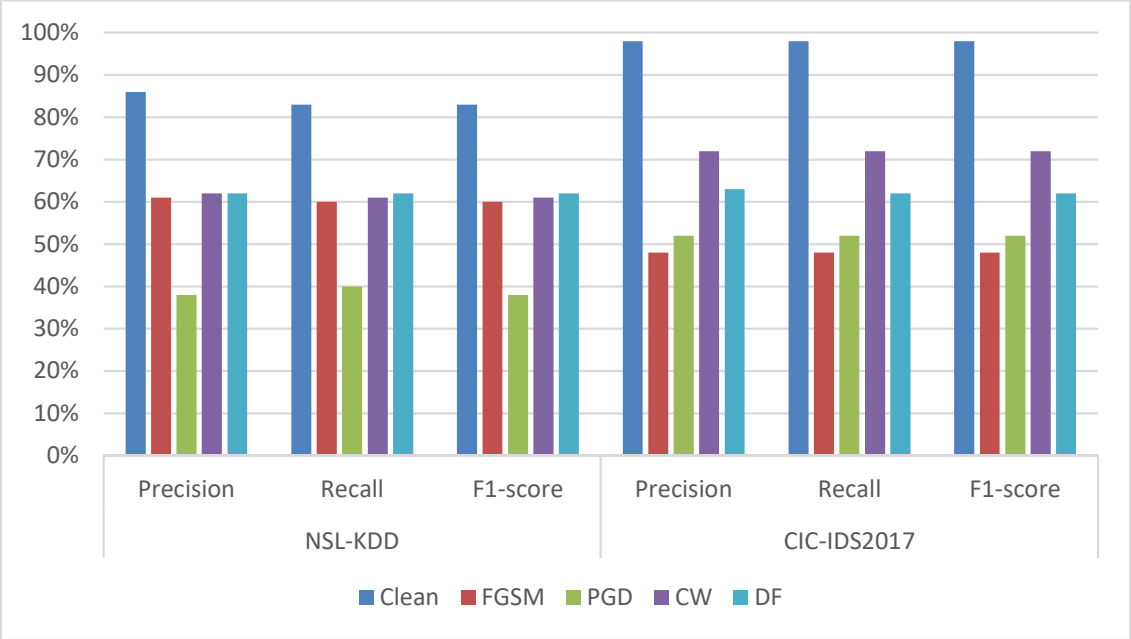


Figure 5.6: IDS performance difference between clean and adversarial network traffic in a serial DNN-IDS design

Adversarial detectors performance

To evaluate the performance of adversarial detectors, we conduct a cross-detection test. For this test, we train a set of adversarial detectors on one of our adversarial training datasets and then evaluate it against all adversarial testing datasets. This allows us to see how an adversarial detector trained on a specific attack or a specific subset of attacks would perform against an unknown attack. We also compare the results between detectors to analyze any differences in performance among them.

As shown in Table 5.2, the first notable observation that can be made is that the detection rate differences between each detector in a set are minimal. If these minimal differences reflect the fact that every detector gives the same answer as the others for a vast majority of the samples, the fusion rules will yield results similar to those of any single detector. On the other hand, if those differences are only the results of the overall proportion of correctly identified samples, the fusion rules should yield better results than any single adversarial detector.

NSL-KDD

AdD1

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.57%	84.06%	50.95%	70.61%	76.35%	76.31%
PGD	71.84%	83.79%	50.89%	70.02%	69.18%	69.14%
CW	53.62%	64.09%	58.82%	63.07%	60.12%	59.94%
DF	71.90%	80.57%	51.13%	70.42%	68.47%	68.50%
Balanced	94.38%	82.12%	58.13%	69.48%	76.13%	76.05%
				Grand Mean		69.99%

AdD2

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.50%	84.37%	50.88%	70.60%	76.37%	76.34%
PGD	66.74%	81.87%	51.02%	67.87%	66.94%	66.89%
CW	53.21%	63.66%	60.26%	63.72%	60.40%	60.25%
DF	70.38%	80.04%	50.77%	69.87%	67.76%	67.77%
Balanced	94.45%	83.00%	57.98%	70.05%	76.32%	76.36%
				Grand Mean		69.52%

AdD3

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.43%	85.07%	51.04%	70.74%	76.58%	76.57%
PGD	64.24%	83.37%	50.98%	68.19%	66.78%	66.71%
CW	59.64%	73.45%	60.02%	64.23%	64.34%	64.34%
DF	67.12%	77.38%	52.19%	69.85%	66.61%	66.63%
Balanced	92.22%	83.70%	56.35%	69.60%	75.54%	75.48%
				Grand Mean		69.94%

CIC-IDS2017

AdD1

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.80%	78.91%	54.61%	71.54%	76.20%	76.21%
PGD	93.78%	86.46%	55.21%	74.76%	77.72%	77.58%
CW	62.70%	66.27%	72.84%	68.25%	67.18%	67.45%
DF	91.23%	81.99%	54.69%	78.63%	76.92%	76.69%
Balanced	98.96%	85.10%	71.87%	77.76%	83.56%	83.45%
				Grand Mean		76.28%

AdD2

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.75%	78.88%	54.28%	71.54%	76.10%	76.11%
PGD	85.28%	85.50%	53.45%	68.28%	73.38%	73.18%
CW	80.45%	78.44%	72.05%	67.78%	74.70%	74.69%
DF	86.74%	80.26%	51.93%	75.76%	73.87%	73.71%
Balanced	92.64%	82.70%	64.32%	76.93%	79.39%	79.20%
				Grand Mean		75.38%

AdD3

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.82%	79.03%	54.59%	72.07%	76.43%	76.39%
PGD	88.27%	84.72%	53.77%	67.90%	73.98%	73.73%
CW	76.22%	75.24%	67.36%	65.51%	71.19%	71.11%
DF	85.39%	78.87%	55.67%	75.89%	74.31%	74.03%
Balanced	95.46%	83.82%	65.65%	76.04%	80.44%	80.28%
				Grand Mean		75.11%

Table 5.2: Detection rate of the individual adversarial detectors in a serial DNN-IDS design on NSL-KDD and CIC-IDS2017

The detection rate results themselves reflect the fact that the CW attack is harder to detect than the other attacks. The detection rate difference could be amplified by the fact that our attack set comprises two FGSM-based attacks, FGSM itself and PGD, and that both could be detected similarly, resulting in a virtually unbalanced attack set. This hypothesis is also supported by the fact that the DF attack yields worst detection rates than both FGSM-based attacks.

Fusion rules

The final step is to implement the three fusion rules we decided to use: Majority Voting, Bayes Simple Average, and Dempster-Shafer Combination rules. To test these fusion rules, we use the same method as when testing the cross-detection of individual adversarial detectors. The only difference is that the decisions of each adversarial detector are now combined using the chosen rule to obtain a final decision.

As shown in Table 5.3, the three fusion rules produced similar results to those of the individual detectors. This confirms our previous hypothesis that each detector is probably giving the same response as the other detectors for most samples. Even though each detector has access to the information of an additional layer compared to

NSL-KDD

Majority Vote Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.80%	84.58%	50.93%	70.76%	76.57%	76.53%
PGD	68.65%	83.13%	50.98%	69.26%	68.11%	68.03%
CW	52.87%	64.71%	60.43%	62.94%	60.38%	60.26%
DF	69.62%	79.71%	50.98%	70.04%	67.53%	67.58%
Balanced	96.41%	84.12%	57.59%	70.60%	77.19%	77.18%
				Grand Mean		69.92%

Simple Bayes Average Fusion Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.80%	84.60%	50.93%	70.77%	76.58%	76.54%
PGD	70.22%	83.57%	51.06%	69.32%	68.59%	68.55%
CW	53.20%	68.62%	60.16%	64.88%	62.02%	61.78%
DF	74.24%	81.10%	51.24%	70.41%	69.20%	69.24%
Balanced	96.70%	83.20%	58.44%	70.22%	77.20%	77.15%
				Grand Mean		70.65%

Dempster-Shafer Combination Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.83%	84.64%	50.94%	70.78%	76.61%	76.56%
PGD	70.67%	83.63%	51.09%	69.17%	68.69%	68.65%
CW	53.33%	68.64%	60.16%	64.96%	62.07%	61.83%
DF	74.91%	81.22%	51.25%	70.44%	69.43%	69.45%
Balanced	96.58%	83.00%	58.47%	70.10%	77.10%	77.05%
				Grand Mean		70.71%

CIC-IDS2017

Majority Vote Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.88%	78.90%	54.53%	71.61%	76.22%	76.23%
PGD	89.09%	85.62%	53.89%	69.32%	74.81%	74.54%
CW	79.61%	78.04%	72.12%	67.48%	74.41%	74.33%
DF	92.91%	81.44%	54.20%	77.05%	76.71%	76.46%
Balanced	96.36%	84.54%	68.83%	77.23%	81.95%	81.78%
				Grand Mean		76.67%

Simple Bayes Average Fusion Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.89%	78.92%	54.50%	71.57%	76.22%	76.22%
PGD	94.20%	86.42%	55.34%	75.09%	77.93%	77.80%
CW	82.15%	78.75%	73.29%	70.90%	76.36%	76.29%
DF	93.16%	82.47%	54.65%	77.93%	77.38%	77.11%
Balanced	99.02%	85.10%	71.35%	77.57%	83.35%	83.28%
				Grand Mean		78.14%

Dempster-Shafer Combination Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.90%	78.82%	54.39%	71.54%	76.13%	76.16%
PGD	94.32%	86.45%	55.38%	75.26%	78.03%	77.89%
CW	82.30%	78.94%	73.30%	72.34%	76.87%	76.75%
DF	93.19%	82.97%	54.89%	78.11%	77.61%	77.35%
Balanced	99.00%	85.16%	71.64%	77.61%	83.44%	83.37%
				Grand Mean		78.30%

Table 5.3: Detection rate of the fusion of the adversarial detectors in a serial DNN-IDS design on NSL-KDD and CIC-IDS2017

its predecessor, it seems that all detectors are obtaining similar levels of information about the tested instances and therefore giving the same results. This means that the use of multiple adversarial detectors in a serial DNN-IDS design does not represent a significant improvement over the use of a single detector in our experimental setting.

5.4.2 Parallel DNN-IDS

Adversarial attacks effect on DNN-IDS-parallel

To perform each adversarial attack on our DNN-IDS-parallel, we decided to craft adversarial input for each model in the ensemble layer. To do this, each attack was performed against each specific subset of features used as input by that model.

As shown in Fig. 5.7, similarly to the DNN-IDS-serial, we observe a significant drop in the IDS performance after performing adversarial attacks against our DNN-IDS-parallel. For example, using CW on the NSL-KDD dataset resulted in a decrease in F1-score from 81% to 15%, while using PGD on the CIC-IDS2017 dataset resulted in a decrease in F1-score from 98% to 10%. These results are compatible with those found in the literature [59].

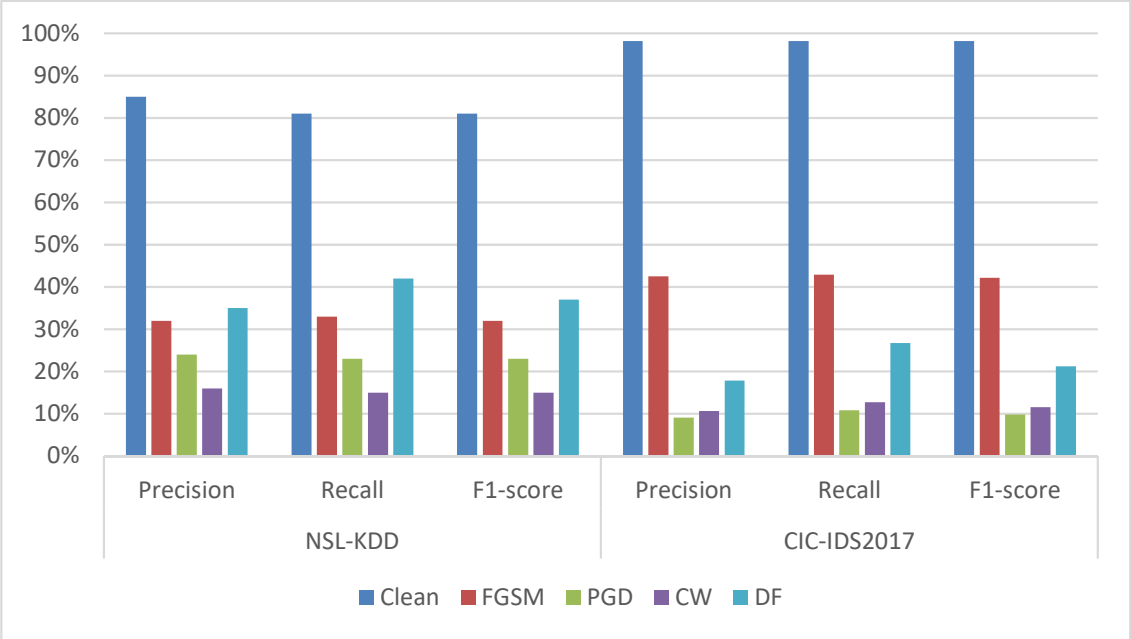


Figure 5.7: IDS accuracy difference between clean and adversarial network traffic in a parallel DNN-IDS design

Adversarial detectors performance

When training the detectors on the various datasets, we observed greatly varying detection rates between the detectors as shown in Table 5.7. Indeed, some detectors achieve an accuracy as low as 50% while others can often reach as high as 100% detection rate. We notice also that many individual detectors are having low detection rate against CW and DF attacks compared to FGSM and PGD. The differences in detection rate observed between the detectors are believed to come from the fact that most attacks alter specific features and disregard others. This would lead to some detectors rarely seeing any altered features, thus classifying almost every sample as legitimate and achieving only a 50% detection rate. On the other hand, some detectors would receive the often-altered features and thus be able to detect an adversarial attack. As the best-performing detector will vary depending on the training dataset used, we expect that the use of multiple detectors will be an improvement over the use of a single detector.

Fusion rules

Considering the results obtained on individual detectors, it is now obvious that the way in which we will combine those individual classifications will be of great impor-

tance for the overall detection rate of our system. Indeed, from the three proposed fusion rules (cf. 5.2.3), the majority vote rule is expected to be the least effective. Since some attacks only alter a small number of features, it is to be expected that only a few adversarial detectors will be able to detect these attacks while the majority will only see unaltered legitimate features, thus resulting in a false classification. While the Dempster-Shafer combination rule also uses the multiplicity of belief as a decisive factor, we believe that the effect will be less important than for the majority vote rule. The Dempster-Shafer combination rule only eliminates outcomes if they are absent from the possible outcomes decided by one of the detectors. This means that for this effect to eliminate one of the outcomes, it is required that one of the detectors returns a classification with a 100% confidence. As the Dempster-Shafer combination rule is a more sophisticated rule and takes the detector uncertainty into consideration, it is expected that this rule will outperform both other rules.

The same cross-detection rate as before was computed, but this time combining the different classifications with each tested fusion rule. This method allows us to compare the different fusion rules but also to observe the possible generalization of detection through multiple attacks.

NSL-KDD

Majority Vote Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	90.48%	73.53%	50.57%	50.37%	50.98%	63.19%
PGD	75.21%	50.02%	50.00%	50.00%	74.05%	59.86%
CW	63.44%	50.76%	57.42%	51.83%	61.67%	57.02%
DF	57.24%	58.35%	50.27%	93.63%	59.99%	63.90%
Balanced	99.38%	86.66%	55.10%	87.25%	80.29%	81.74%
				Grand Mean		65.14%

Simple Bayes Average Fusion Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.88%	95.19%	51.01%	87.85%	76.11%	82.01%
PGD	92.86%	98.00%	50.93%	78.21%	81.98%	80.40%
CW	58.17%	56.55%	91.92%	74.43%	70.30%	70.27%
DF	75.88%	66.40%	53.85%	99.99%	77.11%	74.65%
Balanced	98.96%	99.91%	84.13%	99.75%	94.65%	95.48%
				Grand Mean		80.56%

Dempster-Shafer Combination Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.91%	95.38%	50.99%	86.83%	75.25%	81.67%
PGD	93.52%	99.13%	51.07%	79.23%	84.82%	81.55%
CW	61.36%	54.28%	93.12%	80.00%	74.24%	72.60%
DF	80.81%	72.30%	52.69%	100.00%	78.42%	76.84%
Balanced	98.18%	99.79%	88.93%	99.73%	96.21%	96.57%
				Grand Mean		81.85%

CIC-IDS2017

Majority Vote Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	100.00%	91.19%	50.68%	52.98%	76.15%	74.20%
PGD	98.75%	99.99%	50.07%	57.56%	74.70%	76.21%
CW	93.63%	89.41%	92.92%	82.98%	66.03%	84.99%
DF	90.79%	99.47%	52.14%	76.42%	77.24%	79.21%
Balanced	99.98%	99.94%	77.71%	68.80%	82.34%	85.75%
				Grand Mean		80.07%

Simple Bayes Average Fusion Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	100.00%	99.70%	50.57%	69.24%	77.66%	79.43%
PGD	99.94%	100.00%	51.17%	68.91%	75.79%	79.16%
CW	98.11%	89.02%	95.59%	82.97%	92.29%	91.60%
DF	99.88%	99.29%	67.42%	96.58%	89.77%	90.59%
Balanced	99.91%	99.87%	77.90%	83.54%	95.06%	91.26%
				Grand Mean		86.41%

Dempster-Shafer Combination Rule

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	100.00%	99.99%	51.11%	69.37%	80.05%	80.10%
PGD	99.96%	100.00%	51.29%	70.05%	79.26%	80.11%
CW	99.35%	95.89%	96.41%	89.08%	94.93%	95.13%
DF	99.79%	99.49%	75.43%	97.97%	91.55%	92.85%
Balanced	99.87%	99.84%	88.38%	91.69%	95.99%	95.16%
				Grand Mean		88.67%

Table 5.4: Detection rate of the fusion of the adversarial detectors in a parallel DNN-IDS design on NSL-KDD and CIC-IDS2017

From Table 5.4, we notice that the fusion of the adversarial detectors using the

Dempster-Shafer combination rule is outperforming the individual detectors in terms of detection rate. This confirms the relevance of using multiple, strategically placed adversarial detectors.

The contrast in observed results could come from the differences in the way every attack is performed. Indeed, both the CW and DF attacks approach the feature alteration in a way that is different from the two other, more similar, attacks (i.e., FGSM and PGD). The fact that using the Dempster-Shafer fusion rule or the Simple Bayes Average fusion rule leads to overall better results than the Majority Vote fusion rule confirms the importance of taking the contextual information into account.

5.4.3 Comparison with existing defensive strategies

Our proposed defense falls into the category of anomaly detection, as explained in Section **??**. There are other detection methods, some of which use robust classifiers as detectors. Indeed, it is possible to detect adversarial samples by comparing the classification of two models: a baseline model, trained only on non-adversarial samples, and a robust adversarial model trained on both non-adversarial and adversarial samples. When subjecting a sample to the two models, one can assume that if the two models classify it differently, the sample is adversarial. In theory, if the

NSL-KDD						
Adversarial training detection method						
Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	74.58%	75.61%	52.62%	77.21%	70.95%	70.19%
PGD	72.40%	75.87%	58.65%	70.54%	71.74%	69.84%
CW	67.33%	61.68%	86.19%	68.00%	71.05%	70.85%
DF	70.68%	67.37%	57.59%	80.69%	67.91%	68.85%
Balanced	73.29%	76.40%	83.97%	81.72%	78.21%	78.72%
				Grand Mean		71.69%

Our defense						
Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.91%	95.38%	50.99%	86.83%	75.25%	81.67%
PGD	93.52%	99.13%	51.07%	79.23%	84.82%	81.55%
CW	61.36%	54.28%	93.12%	80.00%	74.24%	72.60%
DF	80.81%	72.30%	52.69%	100.00%	78.42%	76.84%
Balanced	98.18%	99.79%	88.93%	99.73%	96.21%	96.57%
				Grand Mean		81.85%

Table 5.5: Detection rate comparison between adversarial training and our defense in a parallel DNN-IDS design on NSL-KDD dataset

CIC-IDS2017						
Adversarial training detection method						
Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	71.90%	75.73%	70.62%	66.55%	67.48%	70.46%
PGD	60.83%	87.31%	70.24%	68.63%	74.49%	72.30%
CW	62.38%	66.38%	89.08%	68.71%	73.07%	71.93%
DF	64.32%	70.80%	78.89%	80.79%	74.80%	73.92%
Balanced	70.29%	86.58%	88.39%	82.66%	80.35%	81.65%
				Grand Mean		74.05%

Our defense						
Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	100.00%	99.99%	51.11%	69.37%	80.05%	80.10%
PGD	99.96%	100.00%	51.29%	70.05%	79.26%	80.11%
CW	99.35%	95.89%	96.41%	89.08%	94.93%	95.13%
DF	99.79%	99.49%	75.43%	97.97%	91.55%	92.85%
Balanced	99.87%	99.84%	88.38%	91.69%	95.99%	95.16%
				Grand Mean		88.67%

Table 5.6: Detection rate comparison between adversarial training and our defense in a parallel DNN-IDS design on CIC-IDS2017 dataset

NSL-KDD

AdD1

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	77.69%	68.22%	52.14%	96.74%	84.20%	75.80%
PGD	72.50%	68.24%	50.39%	74.34%	78.70%	68.83%
CW	61.55%	58.22%	67.70%	71.56%	67.84%	65.37%
DF	75.22%	69.08%	52.84%	98.20%	74.71%	74.01%
Balanced	77.91%	68.48%	65.77%	95.73%	85.27%	78.63%
				Grand Mean		72.53%

AdD2

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	50.00%	55.66%	51.13%	51.36%
PGD	71.44%	50.00%	50.00%	55.70%	51.47%	55.72%
CW	50.17%	50.19%	54.08%	55.26%	55.49%	53.04%
DF	59.60%	50.00%	50.50%	56.45%	61.99%	55.71%
Balanced	50.00%	53.33%	50.00%	58.04%	55.71%	53.42%
				Grand Mean		53.85%

AdD3

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	50.00%	87.04%	50.00%	57.41%
PGD	52.90%	91.60%	50.00%	50.00%	50.01%	58.90%
CW	60.27%	68.05%	50.00%	89.66%	73.03%	68.20%
DF	72.59%	75.79%	59.22%	95.56%	75.80%	75.79%
Balanced	81.55%	76.28%	64.36%	94.91%	76.48%	78.72%
				Grand Mean		67.80%

AdD4

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	50.08%	84.83%	83.93%	63.77%
PGD	58.81%	50.00%	50.00%	50.31%	55.39%	52.90%
CW	62.79%	51.28%	87.94%	51.96%	58.69%	62.53%
DF	50.00%	62.59%	70.68%	52.01%	50.66%	57.19%
Balanced	50.00%	72.15%	86.26%	84.57%	87.31%	76.06%
				Grand Mean		62.49%

AdD5

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	66.27%	50.00%	50.96%	53.45%
PGD	50.00%	93.50%	51.32%	94.95%	84.05%	74.76%
CW	64.81%	51.20%	73.59%	68.51%	67.01%	65.02%
DF	84.32%	86.75%	67.58%	98.24%	81.82%	83.74%
Balanced	97.33%	97.16%	74.34%	97.97%	91.03%	91.57%
				Grand Mean		73.71%

AdD6

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	81.70%	90.87%	50.00%	50.00%	51.56%	64.83%
PGD	74.14%	50.00%	50.00%	50.00%	75.33%	59.89%
CW	53.34%	60.66%	78.06%	66.00%	66.12%	64.84%
DF	74.74%	65.10%	77.42%	99.45%	71.84%	77.71%
Balanced	89.15%	96.63%	80.38%	92.05%	90.36%	89.72%
				Grand Mean		71.40%

AdD7

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	50.00%	50.00%	50.54%	50.11%
PGD	50.00%	50.00%	50.00%	51.83%	49.33%	50.23%
CW	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
DF	50.11%	50.00%	50.00%	86.56%	51.28%	57.59%
Balanced	77.48%	73.02%	50.00%	54.04%	51.26%	61.16%
				Grand Mean		53.82%

AdD8

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
PGD	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
CW	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
DF	50.00%	50.00%	50.00%	50.00%	50.00%	50.00%
Balanced	50.00%	50.00%	50.00%	50.00%	55.66%	51.13%
				Grand Mean		50.23%

AdD9

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.79%	50.00%	50.59%	64.47%	69.16%	66.80%
PGD	50.00%	51.36%	50.34%	52.64%	50.00%	50.87%
CW	68.51%	50.10%	51.43%	56.79%	60.83%	57.53%
DF	50.02%	61.52%	51.18%	59.50%	67.52%	57.95%
Balanced	99.00%	71.90%	62.41%	67.48%	78.05%	75.77%
				Grand Mean		61.78%

CIC-IDS2017

AdD1

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.98%	99.67%	60.36%	75.10%	84.09%	83.84%
PGD	99.30%	99.98%	61.80%	76.48%	84.44%	84.40%
CW	91.11%	92.41%	93.97%	79.16%	90.04%	89.34%
DF	89.67%	90.69%	66.68%	90.90%	84.44%	84.47%
Balanced	99.26%	99.62%	84.35%	88.63%	91.85%	92.74%
				Grand Mean		86.96%

AdD2

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.62%	76.28%	50.93%	59.52%	71.64%	71.60%
PGD	98.88%	99.68%	50.90%	56.59%	75.34%	76.28%
CW	50.00%	50.00%	50.89%	51.22%	52.11%	50.84%
DF	98.64%	75.05%	52.46%	59.30%	75.34%	72.16%
Balanced	99.41%	75.60%	51.21%	62.05%	71.48%	71.95%
				Grand Mean		68.57%

AdD3

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	99.82%	99.67%	54.14%	72.49%	80.99%	81.42%
PGD	99.81%	99.78%	51.31%	71.61%	81.26%	80.76%
CW	98.77%	89.84%	67.97%	74.00%	81.79%	82.47%
DF	99.22%	99.79%	57.49%	89.24%	85.23%	86.20%
Balanced	99.74%	99.71%	72.70%	78.97%	90.41%	88.31%
				Grand Mean		83.83%

AdD4

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	97.05%	51.70%	53.80%	52.24%	70.16%	64.99%
PGD	50.00%	50.00%	60.06%	50.00%	50.15%	52.04%
CW	50.00%	49.99%	84.13%	51.01%	59.41%	58.91%
DF	50.00%	50.00%	60.62%	52.54%	55.35%	53.70%
Balanced	93.45%	97.29%	50.00%	53.67%	50.00%	68.88%
				Grand Mean		59.71%

AdD5

Train\Test	FGSM	PGD	CW	DF	Balanced	Mean
FGSM	94.03%	72.39%	53.77%	60.39%	68.18%	69.75%
PGD	92.40%	93.97%	53.23%	63.51%	74.45%	75.51%
CW	50.62%	75.77%	53.93%	59.06%	59.17%	59.71%
DF	61.08%	85.08%	62.35%	84.64%	71.57%	72.94%
Balanced	92.89%	93.78%	67.77%	67.32%	84.80%	81.31%
				Grand Mean		71.85%

Table 5.7: Detection rate of the individual adversarial detectors in a parallel DNN-IDS design on NSL-KDD and CIC-IDS2017

submitted sample is non-adversarial, the base model and the robust model should classify it correctly. On the other hand, if the sample is adversarial, the base model should classify it incorrectly, while the robust model should classify it correctly.

We implemented this defense by training a second parallel DNN-IDS model on a

dataset consisting of equal amounts of adversarial and non-adversarial samples. We then submitted a likewise test dataset to both models and recorded their predictions. By comparing these two sets of predictions, we obtained a final set of predictions classifying each sample as adversarial or non-adversarial. As shown in Table 5.5 and Table 5.6, this method yielded a detection rate of 71.69% and 74.05% for NSL-KDD and CIC-IDS2017 respectively. As we demonstrated earlier, our proposal using the Dempster-Shafer combination rule obtained a strictly higher detection rate on both datasets.

These differences in results can be explained by the contrasting objectives of the two strategies. Our proposed defense is specifically aimed at detecting adversarial samples. On the other hand, the adversarial training detection method is a by-product of training a robust classifier which should remove the need for detection (in theory, at least). In addition, the difference in result could stem from the fact that our proposed defense has a higher level of granularity by inspecting each subset of features separately and then reaching a consensus decision using an appropriate fusion rule.

5.5 Summary

In this work, we proposed a new defense approach for evasion attacks against network-based intrusion detection systems. To evaluate it, we implemented two deep learning-based IDS models (one serial and one parallel) and assessed the effect of four known adversarial attacks on their performance, namely: Fast Gradient Sign Method, Projected Gradient Descent, Carlini & Wagner, and DeepFool. A transfer learning technique was employed in the design of the adversarial detection scheme to enable a better information flow between the IDS and the adversarial detectors. Simulation of zero-day attack scenarios allowed for a more reliable evaluation of the performance of the proposed defense when exposed to evasion attacks that were not seen in the training phase. To further improve the detection rate, we proposed and investigated an alternative to a single detector by using multiple strategically placed detectors combined with a sophisticated fusion rule. We show that combining multiple detectors can further improve the detection rate over a single detector in

a parallel IDS design. The reported results confirm the relevance of the proposed defense with respect to existing techniques in the literature.

In future work, we would like to improve the fusion rules used in our design. Indeed, while the use of the Dempster-Shafer fusion rule gave some promising results, it was not fully successful in all cases, which could mean that a more sophisticated fusion rule is needed to model the complex relationship between detectors classification. In addition, we would like to further investigate the relationship between detectability and the strength of adversarial attacks.

Adversarial attacks take advantage of inherent vulnerability of ML algorithms. This raises many questions in the cybersecurity field, where a growing number of researchers are recently investigating the feasibility of such attacks against machine learning-based security systems, such as intrusion detection systems. The majority of this research demonstrates that it is possible to fool a model using features extracted from a raw data source, but it does not take into account the real implementation of such attacks, i.e., the reverse transformation from theory to practice. The real implementation of these adversarial attacks would be influenced by various constraints that would make their execution more difficult. As a result, the purpose of the following chapter is to investigate the actual feasibility of adversarial attacks against network-based intrusion detection systems.

Chapter 6

Adv-Bot: Realistic Adversarial Botnet Attacks against Network Intrusion Detection Systems

6.1 Introduction

Sophisticated methods of cybercrime, such as botnets, are becoming increasingly rampant. Given the severity of the threat, it is essential to have a defense mechanism that can detect all types of botnet traffic. Among these mechanisms, the use of intrusion detection systems dedicated to network analysis, known as NIDS, is gaining popularity. And given the difficulty of some signature-based NIDSs in detecting new botnet attacks or even variants of known botnet attacks, NIDSs based on machine learning algorithms have become more prevalent. These machine learning-based NIDS can detect not only variants of known attacks, but also novel attacks, also known as zero-day attacks [7, 8].

Despite this encouraging achievement, many recent studies have shown that it is possible to fool the ML algorithms used in these detection methods [24, 50], as has been discovered previously in other application areas, such as computer vision [21, 31]. These ML algorithms have been shown to be vulnerable to a variety of adversarial attacks, including poisoning, extraction, and evasion. Evasion attacks are studied in this work because they are more realistic in cyber security scenarios than the other two types of attacks [12]. Evasion attacks can fool any ML model during its inference process by adding slight, often imperceptible, perturbations to the original instance sent to create so-called adversarial instances.

The literature study shows that there has been considerable research on the impact of adversarial attacks on machine learning models [10, 110, 111]. However, their feasibility in domain-constrained applications, such as intrusion detection systems,

is still in its early stages [27–29]. Adversarial attacks can be performed in either white-box or black-box settings. Many white-box adversarial attacks, originally developed for computer vision applications [22, 30, 112], have been applied directly to network traffic without addressing domain constraints properly [13, 23, 113].

Aside from the issue of functionality preservation, white-box attacks necessitate knowledge of the target system's internal architecture. It is unlikely that an attacker would have access to the internal configuration of the ML model [10], making a white-box attack in a realistic environment less likely [12]. As a result, recent research [14, 114] has focused on designing adversarial network traffic in a black-box setting where the attacker has little or no knowledge of the defender's NIDS. Model querying [36] and transferability [22] are two methods for launching black-box attacks.

An attacker can extract useful information such as the classifier's decision by sending multiple queries to the target NIDS, allowing the attacker to craft adversarial perturbations capable of evading the NIDS [114–116]. Although this approach achieves high success rates, it has two limitations. The first is that NIDSs are not designed to provide feedback when queried, unless side-channel attacks are used [117]. The second limitation is that it requires a relatively high number of queries to function properly, exposing the attacker to detection by a simple query detector [14, 54]. As a result, for a realistic attack, the IDS querying approach is less feasible [12].

The transferability property of adversarial examples is used as an alternative black-box approach. Goodfellow et al. [22] were the first to investigate this property in computer vision, demonstrating that adversarial examples that fool one model can fool other models with a high probability without the need for the models to have the same architecture or be trained on the same dataset. An attacker can use the transferability property to launch black-box attacks by training a surrogate model on the same data distribution as the target model [118]. Sniffing network traffic is a simple way to accomplish this, especially in a botnet scenario where the attacker already has a foothold in the corporate network [119]. To the best of our knowledge, this paper represents the first proposal that exploits the transferability property to design a realistic black-box evasion attack by considering the constraints of the NIDS domain to create adversarial botnet traffic.

This work provides a dedicated framework for leveraging evasion attacks to mis-

lead the NIDS into classifying botnet traffic as benign under realistic constraints. Three contributions are included in this work:

- An in-depth analysis of the feasibility constraints required to generate valid adversarial perturbations while preserving the underlying logic of the network attack.
- A new black-box adversarial algorithm that can generate valid adversarial botnet traffic capable of evading botnet detection without any knowledge of the target NIDS.
- A defense that allows the proposed botnet evasion attack to be mitigated. This defense is inspired by adversarial detection and an ensemble method known as bagging.

The remainder of the chapter is organized as follows. Background and related work are presented in Section 6.2. The proposed method is explained in Section 6.3. Results and discussions are presented in Section 6.4. Concluding remarks and suggestions for possible follow-up work are given in Section 6.5.

6.2 Background

With the increasing research on evasion attacks, the feasibility of such attacks in the real world is gaining more and more attention, regardless of the application domain. It is possible to distinguish three criteria that influence the realism of such attacks, namely: knowledge restriction, domain constraints, and manipulation space. The focus of this work is on the analysis of these criteria in the domain of network-based intrusion detection system.

Knowledge Restriction

Adversarial attacks come in three varieties: white-box, grey-box, and black-box. White-box attacks mean that the adversary knows everything about the model architecture and training dataset, in particular all the parameters and meta-parameters, which are, for example, the inputs and gradients in the case of neural networks, or

the tree depth for decision trees. In addition, the adversary may also know the chosen cost function or the optimizer type in the case of neural networks. The gray-box attack, on the other hand, implies that the attacker has some limited knowledge of the target model. He may, for instance, know what data was used to train the model without knowing the type of model used or its internal architecture. A black-box attack occurs when the attacker does not know the architecture of the target model and the dataset used. Nevertheless, even without knowing anything about the model, it could still approach a decision boundary similar to that of the target model and thus fool it by querying the target model and receiving responses in the form of decisions or probabilities. An alternative black-box approach is known as "transferability" where the attacker can create his own model (i.e. surrogate model) with similar functionality to the target model in order to fool it by creating adversarial instances based on his surrogate model and then transferring these instances to the target model to fool it as well. Obviously, black-box attacks are more complicated to perform, not only due to lack of knowledge but also because they require more computational resources to accommodate these accumulated knowledge biases.

Domain constraints

Regarding the feasibility of adversarial attacks, it varies according to the domain in question as it is strongly limited by several constraints. Such constraints can be divided into two main categories: syntactic constraints and semantic constraints.

Syntactic constraints refer to all constraints related to syntax. In their work, Merzouk et al. [23] identified three syntactic constraints that an adversarial instance must respect, namely out-of-range values, non-binary values, and multi-category membership. Out-of-range values are values that exceed a theoretical maximum value that cannot be exceeded. Non-binary values are entries that violate the binary nature of a feature, and multi-category membership are values that violate the concept of one-hot encoding.

Semantic links represent the connections that various features may have with each other. It is challenging to identify precisely such constraints since they are specific to each application and to each particular feature used. Nevertheless, the work of Hashemi et al. [24], and Teuffeunbach et al. [25] suggest an intuitive approach

in the NIDS domain by categorizing the features into roughly three different groups with different semantic links. The first group includes features that can be directly modified by the attacker (e.g., the number of forwarding packets, the size of the forwarding packets, and flow duration). The second group concerns features that depend on the first group. They must be recalculated on the basis of the latter (e.g., number of packets/second or average forward packet size). The third group includes features that cannot be changed by the attacker (e.g., IP address, protocol number).

Manipulation Space

An essential property of a realistic adversarial instance is the ability of an attacker to modify its characteristics. In theory, it is possible to directly modify the features of adversarial instances. However, in real-world scenarios, this approach is considered unsuitable for certain domains, such as IDSs that analyze network traffic. This is mainly due to the fact that the feature extraction process (i.e., from raw traffic to feature space) is not a fully reversible process, unlike in other domains such as computer vision. This means that features can be extracted, modified, but not easily reintroduced into network traffic due to the semantic links between features. Moreover, direct feature modification requires full knowledge of the feature extraction process used by the IDS in order to respect the syntactic or semantic constraints assigned to them. We can therefore deduce that working on the feature space is not very realistic. For this reason, recent studies [18–20] propose to directly manipulate the network traffic, so that it is not necessary to know the features used, nor to transform the feature values into traffic form. In this way, we can distinguish two manipulation spaces, the feature-based and the traffic-based.

6.3 Proposed method

6.3.1 Threat scenario

With the increasing use of machine learning-based NIDS, some research has focused on the possibility of evading these NIDS using adversarial attacks already well known in the field of computer vision, which have shown various weaknesses inherent in machine learning algorithms. In order to overcome these weaknesses, it

is therefore essential to focus on the robustness of machine learning techniques in the cybersecurity domain.

As shown in Figure 6.1, our work is based on a realistic scenario involving a connected computer device in an enterprise network infected with malware, making it part of the botnet controlled by a dedicated centralized server called Command and Control (C&C), which is designed to command a multitude of bots. Within this network, a NIDS using a neural network-based model is present to detect any form of attack on the network. In high-speed networks environment and considering the difficulties of analyzing each individual packet, it is realistic to consider that the NIDS is a flow-based system rather than a packet-based system. This NIDS therefore analyzes the flow data generated by the router based on the traffic outgoing/entering the network. All flows first pass through a flow exporter, which extracts the network features, as described in Table 6.1, containing the information of each network flow and sends it to the NIDS for preprocessing and classification. In this table, three groups can be highlighted:

- The first one, with a green color, contains all the features that can be directly manipulated by an attacker.
- The second one, highlighted in yellow, contains the features depending on the first group, and can therefore be manipulated indirectly.
- The last group, with a red background, contains features that cannot be manipulated by an attacker.

Concerning the temporal aspect of the execution, the processing of the network flow is done in real time on the NIDS, which processes the information online while its learning phase has been carried out offline.

Concerning the knowledge assumptions, it is considered that the attacker has limited access to network data and has no information about the model and parameters of the NIDS used by the company. This means he can only intercept traffic that passes through his machine and gathers limited information about the benign traffic that passes through the bot's machine. In terms of requirements, our scenario assumes that the attacker has previously breached the network by infecting a machine with malware in order to connect to the C&C server. As a result, the

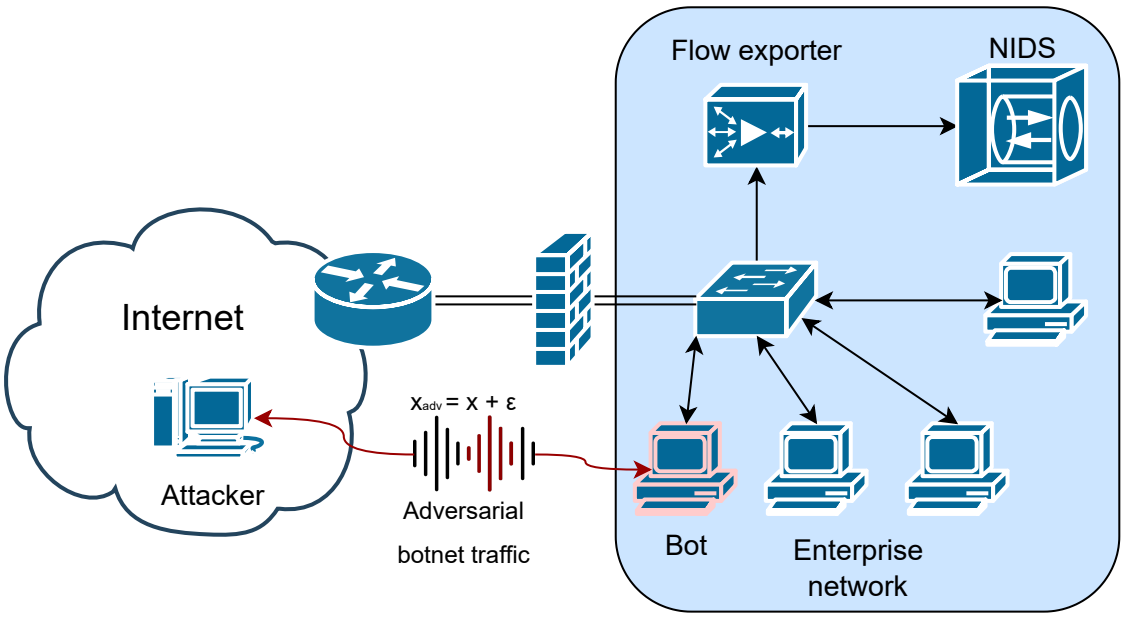


Figure 6.1: An illustration of the considered threat scenario

attacker will be able to gather information about the benign traffic. Regarding the flow exporter, the attacker does not need to have knowledge about it or about the extracted features used by the NIDS since he acts directly on factors that he can manipulate in the traffic space. Given the recurring use of certain network factors (in particular, packet duration, number, and size) in botnet detection by state-of-the-art NIDSs, the attacker can assume that these factors are part of the set of features used by the NIDS. The attacker can retrieve these feature lists from known exporters such as Argus [1], CIC-FlowMeter [2] or nProbe [3] and from scientific papers [50, 120, 121] explaining the features used by NIDS models. The attacker has the ability to communicate with and target the infected computing device. It is therefore also considered that the attacker can manipulate, in both directions of communication, *the duration of packet transmission*, *the number of packets sent and received*, as well as *the number of bytes sent and received*, as shown in Table 6.1, respecting the semantic and syntactic constraints related to the network protocols used and maintaining the underlying logic of his malicious communications. In order to maintain the malware's full behavior, the attacker cannot directly act on certain features, such as the source and destination IP addresses or the type of service. In

[1] https://openargus.org/
[2] https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter
[3] https://www.ntop.org/products/netflow/nprobe/

Table 6.1: Common features description used by flow-based NIDS for botnet attacks [4]

Features Name	Description	Type	Category
Dur	Duration of the flow	Float	Modifiable
Out/InBytes	Bytes outgoing/entering the network	Integer	
TotPkts	Total of exchanged packets	Integer	
TotBytes	Total of exchanged bytes	Integer	Dependent
BytesPerPkt	Total bytes by packet	Float	
BytesPerSec	Total bytes by second	Float	
PktsPerSec	Total packet by second	Float	
RatioOutIn	Ratio between outgoing and entering bytes	Float	
ConnectionState	The state of the connection	Categorical	Unmodifiable
FlowDirection	Direction of the flow	Categorical	
Src/DstPortType	The destination port is a private, registered or known port	Categorical	
IPSrc/DestType	The network IP source or destination	Boolean	
Src/DstTos	Source and destination type of service	Integer	

fact, the features highlighted in red in Table 6.1 cannot be modified by the attacker, either directly or indirectly. Only the three green features: Dur, Out/InBytes, and TotPkts can be manipulated by the attacker. It should be noted that modifying the green features will result in some indirect changes to the yellow features. These changes should be taken into account in order to properly address the respect of semantic and syntactic constraints. To manipulate the network factors explained just before, the attacker can use the following three approaches:

1. Time manipulation attack: with this approach, it is possible to act on two time-related aspects during the attack. On the one hand, by reducing the frequency of the attack packets by increasing the sending time between packets of the same flow, as in the work of Zhijun et al. [122] which shows the implication of this variant on DoS attacks. On the other hand, by accelerating the frequency of attack packets in a moderate way by decreasing the time taken to send the packets. These two variants allow to influence directly the "Duration" feature and indirectly the "BytesPerSec" and "PktsPerSec" features.

2. Packet quantity manipulation attack: This attack can be carried out in two different manners. The first is packet injection, which suggests injecting new

packets into the network flow by creating them directly, with tools such as Hping [4] or Scapy [5], or by breaking a packet into several fragments, using packet fragmentation, so as to preserve the content and underlying logic of the packet without damaging its overall behavior. Packet fragmentation is, for example, used by some attacks such as the TCP fragment attack [123] or a DoS attack. A variant suggested by this possibility would be to resend the same packet multiple times using a tool like Tcpreplay [6]. The other way to do this is packet retention, which would be to not send a packet immediately, but rather to send it after a certain amount of time, thus adding that packet to the next flow, thus avoiding having more packets in a single flow. With the same idea of retention, another suggested possibility could be a modification of the general communication system used in the botnet attack to shorten the number of packets sent between the botnet devices. In both cases, this attack directly influences the "TotPkts" feature, and indirectly the "PktsPerSec" and "BytesPerPkt" features.

3. Byte quantity manipulation attack: Similar to the previous approach, there are two ways to perform this attack. The first is byte expansion. In the case where the communication is not encrypted, a suggestion would be to directly modify the payload to obtain the desired number of bytes. In case it is encrypted, it is assumed that the attacker knows the cryptographic suite used to encrypt the two-way communication channel. This would allow him to add padding using a tool like Scapy, calculate its encrypted version, and then verify that the packet size is the desired one. The addition of padding is known to both communicating parties, allowing the receiver to remove the unnecessary part and thus recover the original payload. The second method is byte squeezing. The idea is to reduce the number of bytes sent. To do this, it would be possible to compress the data to directly reduce the size of the payload using a tool like Zlib [124]. This is a common tool already used in the IoT domain to reduce the bytes exchanged, as explained in [125]. Another possibility would be to modify the general behavior of the botnet system by

[4] http://www.hping.org/
[5] https://scapy.net/
[6] https://tcpreplay.appneta.com/

minimizing the content of the payload to be sent. This second possibility can only be done before the machine is affected by the malicious bot software, but can be proactively prepared by an attacker. Furthermore, whether it is byte expansion or byte compression, the attack directly influences the "OutBytes" and "InBytes" features, and indirectly the TotBytes, "BytesPerPkt", "BytesPerSec" and "RatioOutIn" features.

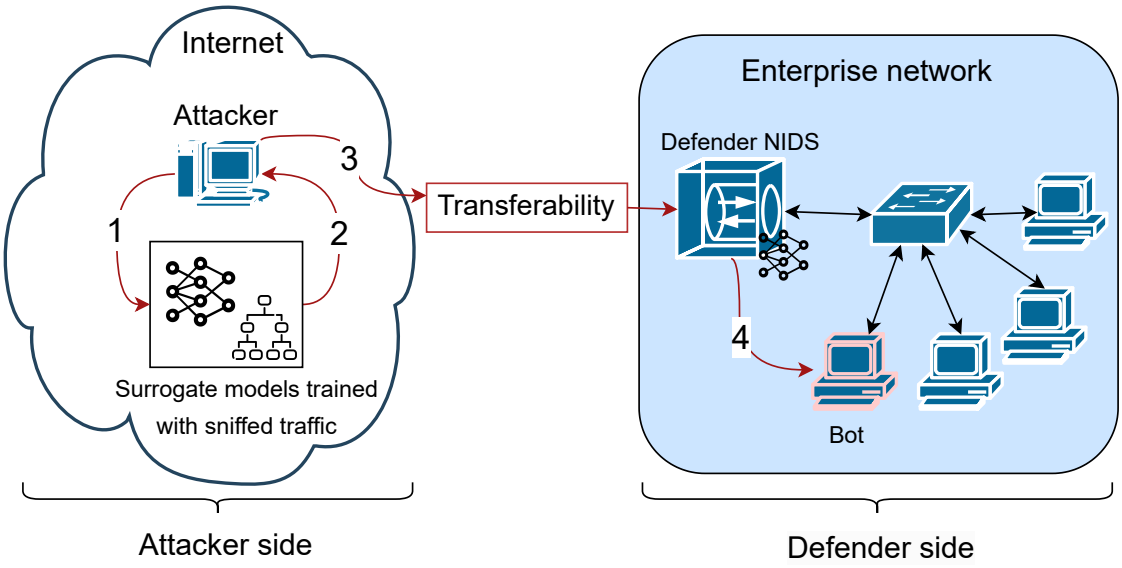


Figure 6.2: An illustration of the adversarial botnet traffic generation steps

As shown in Figure 6.2, There are four steps in the process of creating adversarial botnet traffic. During step 1, the attacker generates adversarial traffic that is specifically designed to bypass the surrogate models that the attacker previously trained using sniffed traffic. The attacker then receives and analyzes the adversarial traffic that managed to avoid detection by the surrogate models during step 2. During step 3, the attacker uses the transferability property to send adversarial botnet traffic to the defender NIDS. In step 4, the adversarial botnet traffic that successfully bypassed the defender NIDS will arrive at its final destination, the bot.

6.3.2 Datasets

To perform reproducible experiments, the CTU-13 [126] and CSE-CIC-IDS2018 [108] datasets were used to provide results that could be comparable to multiple datasets as well as a wider range of attacks.

Other than botnets, CTU-13 and CSE-CIC-IDS2018 contain a variety of attack types. Because our research focuses solely on botnet attacks, all other attack types were removed to create a dataset containing only botnet attack records. To avoid incorporating potential biases when creating these new datasets, we relied on the work done by Venturi et al. [4]. Features were filtered to keep only those consistent for the study of botnet attacks and common to both datasets used. These features are those described in Table 6.1. Regarding CTU-13, the botnet attacks considered in this work are Neris, Virut, and Rbot. Other attacks were not included because they do not have enough malicious instances to provide consistent results. For CSE-CIC-IDS2018, after initially being indistinguishable in the original dataset, the Zeus and Ares botnet attacks were extracted into the same dataset (Zeus & Ares).

To ensure the practicality of the present work, the CTU-13 and CSE-CIC-IDS2018 datasets were divided into two equivalent datasets and stratified according to the labels. These datasets are equivalent in terms of size and distribution, which represents more than 32,000 instances for each side. The first is used for training and evaluation of the model used by the defender. The second is used by the attacker to train the surrogate model independently. The attacker can obtain this data by sniffing the network. This is particularly possible in the case of a botnet scenario, as the attacker communicates bidirectionally with the infected device. The datasets for each side (defender and attacker) are separated into a training dataset and a test dataset with proportions of 75% and 25% respectively. Each training and test data subset is evenly split in terms of malicious and benign traffic. The datasets are separated in this manner to have the most balanced representation so as to avoid the problem of unbalanced data given that it is out of the scope of this study. The attacker and defender datasets thus follow similar but not identical distributions since they are not the same datasets. The arrangement of the datasets is illustrated in Figure 6.3.

6.3.3 Preprocessing

General preprocessing has already been performed on the dataset provided by Venturi et al. [4], resulting in clean data where outliers, empty values, and non-TCP traffic were removed. Data filtering and "one-hot encoding" of categorical features

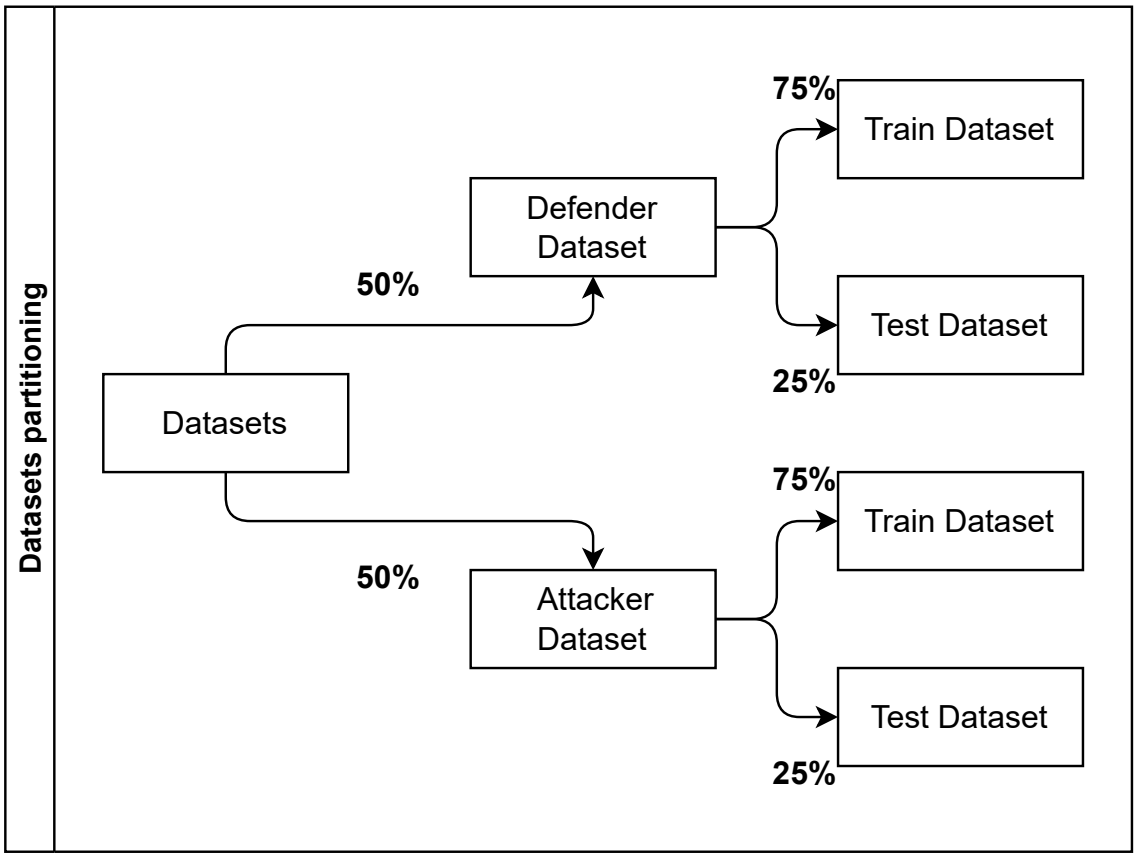


Figure 6.3: Partitioning of the datasets for experiments

were carried out. This encoding transforms the categorical data into a binary format that can be exploited by the model. However, some inconsistencies are present in the dataset provided by Venturi et al. [4]. By default, when the "OutBytes" and "InBytes" features are set to 0, the "RatioOutIn" feature is set to the maximum value present in the data set, simulating an infinite value. Since the ratio in this case is $0/0$, we chose to replace it with 0 instead of a pseudo-infinite value, representing a zero byte exchange.

For training the neural network algorithms, some additional preprocessing was applied to the training and test data. First, the data were normalized using a minmax scaling method, transforming the data to be projected to a value between zero and one, according to Eq. 6.1. Then, the labels undergo a one-hot encoding to transform them to binary values so that they can be processed by the MLP.

$$X\_scaled = \frac{X - min(X)}{max(X) - min(X)} \tag{6.1}$$

where X is an instance and X_scaled is the normalized instance.

6.3.4 ML-based NIDS

As neural networks are increasingly used in the context of NIDS based on machine learning algorithms, a couple of them are chosen, as well as more classical ML algorithms. On the defender side, the defender uses a Multilayer Perceptron (MLP), Random Forrest (RF) and K-Nearest Neighbors (KNN) algorithm as a model for his IDS. For the attacker, the same algorithms are chosen with different parameters and hyperparameters and trained with a different dataset. These three ML models were chosen in our work given their popular use in the IDS community [127–129]. All these algorithms follow the same training and testing process, as shown in Figure 6.4.

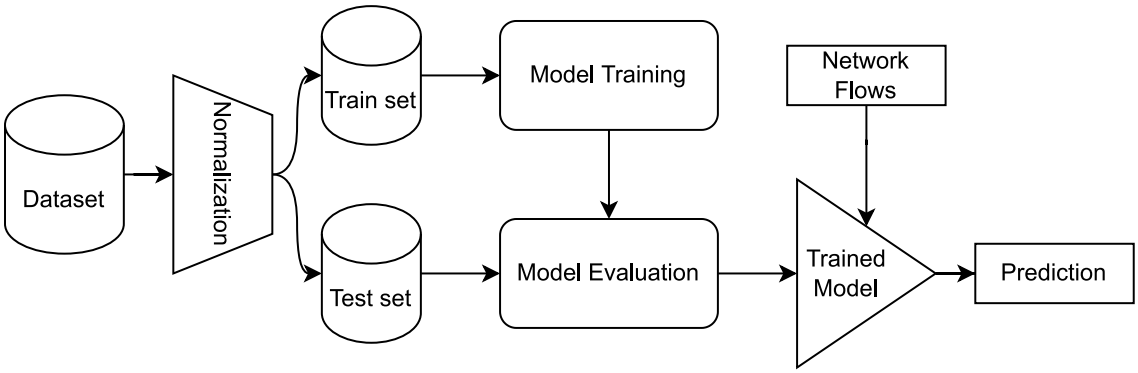


Figure 6.4: Machine Learning pipeline for both attacker and defender

Meta-parameters are chosen randomly to avoid having the same model parameters between the attacker and defender. Examples of meta-parameters are the the number of neighbors k defined in the KNN algorithm, the number of hidden layers or neurons in the DNN, or the number of estimators used in the RF. The choice to use several algorithms allows for more comparable results, especially in the case of transferability. Having several ML models allows for a better understanding of the impact of transferability of adversarial network traffic, both intra and cross-transferability. The meta-parameters of all models can be seen in Table 6.2.

6.3.5 Proposed evasion attacks

In order to generate adversarial perturbations, which are added to the malicious instances to make them benign, we propose two evasion attack variations formulated in Eq. 6.2 and Eq. 6.3 .

Table 6.2: Meta-parameters of the selected ML models

Classifier	Attacker Parameters	Defender Parameters
MLP	**Hidden Layers (HL) = 3** **Neurons by HL = 128** Activation = ReLU Optimizer = Adam	**Hidden Layers (HL) = 2** **Neurons by HL = 256** Activation = ReLU Optimizer = Adam
Random Forest	**Nb estimators = 300** Criterion = Gini Bootstrap = True	**Nb estimators = 200** Criterion = Gini Bootstrap = True
KNN	**Nb neighbors = 5**	**Nb neighbors = 3**

$$x^t_{adv}(f) = Proj[x^{t-1}(f) + sign(benign\_mean(f) \\ -x^0(f)) * (c * t) * mean\_ratio(f)] \tag{6.2}$$

where $x^0(f)$ is the initial value of the targeted network factor f in the instance, the sign function specifies the direction of the perturbation, c is a multiplicative coefficient that regulates the perturbation rate generated at each step t, $benign\_mean$ is the mean of the benign set of the targeted network factor f that the attacker can obtain from sniffing network traffic, $mean\_ratio$ is the ratio between the mean of the malicious set and the benign set of the targeted network factor f, and $Proj$ is a projection function that projects the values of modified features that violate syntactic and semantic constraints into the space of valid values. These modified features are only network factors that the attacker can manipulate directly or indirectly, as represented by the green and yellow groups in Table 6.1. If the attacker manipulates one of the modifiable features shown in Table 6.1 with green color, the $Proj$ function will make sure that the dependents features will change values accordingly. For instance, changing the flow duration by the attacker will induce a change in "total packet by second" feature which equals number of packets divided by the flow duration. In nutshul, the projection function is what allows, when features are modified, to ensure that the domain constraints are respected. This enables the adversarial instances created to be valid and reversible in the targeted domain. In this manner, the malware's intended behavior is preserved.

$$x^t_{adv}(f) = Proj[x^{t-1}(f) + sign(benign\_mean(f) \\ -x^0(f)) * (c * t) * |mean\_diff(f)|] \tag{6.3}$$

where $mean\_diff$ is the difference between the mean of the benign set and the malicious set of the targeted network factor f. In this case, $mean\_diff$ is an absolute value to avoid influencing the direction of added perturbations during adversarial generation.

Although both methods yield similar results, we decided to use the second method (i.e. mean difference method as in Eq. 6.3) to conduct the experiments. This choice is purely for illustrative purposes, as the mean difference method is more intuitive and can be illustrated by Figure 6.5 which shows how a malicious instance tends to become benign, and thus adversarial. Mean is the average of the benign data. The figure is shown in two dimensions for the sake of simplicity, but in reality the manipulation space is much larger because many factors are being manipulated.

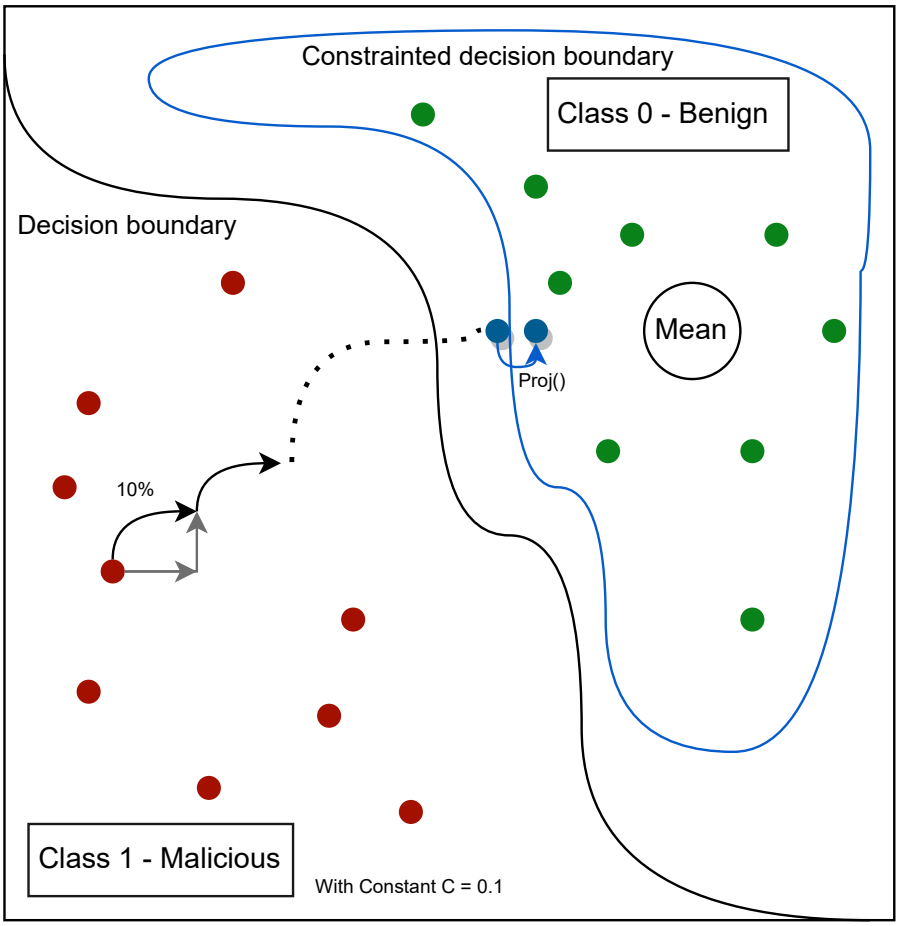


Figure 6.5: 2D illustration of a malicious instance transformation with the mean difference method

6.3.6 Adversarial instances generation

To generate adversarial instances, the process illustrated in Figure 6.6 is followed. During this process, the previously defined Eq. 6.3 is applied during the execution of the adversarial generation step described in Algorithm 1.

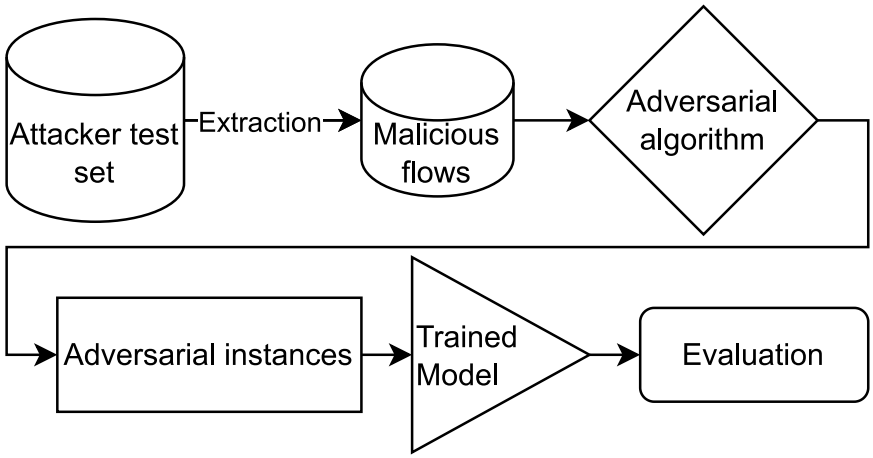


Figure 6.6: Process to generate adversarial instances

Algorithm 1 Crafting adversarial examples for flow-based IDS

procedure CRAFTADVEX(x) ▷ where x is a malicious flow
  $x_{adv} \leftarrow x$
  $t \leftarrow 1$
  $m \leftarrow mean\_difference()$ ▷ or mean_ratio()
  repeat
    for $mask \leftarrow mask_1, ..., mask_{15}$ **do**
      $\epsilon \leftarrow sign[benign\_mean(f) - x^0(f)] * (c * t) * m(f)$
      $\epsilon \leftarrow \epsilon * mask$
      $x_{adv} \leftarrow x_{adv} + \epsilon$
      $x_{adv} \leftarrow Proj(x_{adv})$
      if $predict(x_{adv}) == benign$ **then**
        return x_{adv}
      end if
    end for
    $t \leftarrow t + 1$
  until $predict(x_{adv}) == benign$
end procedure

procedure PROJ(x_{adv}) ▷ applying domain constraints to x_{adv}
  $x_{adv} \leftarrow ApplySyntacticConstraints(x_{adv})$
  $x_{adv} \leftarrow ApplySemanticConstraints(x_{adv})$
  return x_{adv}
end procedure

It is worth mentioning that this algorithm applies increasingly large perturba-

tions to the initial malicious stream at each iteration, whenever the new adversarial instance is not classified as benign. Both formulas guarantee small initial perturbations and are modulated by the multiplicative constant c.

To select the combinations of attacks (presented in Section 4) that should be applied to manipulate the network traffic flow, a method of masks is employed where each corresponds to a combination of the manipulable factors (i.e., the total number of packets, the transmission time of the packets, and the number of outgoing or incoming bytes in those packets). At each iteration, the masks are applied in a gradual manner by multiplying them by the previously generated perturbations using Eq. 6.3 and Eq. 6.2. This ensures that only the perturbations needed at that current stage are applied to the selected factors. These masks follow a binary logic and represent 15 combinations. The representation of these masks with their corresponding manipulable factors is shown in Table 6.3.

Table 6.3: Features used by combination with their corresponding mask

Combinaison	**Mask**	**Target factor**
1	0001	Duration
2	0010	TotPackets
3	0011	Duration, TotPackets
4	0100	InBytes
5	0101	InBytes, Duration
6	0110	InBytes, TotPackets
7	0111	InBytes, TotPackets, Duration
8	1000	OutBytes
9	1001	OutBytes, Duration
10	1010	OutBytes, TotPackets
11	1011	OutBytes, TotPackets, Duration
12	1100	OutBytes, InBytes
13	1101	OutBytes, InBytes, Duration
14	1110	OutBytes, InBytes, TotPackets
15	1111	OutBytes, InBytes, TotPackets, Duration

Once perturbations are applied at each stage, syntactic and semantic constraints are enforced, through the $Proj()$ function, to respect the underlying logic of the malicious network communication. This projection function has a couple of actions: it allows to apply semantic and syntactic constraints to make adversarial instances valid. To do this, this function recalculates the dependent factors (i.e., the yellow group in Table 6.1) that depend on the directly manipulatable factors (i.e., the green group in Table 6.1). Furthermore, when a value obtained exceeds the maximum

value listed in the entire test set, this value is projected to this maximum value to ensure that it is transcribed in its appropriate scope. Given that each dependent feature (in yellow) is linked to a modifiable feature (in green) via an easy and intuitive formula explained in Table 6.1 (e.g., RationOutIn = OutBytes/InBytes), We chose not to list all of the formulae in Algorithm 1, instead providing a list of the corresponding dependent features as shown in Table 6.4, which are updated when an attacker manipulates one of the modifiable features to ensure semantic constraints are respected. This algorithm is feasible in that the attacker acts directly on the network traffic and not on the features. This means that it follows a so-called traffic-based methodology, which is more realistic than the feature-based one because the attacker only needs limited information about the defender, such as information about the network traffic he plans to attack. This allows us to assume black-box knowledge, which is not the case with feature-based manipulation, which requires white-box knowledge, such as access to the internal architecture of the intrusion detection system, its parameters, and the dataset used to train the ML model. It can be noted that our adversarial algorithm was designed to consider two objective functions: one to minimize the amount of modification in each factor and the other to minimize the number of modified factors.

Table 6.4: When an attacker manipulates one of the modifiable features, the corresponding dependent features are updated using the Proj() function to ensure semantic constraints are respected.

Modifiable features	Corresponding dependent features updated using Proj fucntion
Dur	BytesPerSec, PktsPerSec
Out/InBytes	TotBytes, BytesPerPkt, BytesPerSec, RatioOutIn
TotPkts	BytesPerPkt, PktsPerSec

6.3.7 Strengthening adversarial robustness

Since ML algorithms are inherently vulnerable to adversarial examples, it is necessary to find various approaches to defend them. After reviewing a range of defenses found in the literature, we found that many of them are not fully adapted to domain-constrained systems such as NIDS and that there is still room for improvement [47, 48], although some of them have the potential to reduce the impact

of adversarial instances [22,103]. For this reason, a defense is proposed in this work.

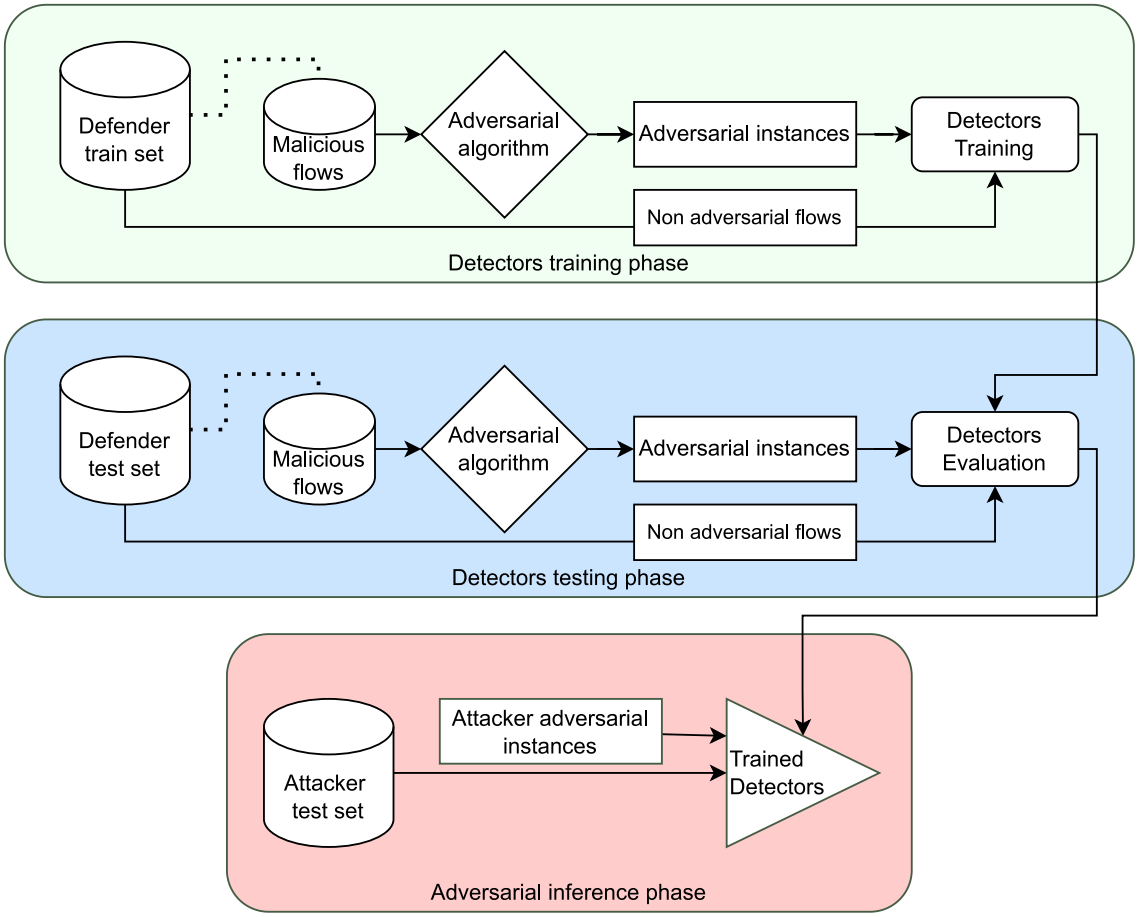


Figure 6.7: Proposed defense process

As shown in Figure 6.8, the proposed reactive defense follows the process illustrated in Figure 6.7. The defense itself is inspired by adversarial training [22], adversarial detection [103], and bagging as an ensemble method. The role of this defense is to act as an adversarial detector (filter) that is placed before the NIDS to prevent adversarial instances from being able to reach and fool the NIDS. The main idea of this defense is to train three MLP models in parallel, each taking as input a specific group of features in order to detect adversarial instances. There are thus three groups corresponding to features that can be directly manipulated by the attacker, features that depend on the first group, and features that cannot be modified. This arrangement corresponds to the manipulability of features in a realistic context, i.e., features that can be modified by the attacker. The instances used to train the three sub-detectors are divided into two classes: the first one contains benign and malicious instances, having been concatenated together and relabeled

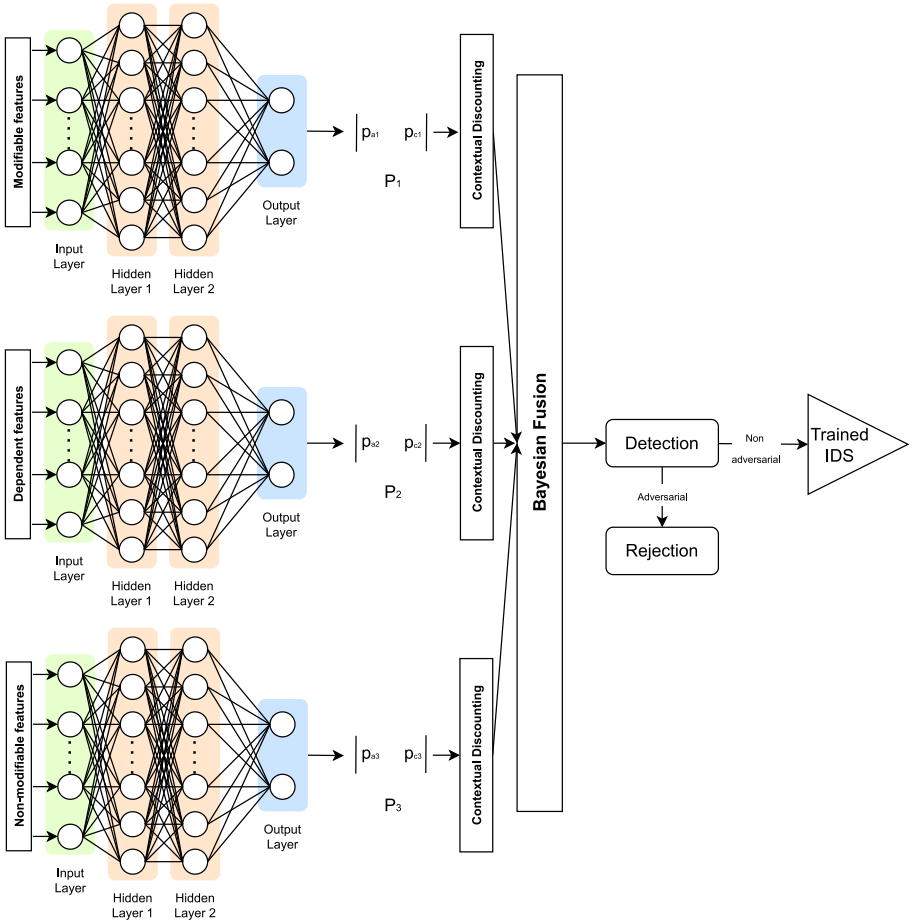


Figure 6.8: Proposed defense approach using MLP sub-detectors

as clean (i.e., non-adversarial), the second class contains adversarial instances previously created based on malicious instances exclusively. Table 6.5 illustrates the different data types used to better understand this distinction.

Table 6.5: Description of datasets and labels used by the detectors

Dataset Type	**Description**
Benign	The network traffic that contain benign communications.
Malicious	The network traffic that contain a botnet attack.
Clean	The concatenation of benign and malicious traffic.
Adversarial	The generated adversarial instances produced with an adversarial algorithm using malicious traffic.

Each of the three feature groups is assigned a particular weight. These weights are set during model training based on the overall detection rate of each detector. Once the predictions have been made by each sub-model for each test instance, they undergo a contextual discounting and then a Bayesian fusion, as defined in Eq. 6.4, where the prediction matrix is multiplied by the corresponding weights, and then

the three resulting matrices are summed. Once this step is complete, the new values are normalized to obtain the final probabilities, where P_a represents the probability that the instance is adversarial, and P_c is the probability that it is clean. The result is a final prediction of whether an instance is adversarial or not, and thus is subject to rejection by the detector.

$$
\begin{aligned}
P_a &= \frac{\sum_{i=1}^{3}(P_{a_i} * w_i)}{\sum_{i=1}^{3}(P_{a_i} * w_i) + \sum_{i=1}^{3}(P_{c_i} * w_i)} \\
P_c &= \frac{\sum_{i=1}^{3}(P_{c_i} * w_i)}{\sum_{i=1}^{3}(P_{a_i} * w_i) + \sum_{i=1}^{3}(P_{c_i} * w_i)}
\end{aligned}
\tag{6.4}
$$

Figure 6.8 illustrates the use of MLP as a sub-detector, but it should be noted that these detectors can use different machine learning algorithms. This defense could even be used in a stacking scheme involving detectors, each using some specific machine learning algorithms.

6.4 Evaluation

This section discusses the findings of several experiments. First, those concerning the performance of the initial attacker and defender models are discussed based on different metrics, namely precision, recall and F1 score. These results concern the models trained with the CSE-CIC-IDS2018 and CTU-13 datasets. In a second step, results regarding the performance of the models in adversarial contexts are discussed. A preliminary study of the models trained with the CSE-CIC-IDS2018 dataset is investigated to see the impact of transferability between the same and different models as well as the training data. A study of the time taken by each attack is also considered, as well as an analysis of the differences in perturbation between the initial malicious instance and the adversarial instance. Then, a general comparison is made on the performance of the proposed adversarial generation algorithm across the botnet attacks present in each dataset. The last section includes the results of the proposed defense against the adversarial instances generated by the previously proposed evasion attack algorithm.

6.4.1 Initial performance of ML-IDS models in clean settings

To evaluate the initial performance of ML-IDS models trained in clean (i.e. non-adversarial) settings on both the attacker and defender sides, several metrics are used, namely: recall, precision, and f1-score. ML-IDS models are requested to perform a binary classification to distinguish malicious from benign traffic in clean settings.

As shown in Figure 6.9, the results for the initial performance of the attacker-side trained ML models yield metrics of 100% for all models trained with CSE-CIC-IDS2018. In the case of CTU-13, they are less significant. Nevertheless, all models have metrics above 96%. These results regarding the performance of ML-IDS on both datasets are comparable to those found in the literature [130–132].

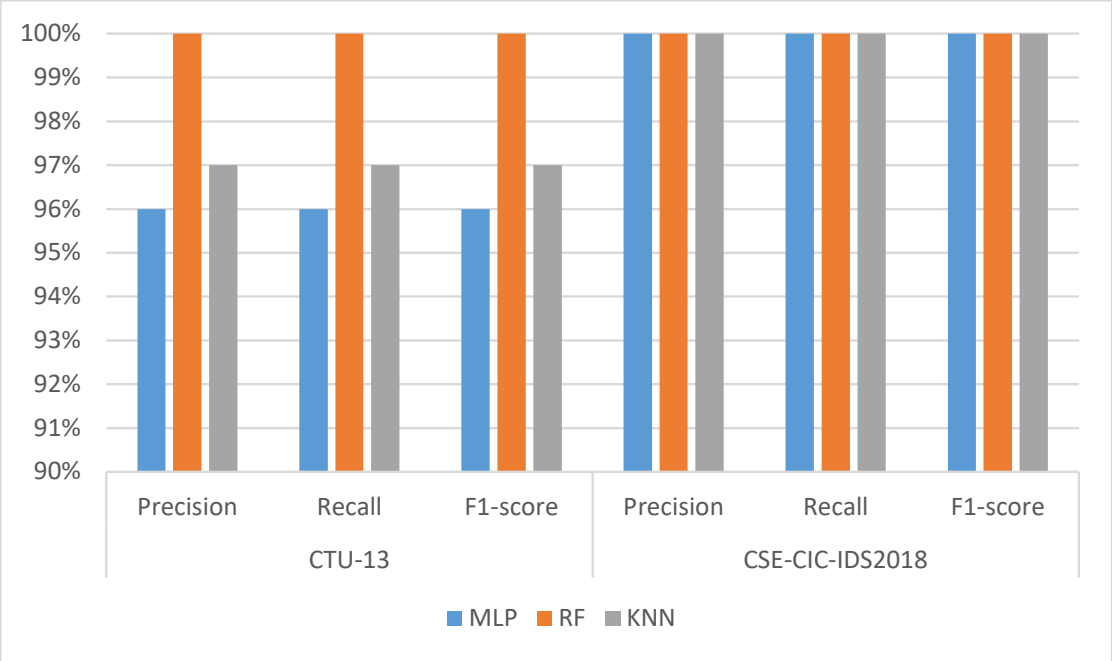


Figure 6.9: Performance of the attacker-side trained ML models on CTU-13 and CSE-CIC-IDS2018 in clean settings

These initial results show good performance in general. Models trained with CTU-13 perform somewhat less well than those trained with CSE-CIC-IDS2018. This may be due to the fact that CSE-CIC-IDS2018 contains only two botnet attacks, while CTU-13 contains five, forcing the models to expand their decision boundary to try to correctly classify all types of attacks into a single class (i.e., malicious).

Regarding the performance of the defender-side trained ML models, presented in Figure 6.10, we can see slight variations in the metrics for the models trained with CTU-13. These variations are very small and of the order of 1% maximum. We can note that the model using a random forest gives a metric of 100%. For the models trained with CSE-CIC-IDS2018, the metrics are all at 100%, which gives identical performance to that of the attacker-side trained ML models. In general, the same observations can be made for the defender models, as the results are relatively similar.

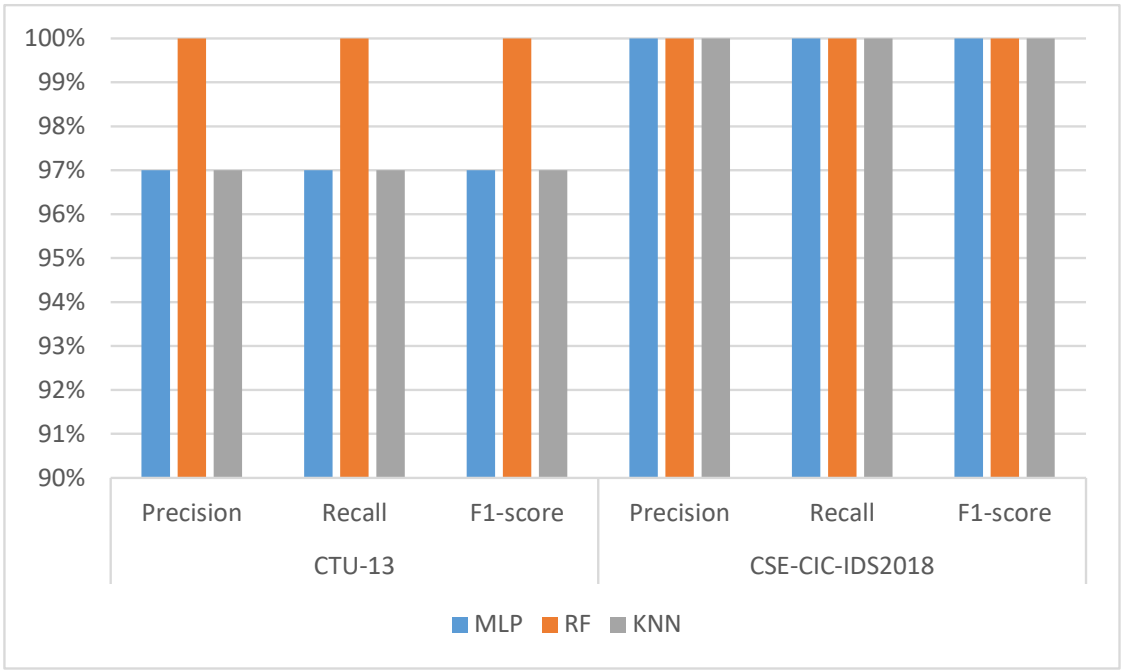


Figure 6.10: Performance of the defender-side trained ML models on CTU-13 and CSE-CIC-IDS2018 in clean settings

6.4.2 Performance of ML-IDS models in adversarial settings

To study the impact of adversarial instances generated by our algorithm 1, as well as the effectiveness of transferring these adversarial instances created by the attacker to the models trained by the defender, the first experiment focuses on the CSE-CIC-IDS2018 dataset containing the Zeus & Ares botnet attacks. To measure the impact of adversarial instances, the detection rate metric, also called recall, is used **??**. It measures the rate of adversarial instances detected by the ML-IDS as malicious. To do this, the attacker first generates adversarial instances for each

model trained on his side (i.e., MLP, RF, and KNN). The adversarial instances generated for one model will be sent to the other models to evaluate the transferability property between the models trained by the attacker. The defender-trained models are different from the attacker-trained models in two aspects: i) they have different hyperparameters; ii) they are trained with different datasets. We test the effect of using the same or different hyperparameters on the transferability of adversarial instances from the attacker's models to the defender's models.

It should be noted that the results of the experiments in the various tables always compare equal proportions for the defender and attacker with different data (i.e. different contents). As shown in Figure 6.3, the size of the training (75%) and test data (25%) is the same for the attacker and defender because the original dataset is divided in half. Since the size of the data is the same, by saying that the data is different, we invariably mean its content.

Table 6.6a illustrates the performance of the adversarial transferability property between the attacker's trained models. It is worth mentioning that all the attacker's models are trained with the same data. As we can see, the diagonal values of the table are all zeros. This is to be expected since the adversarial instances were designed based on the decision boundary of this model, so testing on the same model will give a detection rate of 0%. We also notice that the adversarial instances generated based on MLP were able to drop the detection rate of RF and KNN to 0%, while the adversarial instances based on these two models were able to drop the detection rate to about 50%. Finally, we can see that on average, the detection rate dropped to 21.9%, which is a rather satisfactory result for the attacker.

Table 6.6b represents the impact of adversarial instances, generated using the attacker's model, against the defender's models. The particularity is that the defender uses, in this case, the same hyperparameters as the attacker's models. Only the dataset used during training changes. This allows, compared to Table 6.6a , to see the effect of intra-transferability through the same models and the same hyperparameters using a different dataset for training, but also of inter-transferability through different models with the same hyperparameters. This gives concrete information about the impact of the training data and the model used on the transferability of adversarial instances. Compared to Table 6.6a, Table 6.6b shows slight

changes in the effect of intra-transferability (i.e., the same model) as seen in the diagonal values (1% for RF and 3% for KNN) while the effect of cross-transferability remains similar in both tables. These results indicate that the attacker does not need to train his models with the same data as the defender, data with a similar distribution will be sufficient to make the transferability property work effectively.

Table 6.6c again shows the adversarial performance against the defender models. However, although the models use the same learning algorithms, this time they have different hyperparameters. This allows us to show the effect of intra-transferability and cross-transferability on different models with different hyperparameters as well as different training data sets. This table is therefore the one that provides the closest insight to the reality because, here, the attacker's knowledge is extremely limited since he neither knows the model used, nor the parameters or hyperparameters of the model, nor the data used to train the defender's model. This demonstrates the effect of transferability in a realistic context. The results provide a fairly similar detection rate between Table 6.6b and Table 6.6c, indicating that not using the same hyperparameters as the defender does not have a significant impact on the transferability of the adversarial instances.

From Table 6.6, we can see that not knowing the training data, hyperparameters, or model used by the defender does not prevent the attacker from creating adversarial instances and successfully transferring them to the defender's ML-IDS.

The set of sub-tables in Table 6.7 represents the average and maximum perturbation difference between the malicious instances and their adversarial counterparts using the CSE-CIC-IDS2018 dataset. Each of these tables represents the ML model used by the adversarial generation algorithm 1 to craft the perturbations to be added to the manipulatable factors. Note that the duration factor "Dur" is expressed in seconds.

The main observation that can be made from this set of tables is that the perturbations are rather small for all the manipulable factors and therefore feasible in realistic scenarios. It should be noted that these perturbations are influenced by the number of steps and the regulation coefficient c present in the adversarial generation algorithm 1 and driven by the mean of each manipulable factor.

After confirming from Table 6.6 and Table 6.7, that the transferability property of

Table 6.6: Performance of the adversarial transferability between ML-IDS models in term of detection rate

(a) Performance of the adversarial transferability between the attacker's trained models

Attacker side				
Attacker\Attacker	MLP	RF	KNN	**Mean**
MLP	0%	0%	0%	0%
RF	50%	0%	48%	33%
KNN	50%	49%	0%	33%
		Grand Mean		**21.9%**

(b) Performance of the adversarial transferability from the attacker to the defender models (having the same attacker's hyperparameters)

Defender side (same as attacker's hyperparameters)				
Attacker\Defender	MLP	RF	KNN	**Mean**
MLP	0%	0%	0%	0%
RF	50%	1%	48%	33%
KNN	50%	49%	3%	34%
		Grand Mean		**22.3%**

(c) Performance of the adversarial transferability from the attacker to the defender models (having different hyperparameters from the attacker's ones)

Defender side (different hyperparameters)				
Attacker\Defender	MLP	RF	KNN	**Mean**
MLP	0%	0%	0%	0%
RF	50%	1%	48%	33%
KNN	50%	49%	4%	34%
		Grand Mean		**22.4%**

adversarial instances allows the attacker to create an adversarial malicious instance to evade the defender's IDS without needing to know its internal architecture while ensuring that the needed perturbations are small enough to be feasible in realistic scenarios, we proceed to create adversarial instances for four botnet attacks, namely Neris, Rbot and Virut from the CTU-13 dataset and Zeus & Ares from the CSE-CIC-IDS2018 dataset. Note that the attacker and defender models are trained on different data as shown in Figure 6.3 and have different hyperparameters as shown

Table 6.7: The maximum and average perturbation difference between the malicious and adversarial instance on CSE-CIC-IDS2018

Attacked model	MLP			
Manipulated factor	Dur	InBytes	OutBytes	TotPkts
Average Perturb Diff.	0.075	55.251	10.286	0.001
Max Perturbation Diff.	0.218	348.368	1126.160	0.498

Attacked model	RF			
Manipulated factor	Dur	InBytes	OutBytes	TotPkts
Average Perturb Diff.	0.071	55.251	10.286	0.001
Max Perturbation Diff.	0.218	348.368	1126.160	0.498

Attacked model	KNN			
Manipulated factor	Dur	InBytes	OutBytes	TotPkts
Average Perturb Diff.	0.044	0.813	179.585	0.000
Max Perturbation Diff.	0.073	116.123	375.387	0.000

in Table 6.2.

Table 6.8 represents the detection rate of the attacker's models against adversarial botnet attacks. For each botnet type, the attacker generates adversarial instances corresponding to malicious botnets based on the decision boundaries of one of his models using the adversarial generation algorithm 1, and then tests these generated adversarial instances on other ML models. On average, we can see that the attacker managed to reduce the detection rate to 15.8%, 5.3%, 22.2% and 21.9% for Neris, Rbot, Virut and Zeus & Ares respectively.

On the other hand, Table 6.9 represents the detection rate of the defender's models against adversarial botnet instances generated previously by the attacker. On average, we can see that the attacker managed to reduce the detection rate of the models trained by the defender to 23.0%, 8.1%, 37.8% and 22.4% for Neris, Rbot, Virut and Zeus & Ares respectively.

Using Neris as an example, the attacker generated adversarial instances corresponding to this malware traffic using the decision boundaries of his trained RF model. When testing the effectiveness of his generated adversarial instances on his other trained ML models (i.e., MLP and KNN), he achieves a detection rate of 34%

Table 6.8: Detection rate of the attacker's models against adversarial botnet attacks

Attacker side

Neris				
Attacker/Attacker	MLP	RF	KNN	**Mean**
MLP	0%	19%	7%	9%
RF	34%	0%	22%	19%
KNN	26%	34%	0%	20%
		Grand Mean		**15.8%**

Rbot				
Attacker/Attacker	MLP	RF	KNN	**Mean**
MLP	0%	5%	0%	2%
RF	18%	0%	16%	11%
KNN	2%	7%	0%	3%
		Grand Mean		**5.3%**

Virut				
Attacker/Attacker	MLP	RF	KNN	**Mean**
MLP	0%	69%	1%	23%
RF	28%	0%	23%	17%
KNN	7%	72%	0%	26%
		Grand Mean		**22.2%**

Zeus & Ares				
Attacker/Attacker	MLP	RF	KNN	**Mean**
MLP	0%	0%	0%	0%
RF	50%	0%	48%	33%
KNN	50%	49%	0%	33%
		Grand Mean		**21.9%**

and 22%, respectively, as shown in Table 6.8 and 0% when tested on RF, since these instances are specifically designed to fool this particular trained model. The attacker then sends these RF-based generated adversarial instances to the defender IDS. The detection rates for the three ML models trained by the defender are 41%, 19%, and 23% for MLP, RF, and KNN, respectively, as shown in Table 6.9, resulting in an average of 28% for the defender models. This is a relative success for the attacker since their malicious traffic is only detected 28% of the time and has almost a three-quarter chance of evading the defender's installed intrusion detection systems in a black box setting.

Comparing Table 6.8 and Table 6.9, we can observe that the adversarial instances generated by the attacker are, on average, more efficient on his own models

than on the models of the defender who uses different training data and hyper-parameters. It can also be observed that intra-transferability has more impact than cross-transferability. Even if this loss is not negligible, the results show good performance on average, both through intra- and cross-transferability.

Table 6.9: Detection rate of the defender models against adversarial botnet attacks

Defender side

Neris				
Attacker/Defender	MLP	RF	KNN	**Mean**
MLP	18%	20%	8%	15%
RF	41%	19%	23%	28%
KNN	35%	35%	8%	26%
		Grand Mean		**23.0%**

Rbot				
Attacker/Defender	MLP	RF	KNN	**Mean**
MLP	1%	6%	0%	2%
RF	18%	22%	15%	18%
KNN	2%	8%	1%	4%
		Grand Mean		**8.1%**

Virut				
Attacker/Defender	MLP	RF	KNN	**Mean**
MLP	25%	69%	2%	32%
RF	28%	90%	24%	47%
KNN	29%	71%	2%	34%
		Grand Mean		**37.8%**

Zeus & Ares				
Attacker/Defender	MLP	RF	KNN	**Mean**
MLP	0%	0%	0%	0%
RF	50%	1%	48%	33%
KNN	50%	49%	4%	34%
		Grand Mean		**22.4%**

The time taken to generate an adversarial instance for each of the botnet attacks and models is shown in Table 6.10. It seems that MLP is, for each of the attacks, the algorithm that consumes the most time to generate adversarial instances, followed by RF, which is an ensemble method. On the other hand, KNN seems to be the fastest algorithm to generate adversarial instances for the four botnet attacks, which would make it an interesting option if a trade-off between efficiency and time had to be made.

Table 6.10: Time taken (in seconds) to generate 3000 adversarial instances for all botnet attacks using Algorithm 1

Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	Mean
MLP	32.48	10.21	11.69	15.22	17.40
RF	16.95	5.48	9.55	6.85	9.71
KNN	3.17	1.92	1.95	0.14	1.79
			Grand Mean		9.63

6.4.3 Proposed Defense effectiveness

As we discussed in section 6.4.2, the attacker is able to evade the defender's NIDS with a high success rate (i.e., a low detection rate) by relying solely on the transferability property of adversarial instances. Our proposed defense aims to negate the effect of the transferability property by adding an adversarial detector to filter out adversarial instances, allowing the NIDS to process only clean traffic.

We first evaluate the performance of the proposed adversarial detector and its sub-detectors by generating adversarial instances based on the CTU-13 and CSE-CIC-IDS2018 datasets. During this analysis, each sub-detector is evaluated as shown in Figure 6.7. To measure this performance, various metrics are used: recall as defined in Eq. **??**; precision as defined in Eq. **??**; and F1-score as defined in Eq. **??**.

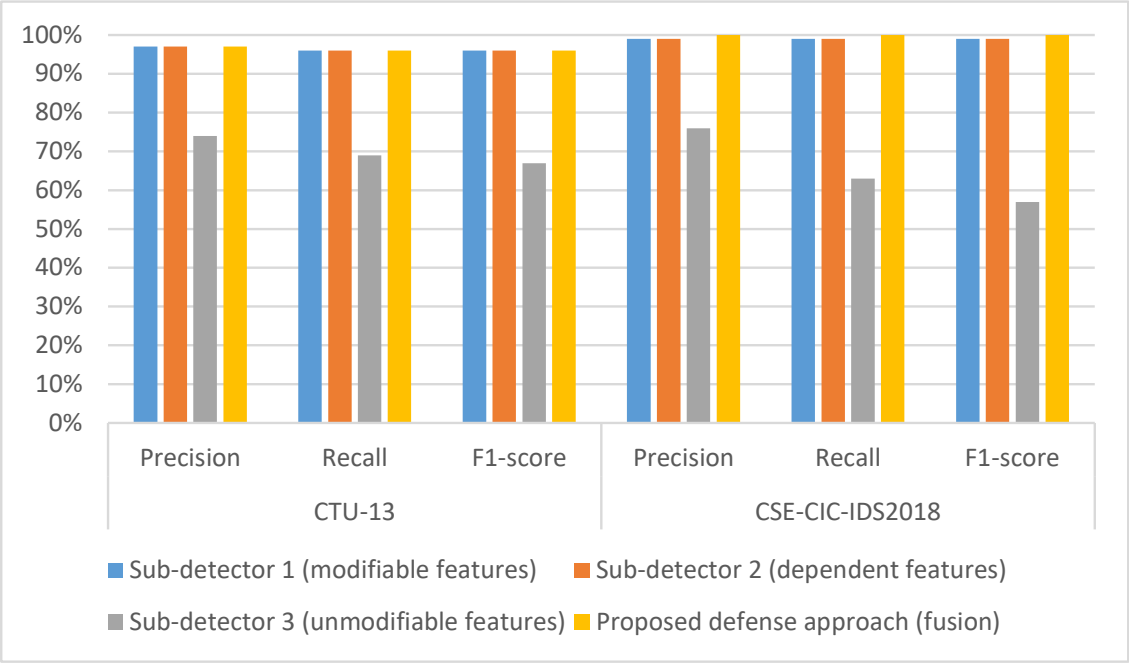
As shown in Figure 6.11, we can see that the first two sub-detectors are performing quite well, reaching more than 96% for each metric for CTU-13 and 99% for CSE-CIC-IDS2018, while the last one seems to be less efficient, with performances around 70%. We can also observe that the final detector performance after Bayesian fusion provides good performances, reaching 97% for CTU-13 and 100% for CSE-CIC-IDS2018.

The poorer performance of the third detector can be explained by the fact that it is trained with the group of non-modifiable features. Since the value of these features does not change, it seems that the detector is not able to distinguish adversarial instances from clean instances, thus behaving randomly.

We also note that the fusion of the three detectors slightly improved the overall performance of the proposed defense compared to the individual detectors. The inferior performance of the third detector does not seem to diminish the performance

of the proposed defense due to the contextual discounting mechanism, which allows the performance of each individual detector to be taken into account during the fusion stage.

Figure 6.11: Performance of our proposed defense against adversarial traffic



After confirming that our proposed adversarial detector works as intended, we integrate it into our threat scenario. The attacker creates adversarial instances of the four adversarial botnet malware types, namely Neris, Rbot, Virut, and Zeus & Ares, using the adversarial instance generation algorithm on three ML model decision boundaries, namely MLP, RF, and KNN. The defender, on the other hand, uses an MLP-based NIDS to detect the malicious traffic. In this experiment, attacks are launched twice: the first with the defender's NIDS protected by our proposed adversarial defense and the second without protection. This is done to evaluate the effect of using such a defense mechanism on the performance of the defender's NIDS. The detection rate metric, also known as recall, is used to measure the performance of the NIDS in identifying malicious and benign traffic, but it is also used to assess the performance of the proposed adversarial detector in identifying adversarial and clean traffic. The results are presented in Table 6.11, consisting of three sub-tables:

Table 6.12a shows the performance results of the defender's MLP-based NIDS against adversarial instances with no adversarial defense. As already shown in Table

6.9, we see that the attacker has successfully decreased the performance of the defender's NIDS by dropping the average detection rate to 24.8%.

Table 6.11b reports the results regarding the ability of our proposed adversarial defense to detect the adversarial instances generated by the attacker. It can be seen that, on average, the proposed adversarial defense was able to detect 93.4% of the total adversarial botnet traffic sent by the attacker, thus protecting the NIDS from getting evaded by these adversarial instances.

Table 6.11c represents the detection rate of the defender's NIDS protected by our proposed adversarial defense. In fact, it shows the impact of adversarial instances that have made it through our adversarial detector and reached the NIDS. It can be seen from this table that NIDS has, on average, a detection rate of 96.9% for any type of machine learning model used for adversarial generation, across all botnet attacks.

These results indicate that NIDS seems to be significantly more robust once the adversarial detection method is used, going from an average detection rate of 21.3% without defense, as shown in Table 6.12a, to 96.9% when using the proposed adversarial detector. This also shows that NIDS is hardly affected by adversarial instances capable of passing the adversarial detector.

6.4.4 Comparison with existing defensive strategies

Several countermeasures against the evasion attacks have been proposed in the literature. However, many of them have been shown to be ineffective. Therefore, we decided in this work to compare our proposed defense with adversarial training, which is a state-of-the-art defense that has already demonstrated its effectiveness. To prevent adversarial training from undermining the effectiveness of the IDS in its primary task, we implement this defense in an anomaly detection manner, hence the name adversarial training detection.

Indeed, adversarial data may be detected by comparing the classification of two models: a baseline model trained solely on non-adversarial samples and a robust adversarial model trained on both non-adversarial and adversarial samples. When applying the two models to a sample, one may assume that if the two models categorize it differently, the sample is adversarial. In principle, if the provided sample

Table 6.11: Proposed adversarial defense effectiveness

(a) Detection rate of the defender MLP-based NIDS against adversarial instances without adversarial defense

		Defender using MLP-based NIDS without adversarial defense				
	Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	**Mean**
Attacker	MLP	18%	1%	25%	0%	11%
	RF	41%	18%	28%	50%	34%
	KNN	35%	2%	29%	50%	29%
				Grand Mean		24.8%

(b) Detection rate of our proposed adversarial defense

		Our proposed adversarial defense performance				
	Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	**Mean**
Attacker	MLP	95%	99%	98%	100%	98%
	RF	88%	94%	91%	84%	89%
	KNN	90%	99%	97%	86%	93%
				Grand Mean		93.4%

(c) Detection rate of the defender MLP-based NIDS against adversarial instances with our proposed adversarial defense

		Defender using MLP-based IDS with our adversarial defense				
	Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	**Mean**
Attacker	MLP	93%	97%	96%	100%	97%
	RF	93%	98%	97%	100%	97%
	KNN	94%	98%	97%	100%	97%
				Grand Mean		96.9%

is not adversarial, the base model and robust model should correctly identify it. If the sample is adversarial, the base model will categorize it wrong, but the resilient model would classify it properly.

Adversarial training detection defense is built by training a second defense-side MLP-based IDS on data sets with an equal mix of adversarial and non-adversarial samples. We then use the same adversarial instances created by the attacker to attack both models and record their predictions. We infer a final set of predictions by comparing these two sets of predictions so as to classify each instance as adversarial or non-adversarial.

Table 6.12: Detection rate comparison between adversarial training detection and our proposed adversarial defense

(a) Detection rate of our proposed adversarial defense

		Our proposed adversarial defense performance				
	Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	**Mean**
Attacker	MLP	95%	99%	98%	100%	98%
	RF	88%	94%	91%	84%	89%
	KNN	90%	99%	97%	86%	93%
				Grand Mean		93.4%

(b) Detection rate of adversarial training detection

		Adversarial training detection				
	Model\Botnet	Neris	Rbot	Virut	Zeus & Ares	**Mean**
Attacker	MLP	94%	98%	96%	100%	97%
	RF	87%	93%	90%	84%	89%
	KNN	89%	98%	96%	86%	92%
				Grand Mean		92.6%

In Table 6.12, we can see that the achieved results are decent compared to the other state-of-the-art defenses. Our proposed defense slightly outperforms adversarial training detection with an average detection rate of 93.4% versus 92.6%.

Furthermore, compared to other state-of-the-art defenses, our proposed defense is considered a reactive defense, as it does not change the overall performance of the defender's NIDS or general behavior. Therefore, this defense is a better choice when it comes to preserving the general performance of the model, which is not the case with adversarial training, for example, which reduces the performance of the underlying model.

6.5 Summary

The potential of using NIDS based on machine learning algorithms raises intriguing security issues. Indeed, despite their impressive performance, these ML models are prone to various types of adversarial attacks, especially evasion attacks. Due to their prevalence and feasibility among all adversarial attacks, evasion attacks were considered in this work to generate botnet adversarial attacks capable of evading

the intrusion detection system.

The proposed framework includes two main contributions. The first is a realistic adversarial algorithm capable of generating valid adversarial network traffic by adding small perturbations, thus evading NIDS protection with high probability while maintaining the underlying logic of the botnet attack. To the best of our knowledge, this is the first complete black-box botnet attack that proposes to evade NIDS by exploiting the transferability property, and without using any query method, with very limited knowledge of the target NIDS, which acts on the traffic space, while respecting the domain constraints.

The second component of the proposed framework is a reactive defense that limits the impact of the proposed attack. This defense, inspired by adversarial detection, capitalizes on the fact that it does not change the initial performance of the NIDS since it provides an additional layer of security independent of the model. The proposed defense is considered modular because it uses an ensemble method called bagging yet can use any type of machine learning algorithm. In addition to this ensemble method, it also includes a contextual discounting method that improves the overall performance of the defense.

The results showed that the proposed defense is able to detect most adversarial botnet traffic, showing promising results with respect to state-of-the-art defenses. Since the proposed framework is easily adaptable to other domains, evaluating its performance in other highly constrained domains would be an interesting future work.

Conclusions and perspectives

Conclusion

In this thesis, we explored the effects of adversarial machine learning on cybersecurity systems driven by machine learning models, focusing on intrusion detection systems.

Chapter 1 is a revised review of the state of the art, focusing on the feasibility of adversarial attacks and defenses. We also provide an update on recent contributions to the feasibility of attacks in real-world settings: for each chosen paper, we propose a comprehensive analysis of the real feasibility of the proposed attacks by demonstrating whether or not the domain constraints are respected. Furthermore, we propose an examination of the defenses employed in the studied papers to highlight the advantages and disadvantages of each. Finally, we identify some realistic aspects that should be considered in future studies of the impact of adversarial attacks on intrusion detection systems.

Chapter 2 proposes a novel contextual discounting method based on the reliability of sources and their ability to distinguish between normal and abnormal behavior. An evidential classifier is built using the Dempster-Shafer theory, a general framework for reasoning under uncertainty. The NSL-KDD dataset, which is a significantly revised and improved version of the existing KDDCUP'99 dataset, is used to evaluate the performance of our new detection method. While our approach produced comparable results on the KDDTest+ dataset, it outperformed some other state-of-the-art methods on the more challenging KDDTest-21 dataset.

Chapter 3 investigates the impact of adversarial attacks on deep learning-based intrusion detection systems. Furthermore, the effectiveness of adversarial learning as a defense against these attacks is investigated. The experimental results show that with sufficient perturbation, adversarial examples are able to mislead the intrusion detection system and that the use of adversarial training can improve the robustness of intrusion detection.

In Chapter 4, we investigate the transferability of adversarial network traffic

against several machine learning-based intrusion detection systems. In addition, we analyze the robustness of the ensemble intrusion detection system, which is known for its improved accuracy compared to a single model, against the transferability of adversary attacks. Finally, we examine Detect & Reject as a defensive mechanism to mitigate the effect of the transferability property of adversarial attacks against machine learning-based intrusion detection systems.

In Chapter 5, we aim to design an efficient adversarial detector based on transfer learning and then evaluate the effectiveness of using multiple strategically placed adversarial detectors compared to a single adversarial detector for intrusion detection systems. In our experiments, we implement existing state-of-the-art intrusion detection models. We then attack these models with a set of selected evasion attacks. In an attempt to detect these adversarial attacks, we design and implement multiple adversarial detectors based on transfer learning, each of which receives a subset of the information transmitted by the IDS. By combining their respective decisions, we show that combining multiple detectors can further improve the detectability of adversarial attacks compared to a single detector in the case of a parallel IDS.

The objective of Chapter 6 is to study the actual feasibility of adversarial attacks, especially evasion attacks, against network-based intrusion detection systems (NIDS), by demonstrating that it is quite possible to fool these ML-based IDSs using our proposed adversarial algorithm while assuming as many constraints as possible in a black-box setting. Furthermore, since it is essential to design defense mechanisms to protect these ML-based intrusion detection systems, a defensive scheme is presented.

Perspectives

As demonstrated in this thesis, evasion attacks are indeed a threat to machine learning-based intrusion detection systems, allowing malicious network traffic to evade detection by adding a small adversarial perturbation. However, other types of attacks are presented in the adversarial learning literature, namely exploratory attacks and poisoning attacks.

Examples of exploratory attacks are model extraction and model inversion at-

tacks. The model extraction attack focuses on parameter inference. The idea is to create a copy of a target model without any information about the model architecture or training data. Another important point is that an attacker could first create a duplicate model in order to generate adversarial examples with the aim of exploiting the transferability property to perform an evasion attack. On the other hand, the model inversion attack focuses on input inference. The idea of model inversion is to try to recover some confidential data from the original training dataset. This attack can raise privacy concerns if an attacker is able to recreate the original model through a model extraction technique, or by directly querying the original model, as it could reveal sensitive information about the training data (e.g., medical records).

Poisoning attacks target the training data and thus occur during the training phase of the machine learning model. They violate the integrity property by altering the training data set. Several types of poisoning attacks can be distinguished, namely data injection, logical corruption and data modification. In the case of data injection attack, the adversary may add malicious data to the training data set without modifying the original training data. This could influence the output result of the model due to these additional inputs that shift the decision boundary of the target model. In the case of a data modification attack, the adversary may modify or delete some data or label in the original training dataset. An appropriate modification can significantly reduce the detection performance of the IDS for some attacks that the adversary intends to perform. For the logical corruption attack, the adversary has access to the algorithm used by the ML model. This allows the attacker to modify the logic of the algorithm (e.g., to create a backdoor in the intrusion detection system).

Lessons learned : feasibility of the evasion attacks

From this thesis and the review of existing literature on adversarial attacks on intrusion detection systems, several important lessons have been learned. First, it is clear that attacks based solely on feature-space manipulation may not provide a realistic approach, as the features cannot be easily transcribed back into network traffic once extracted and modified. Therefore, it is necessary to ensure that the generated

adversarial network traffic is valid by performing a problem-space projection after adding adversarial perturbations to the feature space.

Second, the attacker's knowledge of the mapping of raw network traffic into features, as well as the semantic and syntactic links that exist between these features, should be limited. This is important for creating a truly realistic scenario for the attacker, and assumptions of full knowledge of the IDS should be avoided.

Finally, it has been observed that black box attacks, such as those using Boundary, NES, OPT, or ZOO, can be easily detected by simple defenses such as Query Detection. Moreover, querying the IDS repeatedly is not feasible as IDSs are not designed to provide feedback when queried, and the attacker could easily reveal themselves.

Taken together, these lessons highlight the need to carefully consider the realistic aspects of intrusion detection systems when developing and testing defenses against adversarial attacks. Future studies should focus on addressing these limitations to improve the robustness of intrusion detection systems against adversarial attacks.

Appendix

Network traffic datasets

NSL-KDD

NSL-KDD dataset, which was introduced in 2009 [87], is a widely used dataset for assessing the effectiveness of intrusion detection systems. It is an improved version of the KDD CUP'99 dataset that faced two significant challenges: the presence of numerous redundant records and the tendency of classifiers to favor frequent records. NSL-KDD overcame these issues by eliminating redundant records and rebalancing the dataset classes. Consequently, it facilitates the evaluation of various machine learning algorithms in a comparative manner.

This dataset covers several attacks organized into four classes according to their nature: denial of service (DoS) attacks, probe attacks (Probe), root-to-local (R2L) attacks, and user-to-root (U2R) attacks. The records in the NSL-KDD dataset have 41 features in addition to a class label. These features are grouped into three categories: basic features, content features, and traffic features.

CIC-IDS2017

The network traffic data contained in the CIC-IDS2017 dataset was collected from a real network, which was split into two parts: the first part consisted of 4 machines launching attacks and the second part contained 10 machines being targeted. The dataset includes 50 gigabytes of raw data in PCAP format and a set of 84 features in CSV files. The network traffic was recorded for 5 days, resulting in a total of 2,830,743 instances. The traffic was classified into 15 classes, with one class representing normal traffic and the other 14 classes representing different types of attacks. The features were extracted using a tool called CICFlowMeter, which can generate up to 84 features related to bidirectional flows, including statistics on packet length, interarrival time, and flags used in TCP connections. The source code for CICFlowMeter is available, and the exhaustive list of features generated by the tool

is presented in a paper by Rosay et al. [133].

CTU-13

This dataset, provided by the Czech Technical University in Prague, contains different labeled traffic from 13 different application scenarios. Each of them follows the same process, including a different botnet attack variant (Neris, Rbot, Virut, Murlo, Menti, NSIS.ay and Sogou). The process involves monitoring a network for both benign network communications and malicious traffic executed by the botnet attack. All of this traffic is extracted as PCAP files and then formatted as flows via the Argus tool, which turns raw network data into flows.

CSE-CIC-IDS2018

This dataset contains a set of computer attack scenarios, such as Brute-force, Heartbleed, Botnet, DoS, DDoS, or Web attacks. There are two variants of botnet attacks: Zeus and Ares. The network traffic for each scenario was extracted as a PCAP file and formatted by CICFlowMeter to provide a set of 83 network features for thousands of labeled network flows. This dataset, being relatively recent, provides consistent data following the dataset creation methodology present in [64], allowing to have a reliable and realistic network traffic representation.

Source code

1^{st} conference paper

Efficient Intrusion Detection Using Evidence Theory

`https://github.com/PhD-Thesis2022/1st-conf-paper`

2^{nd} conference paper

Adversarial Training for Deep Learning-based Intrusion Detection Systems

`https://github.com/PhD-Thesis2022/2nd-conf-paper`

3^{rd} conference paper

Detect & reject for transferability of black-box adversarial attacks against network intrusion detection systems

`https://github.com/PhD-Thesis2022/3rd-conf-paper`

1^{st} journal paper

TAD: Transfer learning-based multi-adversarial detection of evasion attacks against network intrusion detection systems

`https://github.com/PhD-Thesis2022/1st-journal-paper`

2^{nd} journal paper

Adv-Bot: Realistic Adversarial Botnet Attacks against Network Intrusion Detection Systems

`https://github.com/PhD-Thesis2022/2nd-journal-paper`

Bibliography

[1] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(12):2935–2947, 2018.

[2] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[3] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[4] Andrea Venturi, Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. Drelab-deep reinforcement learning adversarial botnet: A benchmark dataset for adversarial attacks against botnet intrusion detection systems. *Data in Brief*, 34:106631, 2021.

[5] Patrick Howell O'Neillarchive. 2021 has broken the record for zero-day hacking attacks. *MIT Technology Review*, 2021.

[6] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. On the effectiveness of machine and deep learning for cyber security. In Tomás Minárik, Raik Jakschis, and Lauri Lindström, editors, *10th International Conference on Cyber Conflict, CyCon 2018, Tallinn, Estonia, May 29 - June 1, 2018*, pages 371–390. IEEE, 2018.

[7] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yan-Miao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

[8] Samaneh Mahdavifar and Ali A. Ghorbani. Application of deep learning to cybersecurity: A survey. *Neurocomputing*, 347:149–176, 2019.

[9] R Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

[10] David J Miller, Zhen Xiang, and George Kesidis. Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks. *Proceedings of the IEEE*, 108(3):402–433, 2020.

[11] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[12] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. Modeling realistic adversarial attacks against network intrusion detection systems. *Digital Threats: Research and Practice (DTRAP)*, 3(3):1–19, 2022.

[13] Islam Debicha, Richard Bauwens, Thibault Debatty, Jean-Michel Dricot, Tayeb Kenaza, and Wim Mees. Tad: Transfer learning-based multi-adversarial detection of evasion attacks against network intrusion detection systems. *Future Generation Computer Systems*, 138:185–197, 2023.

[14] Chaoyun Zhang, Xavier Costa-Pérez, and Paul Patras. Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms. *IEEE/ACM Transactions on Networking*, 2022.

[15] Zheng Wang. Deep learning-based intrusion detection with adversaries. *IEEE Access*, 6:38367–38384, 2018.

[16] Kaichen Yang, Jianqing Liu, Chi Zhang, and Yuguang Fang. Adversarial examples against the deep learning based network intrusion detection systems. In *MILCOM 2018-2018 ieee military communications conference (MILCOM)*, pages 559–564. IEEE, 2018.

[17] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. Analyzing the footprint of classifiers in adversarial denial of service contexts. In *EPIA Conference on Artificial Intelligence*, pages 256–267. Springer, 2019.

[18] Dongqi Han, Zhiliang Wang, Ying Zhong, Wenqi Chen, Jiahai Yang, Shuqiang Lu, Xingang Shi, and Xia Yin. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal on Selected Areas in Communications*, 39(8):2632–2647, 2021.

[19] Amir Mahdi Sadeghzadeh, Saeed Shiravi, and Rasool Jalili. Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE Transactions on Network and Service Management*, 18(2):1962–1976, 2021.

[20] Jiming Chen, Xiangshan Gao, Ruilong Deng, Yang He, Chongrong Fang, and Peng Cheng. Generating adversarial examples against machine learning based intrusion detector in industrial control systems. *IEEE Transactions on Dependable and Secure Computing*, 2020.

[21] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[22] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[23] Mohamed Amine Merzouk, Frédéric Cuppens, Nora Boulahia-Cuppens, and Reda Yaich. Investigating the practicality of adversarial evasion attacks on network intrusion detection. *Annals of Telecommunications*, pages 1–13, 2022.

[24] Mohammad J Hashemi, Greg Cusack, and Eric Keller. Towards evaluation of nidss in adversarial setting. In *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, pages 14–21, 2019.

[25] Martin Teuffenbach, Ewa Piatkowska, and Paul Smith. Subverting network intrusion detection: Crafting adversarial examples accounting for domain-

specific constraints. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 301–320. Springer, 2020.

[26] Rey Reza Wiyatno, Anqi Xu, Ousmane Dia, and Archy de Berker. Adversarial examples in modern machine learning: A review. *arXiv preprint arXiv:1911.05268*, 2019.

[27] João Vitorino, Nuno Oliveira, and Isabel Praça. Adaptative perturbation patterns: Realistic adversarial learning for robust intrusion detection. *Future Internet*, 14(4):108, 2022.

[28] Andrew McCarthy, Essam Ghadafi, Panagiotis Andriotis, and Phil Legg. Functionality-preserving adversarial machine learning for robust classification in cybersecurity and intrusion detection domains: A survey. *Journal of Cybersecurity and Privacy*, 2(1):154–190, 2022.

[29] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. Adversarial machine learning applied to intrusion and malware scenarios: a systematic review. *IEEE Access*, 8:35403–35419, 2020.

[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[31] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[32] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016.

[33] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 582–597. IEEE Computer Society, 2016.

[34] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[35] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[36] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.

[37] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[38] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018.

[39] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[40] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[41] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[42] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[43] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[44] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[45] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[46] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[47] Nicholas Carlini and David Wagner. Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*, 2016.

[48] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[49] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th USENIX workshop on offensive technologies (WOOT 17)*, 2017.

[50] Giovanni Apruzzese, Michele Colajanni, and Mirco Marchetti. Evaluating the effectiveness of adversarial attacks against botnet detectors. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2019.

[51] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[52] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 3–14, 2017.

[53] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples. *Advances in Neural Information Processing Systems*, 31, 2018.

[54] Chaoyun Zhang, Xavier Costa-Pérez, and Paul Patras. Tiki-taka: Attacking and defending deep learning-based intrusion detection systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 27–39, 2020.

[55] Maria Rigaki and Ahmed Elragal. Adversarial deep learning against intrusion detection classifiers. In *017 NATO IST-152 Workshop on Intelligent Autonomous Agents for Cyber Defence and Resilience, IST-152 2017; Czech Technical UniversityPrague; Czech Republic; 18-20 October 2017*, volume 2057, pages 35–48. CEUR-WS, 2017.

[56] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 79–91. Springer, 2022.

[57] Arkadiusz Warzyński and Grzegorz Kołaczek. Intrusion detection systems vulnerability on adversarial examples. In *2018 Innovations in Intelligent Systems and Applications (INISTA)*, pages 1–4. IEEE, 2018.

[58] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[59] Joseph Clements, Yuzhe Yang, Ankur A Sharma, Hongxin Hu, and Yingjie Lao. Rallying adversarial techniques against deep learning for network security. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08. IEEE, 2021.

[60] Olakunle Ibitoye, Omair Shafiq, and Ashraf Matrawy. Analyzing adversarial attacks against deep learning for intrusion detection in iot networks. In *2019 IEEE global communications conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[61] Ryan Sheatsley, Nicolas Papernot, Michael J Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial examples for network intrusion detection systems. *Journal of Computer Security*, (Preprint):1–26, 2022.

[62] James Aiken and Sandra Scott-Hayward. Investigating adversarial attacks against network intrusion detection systems in sdns. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE, 2019.

[63] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1332–1349. IEEE, 2020.

[64] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)*, pages 1–6. IEEE, 2016.

[65] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12. IEEE, 2021.

[66] Jason Andress. *The basics of information security: understanding the fundamentals of InfoSec in theory and practice*. Syngress, 2014.

[67] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[68] Richard W Jones, Andrew Lowe, and Michael J Harrison. A framework for intelligent medical diagnosis using the theory of evidence. *Knowledge-Based Systems*, 15(1):77–84, 2002.

[69] Mohamed El Yazid Boudaren, Lin An, and Wojciech Pieczynski. Dempster–Shafer fusion of evidential pairwise Markov fields. *International Journal of Approximate Reasoning*, 74:13–29, 2016.

[70] Huawei Guo, Wenkang Shi, and Yong Deng. Evaluating sensor reliability in classification problems based on evidence theory. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(5):970–981, 2006.

[71] F Salzenstein and A-O Boudraa. Unsupervised multisensor data fusion approach. In *Signal Processing and its Applications, Sixth International, Symposium on. 2001*, volume 1, pages 152–155. IEEE, 2001.

[72] Azzedine Bendjebbour, Yves Delignon, Laurent Fouque, Vincent Samson, and Wojciech Pieczynski. Multisensor image segmentation using Dempster—Shafer fusion in Markov fields context. *Geoscience and Remote Sensing, IEEE Transactions on*, 39(8):1789–1798, 2001.

[73] Peida Xu, Yong Deng, Xiaoyan Su, and Sankaran Mahadevan. A new method to determine basic probability assignment from training data. *Knowledge-Based Systems*, 46:69–80, 2013.

[74] Matt P Wand and M Chris Jones. *Kernel smoothing*. Crc Press, 1994.

[75] Philippe Smets and Robert Kennes. The transferable belief model. *Artificial intelligence*, 66(2):191–234, 1994.

[76] Didler Dubois and Henri Prade. Representation and combination of uncertainty with belief functions and possibility measures. *Computational intelligence*, 4(3):244–264, 1988.

[77] Philippe Smets. The combination of evidence in the transferable belief model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):447–458, May 1990.

[78] Faouzi Sebbak, Farid Benhammadi, Sofiane Bouznad, Abdelghani Chibani, and Yacine Amirat. An evidential fusion rule for ambient intelligence for activity recognition. In *International Conference on Belief Functions*, pages 356–364. Springer, 2014.

[79] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[80] Murray Rosenblatt et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[81] Vassiliy A Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability & Its Applications*, 14(1):153–158, 1969.

[82] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[83] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83(403):596–610, 1988.

[84] Zied Elouedi, Khaled Mellouli, and Philippe Smets. Assessing sensor reliability for multisensor data fusion within the transferable belief model. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):782–787, 2004.

[85] Sandeep Gurung, Mirnal Kanti Ghose, and Aroj Subedi. Deep learning approach on network intrusion detection system using nsl-kdd dataset. *International Journal of Computer Network and Information Security (IJCNIS)*, 11(3):8–14, 2019.

[86] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, 5:21954–21961, 2017.

[87] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009*, pages 1–6. IEEE, 2009.

[88] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[89] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[90] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207. Citeseer, 1996.

[91] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[92] David Aldous. The continuum random tree. i. *The Annals of Probability*, pages 1–28, 1991.

[93] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990.

[94] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[95] Ali Hamache, Mohamed El Yazid Boudaren, Houdaifa Boukersoul, Islam Debicha, Hamza Sadouk, Rezki Zibani, Ahmed Habbouchi, and Omar Merouani. Uncertainty-aware parzen-rosenblatt classifier for multiattribute data. In *International Conference on Belief Functions*, pages 103–111. Springer, 2018.

[96] Jasmin Kevric, Samed Jukic, and Abdulhamit Subasi. An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*, 28(1):1051–1058, 2017.

[97] Islam Debicha, Thibault Debatty, Wim Mees, and Jean-Michel Dricot. Efficient intrusion detection using evidence theory. In *INTERNET 2020 : The Twelfth International Conference on Evolving Internet*, pages 28–32, 2020.

[98] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, et al. Adversarial robustness toolbox v1. 0.0. *arXiv preprint arXiv:1807.01069*, 2018.

[99] Marek Pawlicki, Michał Choraś, and Rafał Kozik. Defending network intrusion detection systems against adversarial evasion attacks. *Future Generation Computer Systems*, 110:148–154, 2020.

[100] Islam Debicha, Thibault Debatty, Jean-Michel Dricot, and Wim Mees. Adversarial training for deep learning-based intrusion detection systems. In *ICONS 2021 : The Sixteenth International Conference on Systems*, 2021.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[102] Lior Rokach. Ensemble-based classifiers. *Artificial intelligence review*, 33(1):1–39, 2010.

[103] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[104] Igino Corona, Giorgio Giacinto, and Fabio Roli. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Inf. Sci.*, 239:201–225, 2013.

[105] Jan Lansky, Saqib Ali, Mokhtar Mohammadi, Mohammed Kamal Majeed, Sarkhel H. Taher Karim, Shima Rashidi, Mehdi Hosseinzadeh, and Amir Ma-

soud Rahmani. Deep learning-based intrusion detection systems: A systematic review. *IEEE Access*, 9:101574–101599, 2021.

[106] Mohamed A. Mohandes, Mohamed A. Deriche, and Salihu O. Aliyu. Classifiers combination techniques: A comprehensive review. *IEEE Access*, 6:19626–19639, 2018.

[107] A. P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232, 1968.

[108] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116. SciTePress, 2018.

[109] Rana Abou Khamis and Ashraf Matrawy. Evaluation of adversarial training on different types of neural networks in deep learning-based idss. In *2020 International Symposium on Networks, Computers and Communications, ISNCC 2020, Montreal, QC, Canada, October 20-22, 2020*, pages 1–6. IEEE, 2020.

[110] Jiliang Zhang and Chen Li. Adversarial examples: Opportunities and challenges. *IEEE transactions on neural networks and learning systems*, 31(7):2578–2593, 2019.

[111] Shilin Qiu, Qihe Liu, Shijie Zhou, and Chunjiang Wu. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5):909, 2019.

[112] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57. IEEE Computer Society, 2017.

[113] Marek Pawlicki, Michal Choras, and Rafal Kozik. Defending network intrusion detection systems against adversarial evasion attacks. *Future Gener. Comput. Syst.*, 110:148–154, 2020.

[114] Giovanni Apruzzese, Mauro Andreolini, Mirco Marchetti, Andrea Venturi, and Michele Colajanni. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management*, 17(4):1975–1987, 2020.

[115] Jinlan Zhang, Qiao Yan, and Mingde Wang. Evasion attacks based on wasserstein generative adversarial network. In *2019 Computing, Communications and IoT Applications (ComComAp)*, pages 454–459. IEEE, 2019.

[116] Yizhi Ren, Qi Zhou, Zhen Wang, Ting Wu, Guohua Wu, and Kim-Kwang Raymond Choo. Query-efficient label-only attacks against black-box machine learning models. *Computers & security*, 90:101698, 2020.

[117] Franziska Boenisch, Reinhard Munz, Marcel Tiepelt, Simon Hanisch, Christiane Kuhn, and Paul Francis. Side-channel attacks on query-based data anonymization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1254–1265, 2021.

[118] Islam Debicha, Thibault Debatty, Jean-Michel Dricot, Wim Mees, and Tayeb Kenaza. Detect & reject for transferability of black-box adversarial attacks against network intrusion detection systems. In *International Conference on Advances in Cyber Security*, pages 329–339. Springer, 2021.

[119] Yahya Al-Hadhrami and Farookh Khadeer Hussain. Real time dataset generation framework for intrusion detection systems in iot. *Future Generation Computer Systems*, 108:414–423, 2020.

[120] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, 27(1):357–370, 2022.

[121] A Pektaş and T Acarman. Effective feature selection for botnet detection based on network flow analysis. In *International Conference Automatics and Informatics*, pages 1–4, 2017.

[122] Wu Zhijun, Li Wenjing, Liu Liang, and Yue Meng. Low-rate dos attacks, detection, defense, and challenges: A survey. *IEEE access*, 8:43920–43943, 2020.

[123] G Ziemba, Darren Reed, and Paul Traina. Rfc1858: Security considerations for ip fragment filtering, 1995.

[124] Jean-loup Gailly. zlib: A massively spiffy yet delicately unobtrusive compression library. *http://www. zlib. net/*, 2012.

[125] Syed Rameem Zahra and Mohammad Ahsan Chishti. Packet header compression in the internet of things. *Procedia Computer Science*, 173:64–69, 2020.

[126] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *computers & security*, 45:100–123, 2014.

[127] Meng Wang, Yiqin Lu, and Jiancheng Qin. A dynamic mlp-based ddos attack detection method using feature selection and feedback. *Computers & Security*, 88:101645, 2020.

[128] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, and Wei Chen. Tr-ids: Anomaly-based intrusion detection through text-convolutional neural network and random forest. *Security and Communication Networks*, 2018, 2018.

[129] Raniyah Wazirali. An improved intrusion detection system based on knn hyperparameter tuning and cross-validation. *Arabian Journal for Science and Engineering*, 45(12):10859–10873, 2020.

[130] Francisco Sales de Lima Filho, Frederico AF Silveira, Agostinho de Medeiros Brito Junior, Genoveva Vargas-Solar, and Luiz F Silveira. Smart detection: an online approach for dos/ddos attack detection using machine learning. *Security and Communication Networks*, 2019, 2019.

[131] V Kanimozhi and T Prem Jacob. Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset cse-cic-ids2018 using cloud computing. In *2019 international conference on communication and signal processing (ICCSP)*, pages 0033–0036. IEEE, 2019.

[132] Beny Nugraha, Anshitha Nambiar, and Thomas Bauschert. Performance evaluation of botnet detection using deep learning techniques. In *2020 11th International Conference on Network of the Future (NoF)*, pages 141–149. IEEE, 2020.

[133] Arnaud Rosay, Kévin Riou, Florent Carlier, and Pascal Leroux. Multi-layer perceptron for network intrusion detection: From a study on two recent data sets to deployment on automotive processor. *Annals of Telecommunications*, 77(5-6):371–394, 2022.