A Case Study with CICIDS2017 on the Robustness of Machine Learning against Adversarial Attacks in Intrusion Detection

Marta Catillo
Università degli Studi del Sannio
Benevento, Italy
marta.catillo@unisannio.it

Andrea Del Vecchio
Università degli Studi del Sannio
Benevento, Italy
a.delvecchio2@studenti.unisannio.it

Antonio Pecchia
Università degli Studi del Sannio
Benevento, Italy
antonio.pecchia@unisannio.it

Umberto Villano
Università degli Studi del Sannio
Benevento, Italy
villano@unisannio.it

ABSTRACT

Intrusion detection systems (IDS) play a key role to assure security properties of modern computer networks. IDS are often based on machine and deep learning techniques; as such, IDS are vulnerable to various forms of adversarial attacks. This paper presents an initial case study on the robustness of machine learning for network intrusion detection against adversarial attacks. Experiments are based on a recent fix of the widely-used CICIDS2017 benchmark dataset, two well-known machine learning techniques for intrusion detection (i.e., deep autoencoders and decision trees), and the virtual adversarial method (VAM) to generate the adversarial examples. Based on the data and experiments at hand, the results provide many interesting findings on the robustness of the IDS models assessed. The autoencoder-based IDS is more robust to evasion rather than overstimulation. On the contrary, the decision tree is vulnerable to evasion; moreover, changes to the learning parameters can strongly affect the robustness of the decision tree against the VAM attack.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; **Intrusion detection systems**;

KEYWORDS

intrusion detection, machine learning, adversarial examples, autoencoder, Denial of Service

ACM Reference Format:
Marta Catillo, Andrea Del Vecchio, Antonio Pecchia, and Umberto Villano. 2023. A Case Study with CICIDS2017 on the Robustness of Machine Learning against Adversarial Attacks in Intrusion Detection. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3600160.3605031

ARES 2023, August 29–September 01, 2023, Benevento, Italy

ACM ISBN 979-8-4007-0772-8/23/08…$15.00
https://doi.org/10.1145/3600160.3605031

1 INTRODUCTION

The ever-increasing use of the Internet technologies, the diffusion of IoT and mobile computing devices and the use of relatively "insecure" software have given an exponential boost to cyber attacks. Terms such as Denial of Service (DoS), phishing, cyber frauds, identity theft, data exfiltration and ransomware have broken into our everyday lives. Unfortunately, these phenomena are not just annoyances, but they can lead to severe disruptions, negation of privacy and huge monetary losses. As a matter of fact, protecting citizens and businesses from cyber attacks has become an indispensable goal for governments. The final solution to this problem requires a diffused culture of security that currently is lacking (and it will be lacking for many years in the future). People should be aware of the risks and learn to protect credentials, passwords and devices; web sites and data centers should be suitably configured and adopt only secure software, just to mention a couple of issues. In the meantime, a viable solution is to protect current intrinsically-insecure networks and systems from external intrusions.

In this context, the use of intrusion detection systems (IDS) plays a key role. Commercial IDS solutions exist and are widely used; however, they often are just palliatives, due to the high volume of attacks and the continuous exploitation of just-discovered vulnerabilities in hardware and software (0-day exploits). At the state of the art, the use of machine learning (ML) and deep learning (DL) for intrusion detection is an active research field [12]. Many ML(DL)-based IDS proposals in the literature can attain very high recognition rates of attacks, at least on the reference datasets commonly used as benchmarks [32]; unfortunately, they seem to be not equally effective on real-world network traffic [1, 5, 39]. Moreover, ML(DL)-based IDS can be rather "fooled out" by *adversarial examples*, which are obtained by applying imperceivable perturbations to the legitimate input [10, 15, 37].

This paper presents an initial case study on the robustness of machine learning against adversarial attacks in network intrusion detection. For instance, in the context of IDS adversarial attacks aim to cause *evasion*, i.e., intrusion patterns going undetected, or *overstimulation*, i.e., patterns that generate many false alerts. The case study is based on two well-known machine learning techniques for intrusion detection, i.e., deep autoencoders and decision trees, and the virtual adversarial method (VAM), which is used to generate

the adversarial examples. We use normal and DoS *flow records*[1] from a recent fix of the widely-used CICIDS2017 public intrusion detection dataset. The flow records are used to obtain different IDS models, at first; VAM is used to perturb the values associated with the flow records in order to implement a feature-level attack against the IDS models.

The results provide many interesting findings on the robustness of the IDS models, which are comprehensively assessed by the typical metrics of of *recall* (R), *precision* (P), *F-score*, *false positive rate* (FPR) and *accuracy* (A). We found out that the autoencoder-based IDS is more robust to evasion rather than overstimulation. On the contrary, the decision tree is much more vulnerable to evasion; moreover, changes to the learning parameters can strongly affect the robustness of the decision tree against the VAM attack. While this study should be contextualized with respect to the data and experiments at hand, we made sure to base our findings on both a semi-supervised (autoencoder) and a supervised (decision tree) learning technique, independent data splits to separate the generation of the IDS models from the adversarial examples, and several parameterization of both the IDS models and the VAM attack in order to mitigate the threats to validity of our study.

The rest of the paper is organized as follows. Sect. 2 presents related work in the area. Sect. 3 summarizes the dataset and techniques adopted. Sect. 4 describes the generation of the IDS models. Sect. 5 discusses the adversarial examples and the results on the robustness of the models assessed. Sect. 6 concludes the work.

2 RELATED WORK

Machine learning for intrusion detection. The recent trends in security research have shown that machine learning and deep learning are highly relevant to network traffic classification. These techniques are often proposed as part of IDS frameworks and applications using different classification models, such as support vector machines (SVM), decision trees (DT), k-nearest neighbors (KNN), and artificial neural networks (ANN) [38]. In general, all these techniques can handle large and multi-dimensional data by automatically reducing the complexity of network traffic. An ever-growing community of researchers and practitioners leverages public intrusion datasets, such as UNSW-NB15 [23], NDSec-1 2016 [2], and CICIDS2017 [34], to design, evaluate and compare novel IDS. In these collections of data, each record typically pertains to a network flow and the label states whether it is an attack or not. It is a fact that machine and deep learning techniques populate much of the intrusion detection literature [42].

The state-of-the-art on network intrusion detection systems shows the potential of **autoencoders** for the development of high-performance models. In general, there are two ways to leverage the autoencoders for intrusion detection. One possibility is to use them for dimensionality reduction and then use well-known classifiers in order to pinpoint the attacks, such as [19] and [40]. It is worth noting that autoencoders were first developed as *nonlinear* extension of the standard linear principal component analysis (PCA) in order to perform dimensionality reduction [18]. The second way of using an autoencoder, instead, is to train it so as to encode and reconstruct the normal traffic. In this context, any deviation from the "normal" traffic behavior allows us to recognize attack points [27]: this is the solution adopted by the autoencoders considered in this work. There are many studies in the literature that use the autoencoders for anomaly detection. For example, in [7] is presented a cross-device method, which allows learning a single IDS model (in lieu of many separate models) atop the traffic of different Internet of Things (IoT) devices. In [20] the authors propose Kitsune, an autoencoder-based technique for online attack detection. The core algorithm of the technique is KitNet, which uses a collection of autoencoders to distinguish between normal and attack traffic. Experimental results show the potential of Kitsune to identify different classes of attacks. In [41], the authors show an effective deep learning method, namely autoencoder-IDS (AE-IDS), based on the random forest algorithm. The main innovation of the approach lies in the combination of 3-layer shallow autoencoders and traditional unsupervised machine learning clustering algorithm. The approach is evaluated by means of the CSE-CIC-IDS2018 dataset. In [25] is proposed a framework for detecting and explaining anomalies in network traffic. The authors leverage a variational autoencoder in order to detect anomalies. The validity of the proposal is demonstrated on the University of Granada (UGR) dataset. A complete analysis of autoencoders for network intrusion detection is reported in [36].

Adversarial attacks in intrusion detection. Machine learning, although useful to tackle the growing number and increasing sophistication of attacks, is highly susceptible to adversarial examples [37]. *Adversarial attacks* are specific kinds of attacks, which aim to induce machine and deep learning models to produce wrong classifications. This definition of adversarial attack was first introduced and studied in the image classification domain by Goodfellow et al. [37]. In fact, most of the adversarial machine learning research so far has focused on image and object recognition domains. There are many adversarial attack methods in the literature, such as the Fast Gradient Sign Method (FGSM) [14], the Jacobian-based Saliency Map Attack (JSMA) [30], Deepfool [22], and the Carlini Wagner (CW) [4]. All of these algorithms heavily rely on the knowledge of the model under attack. Therefore, it would be reasonable to consider adversarial attacks as infeasible in intrusion detection scenarios, since most of the time attackers have little or no knowledge of the machine learning algorithms adopted for the intrusion detection task. However, the authors in [28, 29] highlight the concept of *transferability*, in that the adversarial examples, generated for a given model, may retain their ability to cause misclassification when submitted to a different model.

Recent work on adversarial attacks from a security perspective can be found in [11], [33] and [17]. Countermeasures to poisoning attacks have also been proposed. These are based on data sanitization (i.e., a form of outlier detection) [24] and multiple classifier systems [3]. The authors in [35] show the perspective of real enterprises about adversarial attacks. They state that modern organizations are aware of these problems, but do not consider adversarial attacks as a top-priority because there are no defensive mechanisms that are truly effective in real environments. The interested reader is referred to the survey by He et al. [15] for a complete taxonomy of adversarial machine learning for network intrusion detection systems.

[1] A flow record – often informally called network flow – holds the values of categorical and numeric features that provide context data and summary statistics computed from the packets pertaining to a network flow between a source computer and a destination across a network.

Table 1: Total records and number of normal/attack records by data split used in our study.

data split	total records	(records by type) normal	attack
IDS_TRAINING	190,516	123,825	66,691
IDS_VALIDATION	36,696	23,755	12,941
IDS_TEST	36,870	24,045	12,825
ADV_TRAINING	190,729	123,994	66,735
ADV_BASELINE	9,170	2,925	6,245
total	**463,981**	**298,544**	**165,437**

3 DATASET AND TECHNIQUES

In this section we describe the reference dataset, the techniques used to obtain the IDS models and the adversarial attack.

3.1 Dataset: CICIDS2017 (2021 update)

The dataset considered for the following experiments is CICIDS2017, produced by the Canadian Institute for Cybersecurity [34]. The dataset is provided both as a set of `pcap` files and bidirectional labeled flow records (csv files). In the latter format, the records were obtained by `CICFlowMeter`[2] and consist of categorical and numeric features computed from the packets exchanged between a source computer and a destination across a network. More recently, the authors of [13] provided an **update** of `CICFlowMeter` aiming to fix some major issues pertaining to the termination of TCP flows – leading to fragments of incorrectly truncated flows in the original dataset – and an updated release of the CICIDS2017 dataset[3], release which is the one used in the following experiments.

`CICFlowMeter` generates records made of 83 features, such as source-destination IP address and port, duration, number and length of packets, flag counts, min, max, mean, and standard deviation of the packet inter-arrival times. Thus, each record has a fixed-length of 83 values plus a class label, which states if it is an attack or not. CICIDS2017 consists of normal and attack flow records. Records were collected through a five-day capture campaign from July 3, 2017 to July 7, 2017 and made available in day-by-day csv files (ranging from `monday.csv` to `friday.csv`).

Our study is based on 463,981 **normal** and **DoS attack** flow records belonging to the `wednesday.csv` file of the CICIDS2017 collection. DoS attacks are a relevant case study due to their ever-increasing spread and volume, also in the form of Economic Denial of Sustainability (EDoS), i.e., a new security and economic threat to cloud computing [8]. For the purposes of this study the flow records are arranged into five disjoint data splits – serving different steps of the experiments presented below – listed in Table 1 along with their size. Splits are obtained through a stratified random sampling approach with no replacement from the `wednesday.csv` file. DoS attacks addressed by the study are GoldenEye, Hulk, Slowhttptest and Slowloris. It is worth noting that we remove non-relevant and biasing features (i.e., id and timestamp of the records, source IP address and port, destination IP address and port) before the experiments; removal leads to the use of 77 out of 83 features (label excluded).

[2]https://github.com/ahlashkari/CICFlowMeter
[3]https://downloads.distrinet-research.be/WTMC2021/tools_datasets.html

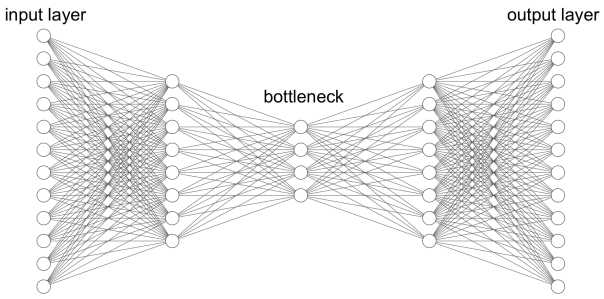


Figure 1: Representation of an autoencoder.

3.2 IDS Techniques

3.2.1 ***Autoencoder.*** The first intrusion detection method considered in this study is based on use of an **autoencoder (AE)**. Autoencoders are a class of feedforward neural networks designed to reconstruct the input data point at the output layer. The architecture of an AE consists of *input*, *output* and one (more) *hidden* layer(s). Typically, the input layer has the same length as the output layer. The middle (hidden) layer of an autoencoder is also known as the **bottleneck** layer and its dimension is lower than the input/output layer. The layers that come before the bottleneck make up the **encoder**, while the layers that come after the bottleneck make up the **decoder**. When multiple hidden layers are added to provide depth, the resulting neural network is known as **deep** or **stacked autoencoder**. Figure 1 shows the representation of an autoencoder with three hidden layers. In the following experiments, an autoencoder is fed with n-dimensional points $\mathbf{x}$= $[x_1,x_2,...,x_n]$, where $\mathbf{x}$ is a vector of n real numbers, such as a flow record representing network traffic for the dataset used in the experiments. The **encoder** transforms $\mathbf{x}$ into $\mathbf{y}$, i.e., a *lower* dimensionality representation of $\mathbf{x}$ at the bottleneck. On the other hand, the **decoder** transforms $\mathbf{y}$ into an n-dimensional point $\mathbf{z}$=$[z_1,z_2,...,z_n]$. Encoding–decoding formulas are given in Eq. (1). They represent the case of an autoencoder with only one hidden layer:

$$y = \sigma(Wx + b) \qquad z = \sigma'(W'y + b') \tag{1}$$

where W, W', b and b' are appropriate – obtained after training – weight matrices and bias vectors; σ and σ' are element-wise activation functions.

The quality of the reconstruction is measured by the **reconstruction error (RE)**, i.e., the difference between the reconstructed, i.e., $\mathbf{z}$, and the original version of the input, i.e., $\mathbf{x}$:

$$RE = \frac{1}{n}\sum_{i=1}^{n}(z_i - x_i)^2 \tag{2}$$

where z_i and x_i (with $1 \leq i \leq n$) denote the components of the output and input vectors, and n is the dimensionality. It is worth noting that $\mathbf{z}$ is also known as the **reconstruction** of $\mathbf{x}$.

IDS approach. In general, an autoencoder is trained with the data points contained in a *training set*. Each point $\mathbf{x}$ of the training set is fed to the autoencoder, and weight matrices and bias vectors are progressively adjusted in order to minimize the difference between $\mathbf{x}$ and its reconstruction $\mathbf{z}$. After training, the autoencoder

will reconstruct accurately, i.e., with low RE, future points "similar" to those used for training. Based on this principle, in order to pursue an IDS approach, we build up an autoencoder and we train it solely by means of normal data points, i.e., flow records related to traffic collected under "normal" network operations; after training, the autoencoder can identify any instance not conforming to the model as a potential intrusion. Therefore, a properly trained autoencoder is able to correctly reconstruct normal data points with *low* RE (good reconstructions), while it will generate *high* RE (bad reconstructions) for data points corresponding to intrusions.

The above approach assumes that the training data has labeled instances only for the normal class and falls within the larger scope of *semi-supervised* anomaly detection [9]. In order to discriminate normal and intrusion records we apply a cut-off **detection threshold** to the RE. In particular, we set the detection threshold in a semi-supervised manner. The threshold is computed through a small (i.e., 10%) disjoint subset of the training set, which we call the **threshold set**. An outlier detection algorithm is applied to the threshold set in order to discriminate *inliers* from *outliers*. Inliers and outliers are fed to the autoencoder by generating two separate vectors of reconstruction errors, i.e., inliers and outliers, respectively. The threshold is set to obtain an optimal balance between inliers and outliers, i.e., inliers whose RE is below the threshold against outliers characterized by a RE above the threshold. The interested reader is referred to [6] for the details of the threshold selection method.

3.2.2 ***Decision Tree.*** The second technique considered in this paper is the **decision tree (DT)**. The decision tree is a popular machine learning algorithm widely-used for both classification and regression problems. In the context of a classification problem, a decision tree is a tree-like structure where each node is a predicate tested on a feature, each link is a decision, and each leaf is an outcome. The goal is to learn simple decision rules from the data to create a model that predicts the value of a target variable. Given a dataset, a decision tree groups and labels data points that are similar between them, and searches for the best rules that split the data points that are dissimilar until the splits reach a given degree of similarity. In general, the main goal of a decision tree is to find the best splits between nodes that optimally classify data points into the correct categories. Therefore, the whole process can be represented in a tree-like structure, and the model generated can be summarized as a set of "if–then" rules. In its simplest form, a decision tree is identified by three types of nodes: *root node*, which is the beginning of the tree; *internal node*, which is a sub-node that might be further split into additional sub-nodes; *leaf node*, which is a sub-node that cannot be split into further additional sub-node and represents a possible outcome. The number of levels, not including the root node, defines the *depth* of a tree. The majority of decision trees deal with the classification problem, which is also the primary concern of this study.

IDS approach. The optimal training of a decision tree is an NP-hard problem. Therefore, training is generally done using heuristics that lead to a non-optimal, but close to optimal, decision tree [16]. During training, the decision tree starts at the root node and distributes the *training set* along the internal nodes. This process continues as a loop and is repeated at each internal node until all the leaves are in the proper order. The result is that the features at every leaf node are from the same class. The specific tree-like structure that is learned from the training set depends on user-supplied hyperparameters, which drive the creation of the predicates (i.e., nodes) meant to classify a given input data point – network flow record in this study – connections and depth of the tree. Typical hyperparameters are *max_depth*, which regulates the length of the path from the root to the furthest leaf of the tree; and *min_samples_leaf*, which indicates the minimum number of data points to be collected at a leaf during the training phase. Based on these core principles, in order to pursue an IDS, we set up and train a decision tree with a root node, where the input data points are passed through. Therefore, to classify a data point, one starts at the root of the decision tree and follows the branch indicated by the outcome of each test until a leaf node is reached. The name of the class at the leaf node is the resulting classification (i.e., normal or attack in our case study). It should be noted that the decision tree is a *supervised* classification technique. At training time it requires both normal and attack data points, and the availability of the labels in order to infer the decision predicates.

3.3 VAM

This study is based on the virtual adversarial method (VAM), i.e., a perturbation method based on virtual adversarial training [21]. Given the classifier to be attacked and an original example x, VAM produces the adversarial example $x_{adv} = x + eps \cdot d$, where d maximizes the Kullback-Leibler divergence $KL[F(x)||F(x+d)]$ as in [26]. It should be noted that F(x) are class probabilities, which are typically returned by the softmax function applied to the classifier logits $Z(x)$, i.e., $softmax(Z(x))$, with $Z : \mathcal{X} \rightarrow \mathbb{R}^K$ (where $\mathcal{X}$ is the space of the inputs and K the number of the classes). VAM does not require too much information (e.g., loss function or the decision boundaries of the classifier) to be applied; as such, it is particularly suitable in intrusion detection because attackers have scarce or no information regarding the target IDS .

The Kullback-Leibler divergence is usually applied on two statistical distributions in order to evaluate the amount of information lost when the first distribution is approximated with the second. It is worth noting that VAM was originally meant to support the training phases of a given model in order to avoid overfitting [21]. By introducing VAM examples in the training dataset, it is possible to make the model more flexible and resilient to potential data and concept drift. This assumption can be leveraged in adversarial contexts. In fact, an altered example crafted to cause an information loss – submitted to a ML model – may cause it to respond in an unexpected way, possibly causing a misclassification.

VAM is implemented here by means of the cleverhans (version 2.1.0) python library, which uses an iterative approach. Iterations are controlled by the **num_iterations** parameter. At each iteration, the perturbation is scaled by the **xi** parameter and then summed to the original example in order to obtain the logits of the altered example and the KL metric. The final operation consists of the computation of the gradient of the KL metric with respect to the perturbation. At the end of the iterations, the perturbation is scaled by the **eps** parameter before being added to the original example x in order to generate x_{adv}.

Table 2: Performance of the IDS models (measured with IDS_TEST).

	conf.	parameters	R	P	F-score	FPR	A
autoencoder	AE_1	bottleneck = 9 epochs = 120 batch size = 1,280	0.974	0.980	0.977	0.011	0.984
	AE_2	bottleneck = 8 epochs = 120 batch size = 1,024	0.959	0.984	0.971	0.008	0.980
	AE_3	bottleneck = 8 epochs = 90 batch size = 1,024	0.956	0.986	0.971	0.007	0.980
decision tree	DT_1	min_samples_leaf = 100	0.997	0.999	0.998	0.000	0.999
	DT_2	min_samples_leaf = 200	0.995	0.994	0.995	0.003	0.996
	DT_3	min_samples_leaf = 250	0.991	0.996	0.994	0.002	0.996

3.4 Evaluation Metrics

Given an IDS model obtained by applying the techniques above, the performance at detecting normal and attack records in a dataset is measured by the typical metrics of *recall* (R), *precision* (P), *F-score*, *false positive rate* (FPR) and *accuracy* (A). The metrics are computed as follows:

$$R = \frac{TP}{TP + FN} \qquad P = \frac{TP}{TP + FP} \qquad \textit{F-score} = 2 \cdot \frac{P \cdot R}{P + R}$$

$$FPR = \frac{FP}{FP + TN} \qquad A = \frac{TN + TP}{TN + FP + TP + FN}$$

where True Positive (TP) and True Negative (TN) represent the number of records that are correctly classified by the IDS, while False Positives (FP) and False Negatives (FN) indicate the misclassifications. For example, TP is the number of attack records that are deemed attacks by the IDS; on the other hand, TN are normal records that are deemed normal by the IDS.

4 IDS MODELS

Setting up and training either a machine or deep learning IDS is a complex matter and it requires establishing a large number of parameters. As for the autoencoder, it entails the number of hidden layers, neurons per layer, size of the bottleneck, activation functions and optimizer (just to mention a few); on the other hand, the decision tree is intertwined with two major parameters, i.e., the maximum depth of the tree (max_depth) and the minimum number of records to be at a leaf node (min_samples_leaf). We implemented the experiments in python by using the ubiquitous Keras (version 2.4.1), tensorflow (version 2.3.0rc0) and scikit-learn (version 0.23.1) for the autoencoder and the decision tree, respectively.

4.1 Parameterization and Training

As in any typical machine learning experiment, for both the autoencoder and the decision tree we set an initial mixture of parameters and then train the model with the IDS_TRAINING data split (listed in Table 1 as the splits below). After training, the "goodness" of the model obtained, parameters and overfitting issues are validated with IDS_VALIDATION: if the model is not satisfactory, the parameters are adjusted and training is done again until a suitable configuration is found. At the end of the procedure we found out a suitable autoencoder whose main parameters are: five layers (made of 77, 48, 8, 48, 77 neurons, respectively), the rectified linear unit (ReLU) activation function for all the layers except of the output layer (using the hyperbolic tangent), the root mean squared propagation optimizer, the mean squared error loss function (which matches the notion of reconstruction error of the autoencoder), 120 epochs and batch size equal to 1,024. As for the decision tree, we choose max_depth = 10 and min_samples_leaf = 200. It is worth noting that IDS_TEST plays no role at all during training and validation.

4.2 Evaluation

Table 2 shows the performance of different IDS models on IDS_TEST, i.e., the data split that provides normal and attack records "held out" from training and validation. The models with the above-mentioned parameters obtained after training are AE_2 (autoencoder) and DT_2 (decision tree). For the sake of completeness, we opt to explore also a few combinations of parameters around AE_2 and DT_2 to make sure our findings do not depend – by chance – on the outcome of a single IDS model. These lead to the models AE_1, AE_3, DT_1 and DT_3 in Table 2. Overall, the values of the metrics indicate that all the models are extremely satisfactory; however, the specific configuration of the parameters underlie a tradeoff in the metrics. For example, augmenting the bottleneck of the autoencoder by 1 neuron and increasing the batch size to 1,280 (AE_1) allows to improve the R from 0.959 to 0.974 compared to AE_2; however, the improvement of R makes the FPR worse in AE_1 with respect to AE_2. Similar considerations hold for the decision tree, which tends to overfit the data when min_sample_leaf is too low, as in DT_1 (FPR=0.000). Other negligible changes can be noted in Table 2. For example, P increases from 0.984 to 0.986 between AE_2 and AE_3; R decreases from 0.997 to 0.995 between DT_1 and DT_2.

Reasoning on the improvement or deterioration of the metrics as affected by the parameters is beyond the scope of this paper. These phenomena are partially known in the literature, such as the overfitting issues. Here we aim to observe that changes of the parameters – negligible for the legitimate examples – can strongly compromise the ability of an IDS to tolerate adversarial examples.

5 ROBUSTNESS ASSESSMENT

We generate the adversarial examples by a transfer-based approach. Transfer-based approaches rely on the use of a **surrogate model**: (i) at first, the adversarial attack – VAM in this study – is run against the surrogate model in order to generate the adversarial examples, and (ii) the adversarial examples obtained are fed to the actual model(s) to be attacked. Transfer-based attacks capitalize on the transferability of adversarial examples [28, 29] to the actual model(s) to be attacked. This step leverages ADV_TRAINING and ADV_BASELINE (listed in Table 1) to ensure that the adversarial examples are based on flow records independent of those used for the models in Sect. 4.

5.1 Adversarial Examples Generation

The surrogate model consists of a typical feedforward neural network of eight fully-connected layers, i.e., six made of 77 neurons and two made of 2 neurons. The surrogate model is trained with the ADV_TRAINING data split. The input layer (77 neurons) maps to the features adopted; the output layer (2 neurons) implements a softmax function that produces the class probabilities to be converted to a binary outcome (i.e., normal or attack). Other relevant parameters include the use of the ReLU in most of the hidden layers and other learning facets, such as epochs (5), batch size (128), optimizer (adam) and loss function (softmax cross entropy with logits). The ADV_BASELINE data split provides a set of legitimate points to be perturbed. For each point x in ADV_BASELINE, VAM computes $F(x)$ on the surrogate model and generates the adversarial example x_{adv} according to the procedure described in Sect. 3.3.

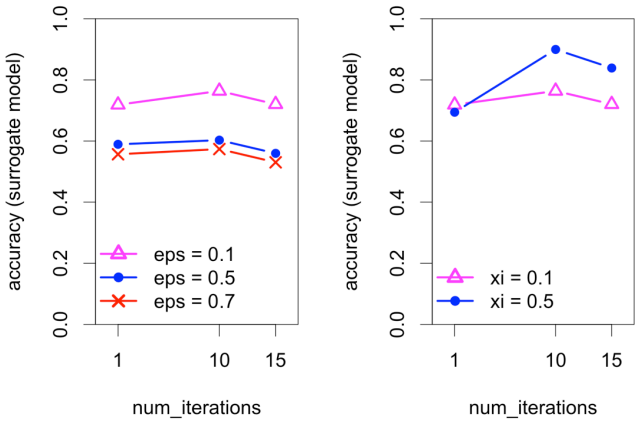
We generate 12 sets of adversarial examples by varying the parameters of VAM, i.e., eps, xi and num_iterations; the sets are named from VAM_1 to VAM_12. Table 3 shows the accuracy of the surrogate model with respect to each set and the corresponding values of the parameters. It should be noted that the parameterizations are sorted by increasing accuracy: the lower the accuracy of the surrogate, the higher the strength of the attack. Figure 2 aims to provide a visual understanding on the effect of the parameters. Figure 2a (accuracies obtained with xi=0.1) indicates that the largest drop of accuracy is obtained when eps varies from 0.1 to 0.5; the accuracy is less impacted by num_iterations, given a value of eps. On the other hand, according to Figure 2b (accuracies obtained with eps=0.1), an increasing value of xi causes the accuracy to increase when num_iterations is higher than 1. It is interesting to note that different combinations of the parameters might lead to similar accuracy on the surrogate model; moreover, the combinations in Table 3 are enough to elicit a significant spectrum of accuracies – strength of the adversarial attack – that range from 0.530 to 0.900.

5.2 Results

Each VAM_i set (with i=1, 2, ..., 12) is fed to the IDS models of Sect. 4 in order to gain insight into the robustness against adversarial examples. Just to mention a few examples, when tested with VAM_1 – accuracy on the surrogate model equals to 0.530 – AE_1 achieves

Table 3: Accuracy of the surrogate model with respect to different parameterizations of VAM.

conf.	eps	xi	num iterations	accuracy (surrogate)
VAM_1	0.7	0.1	15	0.530
VAM_2	0.7	0.1	1	0.557
VAM_3	0.5	0.1	15	0.560
VAM_4	0.7	0.1	10	0.574
VAM_5	0.5	0.1	1	0.589
VAM_6	0.5	0.1	10	0.603
VAM_7	0.1	0.5	1	0.694
VAM_8	0.1	0.1	1	0.719
VAM_9	0.1	0.1	15	0.720
VAM_10	0.1	0.1	10	0.764
VAM_11	0.1	0.5	15	0.839
VAM_12	0.1	0.5	10	0.900



(a) xi = 0.1 (eps and num_iterations vary). **(b) eps = 0.1 (xi and num_iterations vary).**

Figure 2: Accuracy of the surrogate model by value of xi and eps with respect to num_iterations.

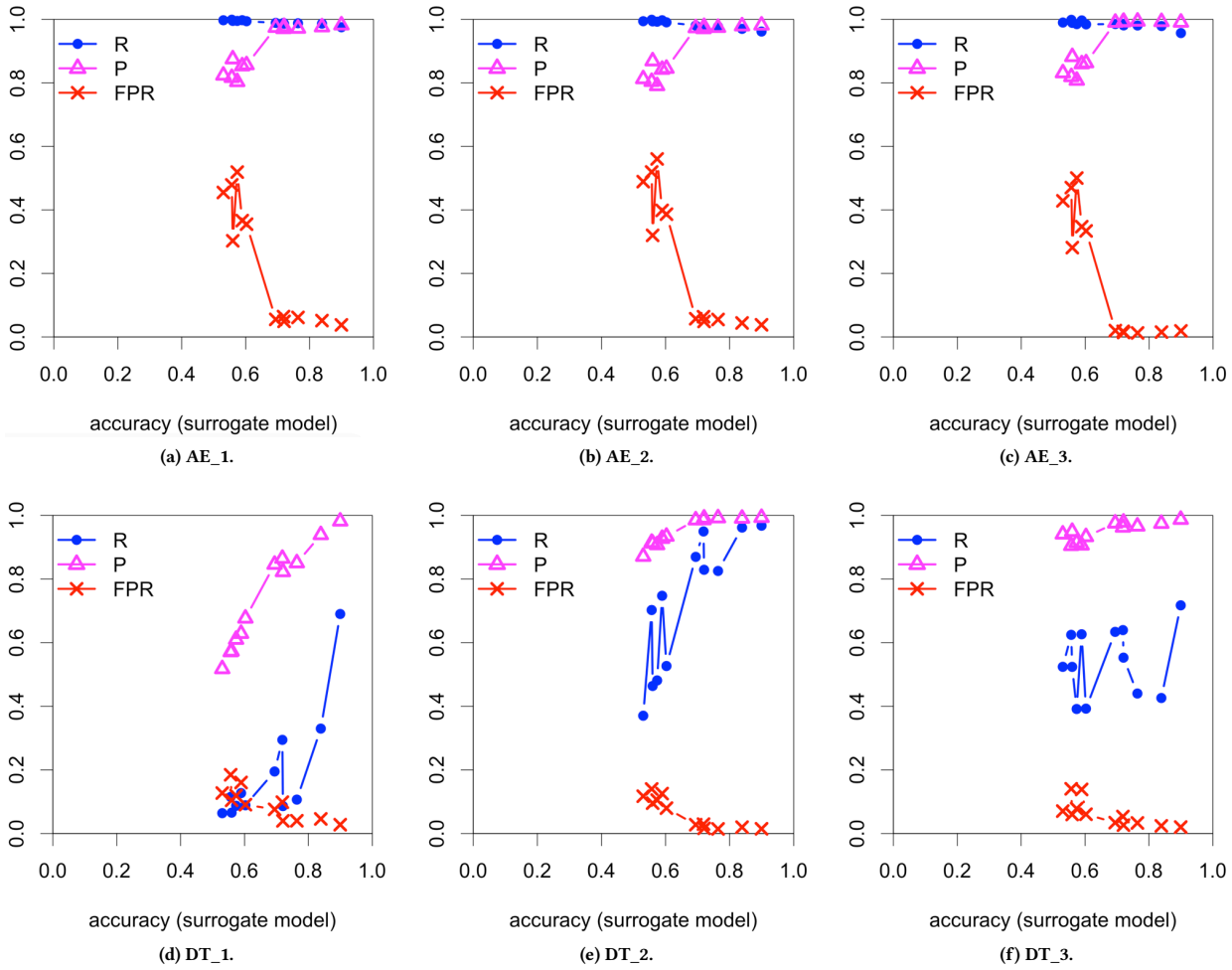
R=0.997, P=0.824, F-score=0.902, FPR=0.455 and A=0.853, while DT_1 achieves R=0.064, P=0.518, F-score=0.114, FPR=0.127 and A=0.322. We collect the value of the metrics for all the 12 VAM_i sets and the 6 IDS models (i.e., 12×6=72 combinations).

R, P and FPR are arranged into the plots in Figure 3 for the sake of a better interpretation. The x-axis is the accuracy of the surrogate model corresponding to a given VAM_i set (i.e, one of the values shown in the rightmost column of Table 3); the y-axis reports R, P and FPR obtained by the model under-test on that VAM_i set. For example, the leftmost points of the •, △ and ×-*marked* data series in Figure 3a correspond to (0.530,0.997), (0.530,0.824) and (0.530, 0.455), i.e., the above-mentioned example with VAM_1 and AE_1. It is worth noting that the strength of the VAM attack decreases as we move from the left to right of each plot in Figure 3.

The plots reveal many interesting findings, which hold with respect to the VAM attack and IDS models at hand (i.e, autoencoder and decision tree).

Robustness and parameters of the IDS models. The metrics of the autoencoder-based IDS behave similarly across AE_1, AE_2 and AE_3 (i.e., Figure 3a, 3b and 3c): R is high for all the cases; on the other hand, P and FPR improve as the accuracy of the surrogate improves. The IDS models based on the decision tree show quite diverse behavior depending on the configuration DT_1, DT_2 and DT_3 (i.e., Figure 3d, 3e and 3f): this is especially true for R. Overall, changes to the model parameters – negligible on the legitimate examples (Table 2) – can strongly affect the robustness to adversarial examples in a supervised approach, such as the decision tree.

Robustness and parameters of the adversarial attack. As said above, different combinations of eps, xi and num_iterations may lead to similar accuracy of the surrogate model. For example, eps=0.7, xi=0.1 and num_iterations=1 cause 0.557 accuracy (surrogate); a similar accuracy can be obtained by lowering eps to 0.5 and setting num_iterations=15. Notwithstanding the similar accuracy of the surrogate, the IDS models may react differently. This can be clearly noted for both FPR (autoencoders) and R (decision tree),



(a) AE_1. (b) AE_2. (c) AE_3.

(d) DT_1. (e) DT_2. (f) DT_3.

Figure 3: Recall (R), precision (P) and false positive rate (FPR) of the IDS models against the adversarial examples with respect to the accuracy of the surrogate model.

which fluctuate strongly over contiguous values of accuracies. For example, in DT_2 R goes from 0.370 to 0.702 when the accuracy (surrogate) goes from 0.530 (VAM_1) to 0.557 (VAM_2); however, R goes back to 0.464 when accuracy (surrogate) increases by just 0.003, i.e., 0.570 (VAM_3). The specific parameters of VAM impact the effectiveness of the attack.

Robustness to overstimulation. Overstimulation pertains to patterns that aim to generate false IDS alerts [10]. A high-level inspection of the FPRs in Figure 3 may suggest that the autoencoder is vulnerable – high FPR – to overstimulation when the accuracy (surrogate) is ≤ 0.603, which corresponds to VAM_6; FPR improves sharply (e.g., it drops from 0.334 to 0.020 in AE_3) when the accuracy (surrogate) is >0.603. The decision tree seems less vulnerable to overstimulation. It is worth noting that a high FPR may underlie the fact the perturbation was strong enough to make a normal example to switch from the normal to the attack class. In this respect, the autoencoder "genuinely" fails at not raising false alerts. This hypothesis is supported by the fact that the highest FPR are noted with strong parameterizations of VAM (i.e, VAM_j with j$\leq$6).

Robustness to evasion. Evasion pertains to an intrusion pattern that is modified so that an IDS will not be able to detect it [10]. According to Figure 3, the autoencoder is robust to evasion attempts. In fact, the R of AE_1, AE_2 and AE_3 is high for all the accuracies of the surrogate model, which means that it is unlikely that an intrusion pattern is misclassified as normal after the perturbation. On the contrary, the decision trees show quite diverse responses to evasion. For example, DT_1, DT_3 are strongly vulnerable to evasion; better results are noted in DT_2, which can tolerate evasion attempts for "weak" parameterizations of VAM, e.g., accuracy (surrogate) equal to 0.839 or 0.900 (rightmost two points of Figure 3e). It is interesting to note that DT_2 is an intermediate parameterization of the decision tree (i.e., not too over- or under-fitted on the legitimate data), which may suggest the existence of a trade-off between the generalization of an IDS model and its robustness to adversarial examples.

6 CONCLUSION

IDS play a key role to protect networks and systems from intrusions. Even though ML(DL)-based IDS can attain high detection performance, they can be fooled out by adversarial examples, obtained by small perturbations applied to the legitimate input. In this paper we studied the robustness of two well-known machine learning techniques for intrusion detection, i.e., deep autoencoders and decision trees, subject to adversarial examples obtained by VAM. The results obtained highlight the strength and weakness of each technique, pointing also out the relevance of their configuration parameters as far as the robustness to adversarial attacks is concerned. A potential limitation of this study consists in the use of a feature-level approach to generate the adversarial examples and their mapping back to the real network traffic, which surely requires a deep exploration of the relationship between feature space and problem space [31]. In this respect, we are aware that the case study presented here is just a first step to a complete understanding of the robustness issues of machine learning-based IDS. Our future work will be devoted to fully explore these topics, considering further machine learning methods under a more complete spectrum of techniques for generating adversarial examples.

REFERENCES

[1] G. Apruzzese, L. Pajola, and M. Conti. 2022. The Cross-Evaluation of Machine Learning-Based Network Intrusion Detection Systems. *IEEE Transactions on Network and Service Management* 19, 4 (2022), 5152–5169.

[2] F. Beer and U. Buehler. 2017. Feature selection for flow-based intrusion detection using rough set theory. In *Proc. International Conference on Networking, Sensing and Control*. IEEE, 617–624.

[3] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli. 2011. Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks. In *Multiple Classifier Systems*, C. Sansone, J. Kittler, and F. Roli (Eds.). Springer, 350–359.

[4] N. Carlini and D. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *Proc. Symposium on Security and Privacy*. IEEE, 39–57.

[5] M. Catillo, A. Del Vecchio, A. Pecchia, and U. Villano. 2022. Transferability of machine learning models learned from public intrusion detection datasets: the CICIDS2017 case study. *Software Quality Journal* 30 (2022), 955–981.

[6] M. Catillo, A. Pecchia, and U. Villano. 2023. CPS-GUARD: Intrusion detection for cyber-physical systems and IoT devices using outlier-aware deep autoencoders. *Computers & Security* 129 (2023), 103210.

[7] M. Catillo, A. Pecchia, and U. Villano. 2023. A Deep Learning Method for Lightweight and Cross-Device IoT Botnet Detection. *Applied Sciences* 13, 2 (2023), 837.

[8] M. Catillo, M. Rak, and U. Villano. 2020. Auto-scaling in the Cloud: Current Status and Perspectives. In *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, L. Barolli, P. Hellinckx, and J. Natwichai (Eds.). Springer, 616–625.

[9] V. Chandola, A. Banerjee, and V. Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys* 41, 3 (2009), 15.

[10] I. Corona, G. Giacinto, and F. Roli. 2013. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences* 239 (2013), 201–225.

[11] M. J De Lucia and C. Cotton. 2019. Adversarial machine learning for cyber security. *Journal of Information Systems Applied Research* 12, 1 (2019), 26.

[12] A. S. Dina and D. Manivannan. 2021. Intrusion detection based on Machine Learning techniques in computer networks. *Internet of Things* 16 (2021), 100462.

[13] G. Engelen, V. Rimmer, and W. Joosen. 2021. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study. In *Proc. Security and Privacy Workshops*. IEEE, 7–12.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[15] K. He, D. D. Kim, and M. R. Asghar. 2023. Adversarial Machine Learning for Network Intrusion Detection Systems: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 25, 1 (2023), 538–566.

[16] L. Hyafil and R. L. Rivest. 1976. Constructing optimal binary decision trees is NP-complete. *Inform. Process. Lett.* 5, 1 (1976), 15–17.

[17] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. 2018. Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables. In *Proc. European Signal Processing Conference*. EURASIP, 533–537.

[18] M. A. Kramer. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* 37, 2 (1991), 233–243.

[19] Y. N. Kunang, S. Nurmaini, D. Stiawan, A. Zarkasi, Firdaus, and Jasmir. 2018. Automatic Features Extraction Using Autoencoder in Intrusion Detection System. In *Proc. International Conference on Electrical Engineering and Computer Science*. IEEE, 219–224.

[20] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Proc. International Conference of Network and Distributed System Security Symposium*.

[21] T. Miyato, S. Maeda, M. Koyama, K. Nakae, and S. Ishii. 2016. Distributional Smoothing with Virtual Adversarial Training. arXiv:1507.00677 [stat.ML]

[22] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *Proc. Conference on Computer Vision and Pattern Recognition*. IEEE, 2574–2582.

[23] N. Moustafa and J. Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Proc. Military Communications and Information Systems Conference*. IEEE, 1–6.

[24] B. Nelson, M. Barreno, F. Jack Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. 2009. Misleading Learners: Co-opting Your Spam Filter. In *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*. Springer, 17–51.

[25] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan. 2019. GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In *Proc. Conference on Communications and Network Security*. IEEE, 91–99.

[26] M. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. M. Molloy, and B. Edwards. 2019. Adversarial Robustness Toolbox v1.0.0. arXiv:1807.01069 [cs.LG]

[27] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. 2021. Deep Learning for Anomaly Detection: A Review. *ACM Computing Surveys* 54, 2 (2021), 38.

[28] N. Papernot, P. McDaniel, and I. Goodfellow. 2016. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. arXiv:1605.07277 [cs.CR]

[29] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proc. Asia Conference on Computer and Communications Security*. ACM, 506–519.

[30] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. 2016. The Limitations of Deep Learning in Adversarial Settings. In *Proc. European Symposium on Security and Privacy*. IEEE, 372–387.

[31] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *Proc. Symposium on Security and Privacy*. IEEE, 1332–1349.

[32] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho. 2019. A survey of network-based intrusion detection data sets. *Computers & Security* 86 (2019), 147–167.

[33] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach. 2021. Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain. *ACM Computing Surveys* 54, 5 (2021), 108.

[34] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proc. International Conference on Information Systems Security and Privacy*. SciTePress, 108–116.

[35] R. S. Siva Kumar, M. Nystrom, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia. 2020. Adversarial Machine Learning-Industry Perspectives. In *Proc. Security and Privacy Workshops*. IEEE, 69–75.

[36] Y. Song, S. Hyun, and Y. Cheong. 2021. Analysis of Autoencoders for Network Intrusion Detection. *Sensors* 21, 13 (2021).

[37] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *Proc. International Conference on Learning Representations*. 1–10.

[38] C. Tsai, Y. Hsu, C. Lin, and W. Lin. 2009. Intrusion detection by machine learning: A review. *Expert Systems with Applications* 36, 10 (2009), 11994–12000.

[39] M. Verkerken, L. D'Hooge, T. Wauters, B. Volckaert, and F. De Turck. 2021. Towards Model Generalization for Intrusion Detection: Unsupervised Machine Learning Techniques. *Journal of Network and Systems Management* 30 (2021), 12.

[40] J. Wu, Y. Wu, N. Niu, and M. Zhou. 2021. MHCPDP: multi-source heterogeneous cross-project defect prediction via multi-source transfer learning and autoencoder. *Software Quality Journal* 29, 2 (2021), 405–430.

[41] L. XuKui, C. Wei, Z. Qianru, and W. Lifa. 2020. Building Auto-Encoder Intrusion Detection System based on random forest feature selection. *Computers & Security* 95 (2020), 101851.

[42] Z. Yang, X. Liu, T. Li, D. Wu, J. Wang, Y. Zhao, and H. Han. 2022. A systematic literature review of methods and datasets for anomaly-based network intrusion detection. *Computers & Security* 116 (2022), 102675.