UNIVERSITÄT KOBLENZ · LANDAU

IWVI

Adversarial Attacks on ML-based Intrusion Detection

Master Thesis

Master of Science (M.Sc.) in Web & Data Science

Submitted by

Lokesh Sharma

[219100827]

Koblenz, October 2021

Erstgutachter: Prof. Dr. Andreas Mauthe
(Institut für Wirtschafts- und Verwaltungsinformatik, FG Mauthe)

Zweitgutachter: Alexander Rosenbaum, M. Sc.
(Institut für Wirtschafts- und Verwaltungsinformatik, FG Mauthe)

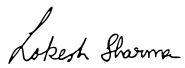
Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden. ja ■ nein □

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ■ nein □

Koblenz, 05.10.2021
(Ort, Datum)


(Unterschrift)

Zusammenfassung

Maschinelles Lernen (ML) ist eine weit verbreitete Methode zur Erkennung von Anomalien in vielen vernetzten Systemen. Trotz ihrer guten Performanz sind tiefe Neuronale Netzwerke (Deep Neural Networks) anfällig für semantische Änderungen in der Eingabe (z. B. durch feindlich Angrife - evasion attacks), was für sicherheitskritische Anwendungen ein erhebliches Risiko darstellt. Bislang wurde maschinelles Lernen in erster Linie in der Bildverarbeitung erforscht und ist in anderen Datenbasierten Anwendungsbereichen noch im Entstehen begriffen. In dieser Dissertation wird die Robustheit von ML-basierten Systemen zur Erkennung von Netzwerkangriffe bewertet, indem die Eingaben zum Testzeitpunkt verfälscht werden (*evasion attacks*). Der Hauptbeitrag dieser Arbeit besteht darin, die Unwahrnehmbarkeit, Funktionalität und Eingabekohärenz der Testmuster beizubehalten, um damit das trainierte Modell zu umgehen. Um dieses Ziel zu erreichen, werden zwei State-of-the-Art gradientenbasierte Algorithmen beschränkt zu erweitern, um realistische Angriffe auf einen Klassifikator zur Erkennung von Eindringlingen zu erzeugen. Die Untersuchungen zeigen, dass in engen Bereichen Angriffe möglich sind, und es wird auch gezeigt, dass Einschränkungen eine Domäne nicht widerstandsähiger machen.

Abstract

Machine learning is a widely used methodology for anomaly detection in most networked systems. Despite their high performance, deep neural networks are vulnerable through semantic modification of the inputs (e.g., adversarial examples), posing significant risks for security-critical applications. So far, adversarial machine learning is primarily studied in computer vision and is still emerging in other data domains. In this dissertation, the robustness of ML-based network intrusion detection systems is evaluated by perturbing the input samples at test time (*evasion attacks*). The main contribution is maintaining the test samples' imperceptibility, functionality, and input coherence while evading a trained model. In pursuit of this objective, two state-of-the-art gradient-based algorithms are extended in a constrained space to generate realistic attacks against a classifier for intrusion detection. The investigations show that the narrow threat surface is still sufficiently large for an adversary, and constraints do not make a domain resilient.

[0] https://github.com/hsekol-hub/Adversarial-Attacks-ML-based-IDS

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abbreviations

ACSC	Australian Cyber Security Centre
AE	Adversarial Example
AIDS	Anamoly-based Intrusion Detection System
AML	Adversarial Machine Learning
AUC	Area Under Curve
CAIDA	Center for Applied Internet Data Analysis
CIC	Canadian Institute for Cybersecurity
CNN	Convolutional Neural Network
CSV	Comma Separated Files
DARPA	Defense Advanced Research Project
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
DoS	Denial of Service
DT	Decision Tree
FGSM	Fast Gradient Sign Method
HIDS	Host-based Intrusion Detection System
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
JSMA	Jacobian-based Saliency Map Attack
KDD	Knowledge Discovery and Data Mining

KNN k-Nearest Neighbours

L-BFGS Limited-memory Broyden Fletcher Goldfarb Shanno

LDAP Lightweight Directory Access Protocol

MC-SQLR Microsoft SQL Server Resolution Protocol

MIDS Misuse-based Intrusion Detection System

ML Machine Learning

NB Naive Bayes

NetBIOS Network Basic Input/Output System

NIDS Network-based Intrusion Detection System

NN Neural Network

OSI Open System Interconnection

PCAP Packet Capture

RF Random Forest

RNN Recurrent Neural Network

R2L Root to Local

ROC Receiver Operating Characteristic

SVM Support Vector Machine

TCP Transmission Control Protocol

U2R User to Root

UDP User Datagram Protocol

VAE Variational Auto Encoders

1. Introduction

Networked systems such as power grids, government infrastructures, critical communication systems, etc.. are the backbone of our present society. Due to their significance, they are under a constant threat of malicious intrusions, failure, availability, or operational overload, and this makes cyber security an important field of research. Cyber attacks target cloud-based infrastructures, and with big data, the situation of cybercrime is severely increasing [5]. The failure of a network system can compromise several businesses and impact physical lives today.

An intrusion is any unauthorized activity to exploit the networked system, such as obtain confidential information, deny availability to legitimate users or degrade the integrity [6]. Although most network systems have continuously evolved in recent years with sophisticated security patches, they often suffer from a high false alarm rate or cannot detect unknown intrusions [7]. In order to address this situation, much of the focus has been on leveraging machine learning from anomaly detection. To this advancement, ML-based IDS has achieved success in identifying novel malicious network traffic with sufficient training data [8]. In addition, the intrinsic generalization nature of ML classifiers refrains domain knowledge from being a bottleneck while designing an IDS.

Despite these developments and increasing demand for machine learning algorithms in several real-world applications, like computer vision, self-driving vehicles, facial recognition, and cybersecurity, the attack surface is still large due to adaptive adversaries and is a rising concern. The primary focus when utilizing machine learning services has been on the overall performance, which shadows the security aspects. It is essential to understand the significant risks and vulnerabilities in the trained ML models before they are put into production. Research on their safety is limited, and thus, this thesis contributes to evaluating the robustness of ML models for network traffic intrusions against an adversary.

In 2014 [9] discovered the vulnerabilities of neural networks, where noise is applied to a data instance to cause misclassification. The perturbed sample was imperceptible to human perception but was able to circumvent the trained classifier. This downside of using ML has been extensively researched in the image domain, but they also exist in sequential and tabular datasets [10].

An *Adversarial Example (AE)* is an input data slightly perturbed by some noise with intentions to deceive the classifier and force it into making a mistake. Since the discovery of adversarial examples, there has been an arms race between the attackers and the defenders [11–13]. Most of the prior work on adversarial machine learning has been in an unconstrained domain where the input is perturbed freely across all the features [14]. The scope of this work is to extend this sub-field of machine learning in a constrained scenario and expose the vulnerabilities of ML-based intrusion detection systems.

1.1. Research Questions

The attack vector of adversarial examples stretches beyond image recognition tasks and is an existing threat in text, speech recognition, and intrusion detection. In other words, no domain (thus far) is robust against adversarial attacks, and the myriad domains they serve are vulnerable. Many industrial applications still rely on tabular datasets, like fraud detection, stock prediction, and intrusion detection. However, the research community has wholly focussed on the unconstrained scenarios where the freedom overestimates the capabilities of an adversary. For example, the input feature values are often bound by the semantics or the functionality of the actual class label. Implicit to this argument is that such attack algorithms need to be questioned on their effectiveness in crafting the adversarial counterparts in tabular contexts [15].

In particular, for a network traffic sample, the adversary should be bound in its capabilities not to perturb the functional features (for example, network packets should always obey TCP/IP protocol). Furthermore, these constraints also extend to describe the correlation among features and the semantic structure. As a part of the literature review, some gaps have been identified:

- The existing research studies adversarial attacks on a synthetically generated dataset that do not represent the modern, sophisticated intrusions (e.g., distributed denial of service).
- ML classifiers broadly used in IDS literature do not seem to provide enough evidence for robustness in the adversarial setting.
- Deep learning-based methods have been used to manipulate the original input without any restrictions, which exploits the imperceptibility of the perturbation.

Naturally, this motivates to the following research questions:

1. How effective are the attack algorithms against the newer intrusion datasets?
2. Which attack methods are applicable in tabular domains, where features vary in their data types (e.g., binary, discrete, or continuous)?

3. Do adversarial examples exist in constrained scenarios maintaining the input coherence and the functionality of a network traffic instance?

4. Is it possible to measure (as a qualitative/quantitative value) the imperceptibility of perturbation without a domain expertise?

5. Are these adversarial counterparts transferable across models and datasets?

1.2. Key Contributions

Unlike previous work, in this dissertation, the union of both adversary and domain constraints is considered. An extensive background review of the existing literature on ML-based intrusion detection systems and adversarial machine learning is performed. Next, the adversarial samples are generated for three publicly available datasets: NSL-KDD, CICIDS2017, and CICDDoS2019. The core idea is to craft adversarial attacks as an optimization problem and test the hypothesis if constrained domains are less vulnerable to the techniques of Adversarial Machine Learning (AML). The characteristics that define the constraints are:

- Values of different features can be correlated.
- Features can have mixed data types (discrete vs. continuous), and the adversary should maintain this order.
- Some features are not to be manipulated and should not be controlled by the adversary.

In summary, the main four contributions are as follows:

1. Introduce a formalization to build the threat model, dynamically express the functionality constraints and the perturbation space for each class label based on the training dataset.
2. Define a metric to measure the imperceptibility of the generated adversarial set.
3. Extend two gradient-based algorithms, LowProFool and DeepFool, for network intrusion detection and provide a comparison of their success rates.
4. Craft targeted adversarial attacks in a constrained space with partial perturbation of the feature set based on their prior relative importance.

1.3. Outline

The remainder of this thesis is structured into five additional chapters and is structured as follows:

- **Chapter 2:** introduces the background knowledge of IDS systems and the necessary theory of machine learning used for intrusion detection. In this chapter, we shall also discuss on security evaluation of the classical design methods and expose the vulnerabilities in greater detail.
- **Chapter 3:** follows a top-down approach beginning with more general concepts relating to adversarial machine learning and different attack types. Next, we define the threat model for the adversary and discuss the existing approaches in the tabular domain. In the end, some metrics are formalized for evaluating the robustness of the deployed IDS models in terms of baseline performance and in an adversarial context.
- **Chapter 4:** describes the dataset selection and a brief theory on the fundamentals of different class labels and their functionalities. The subsequent sub-sections describe the data preprocessing and baseline model training pipelines. In the end, the functionality constraints, the threat surface, and attack algorithms are mathematically formalized.
- **Chapter 5:** presents the different experiments performed to answer the research questions, and the results are analyzed with respect to the metrics and meaningful plots.
- **Chapter 6:** concludes the findings, limitations, and suggestions for future work to continue and improve our research.

2. Machine Learning in IDS

In this section, the required preliminaries relating to areas of machine learning and cyber-security are summarized. The discussion begins with IDS taxonomy, and then the state-of-the-art machine learning algorithms used in IDS. In the end, the security evaluation and vulnerabilities of these classifiers are exposed.

2.1. Intrusion Detection System Taxonomy

An *Intrusion Detection System* is a computer-security program that detects a wide range of security violations and malicious traffic. Any attempt to access information (unauthorized) from a network system or damage its operations is flagged as an intrusion. Some main functionalities of an IDS include monitoring host machines or networks, behavioral analysis of computer systems, triggering alerts, and responding to any suspicious intrusions. If these IDS are placed in such a manner that they can also stop these intrusions, then they are termed as *Intrusion Prevention System (IPS)*. A broad range of algorithms is used to build discriminative models that can understand network traffic patterns. [16] critically examines different techniques and challenges in IDS

The design principles for these models are based on two main application scenarios: data source-based methods and detection-based methods [17, 18]. The former is further classified into *Misuse-based Intrusion Detection System (MIDS)* and *Anamoly-based Intrusion Detection System (AIDS)* and the latter into *Host-based Intrusion Detection System (HIDS)* and *NIDS*. Some primary concepts from the existing taxonomy are summarized below:

2.1.1. Classification based on detection methodology

1. **MIDS:** It is also known as *signature-based or knowledge-based IDS*. The main working principle of these models is to leverage the previously saved signatures of attack behaviors, i.e., match the signature of the current network traffic against the existing database for intrusion detection. A major advantage of these systems is the low false alarm rate while identifying intrusions and also provides a possible detailed explanation. However, they have common issues like designing efficient signatures, maintaining huge databases, and the inability to detect unknown attacks. Figure 2.1 provides a generic working of such an architecture. Some examples of MIDS include methods with pattern matching, expert

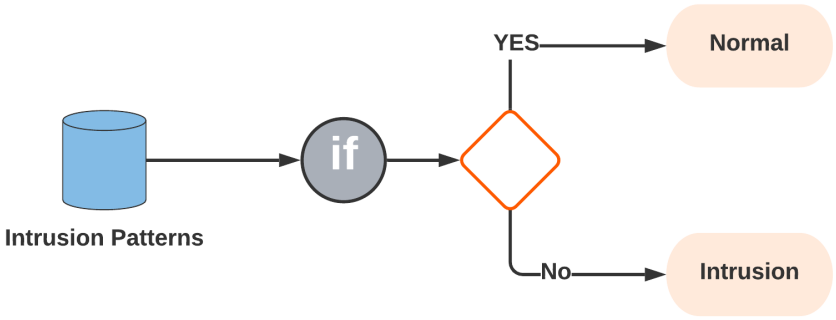


Abbildung 2.1.: Conceptual working of signature-based methods

systems, or finite-state machines.

2. **AIDS:** It is one widely studied architecture where the critical design is to profile an average traffic flow. These methods have drawn significant interest from the researchers due to their ability to overcome the shortcomings of MIDSs. They build a model of the behavior of a network system using methods like statistical models, machine learning, time series analysis and show strong generalizations towards detecting unknown intrusions. However, some evident shortcomings are high false alarm rates, network traffic diversity, class imbalance problems, and the inability to explain the malicious flag [19].

	Advantages	**Disadvantages**
MIDS	▪ very low false alarms. ▪ prompt intrusion detection. ▪ simple architectures. ▪ known attacks is easily identified.	▪ needs frequent updates of new signatures. ▪ can't detect a new variant of malicious traffic. ▪ is unable to detect zero-day attacks. [1]
AIDS	▪ can detect new attacks. ▪ creates standard intrusion signatures. ▪ anomalies are easily identified.	▪ can't identify encrypted packets. ▪ high false alarm rates. ▪ initial model training is required.

Tabelle 2.1.: Comparison of detection-based architectures

[1] Zero-day attacks refer to a newly discovered software vulnerability for which an official patch is still in development. In the meantime, the system is vulnerable to be exploited by hackers

2.1.2. Classification based on data source

1. **NIDS:** It monitors the network traffic packets and router NetFlow records at different TCP/IP protocol layers to detect intrusions. A flow is a set of packets, i.e., the basic units of network communication, and a packet usually contains packet headers and payload information. Most of these models are independent of the operating system, where feature extraction methods are used for packet detection. [20,21] studies on their internal working and architecture details.

2. **HIDS:** It is usually installed on the host machines to inspect malicious activities like illegal processes, log files (firewall logs, window server logs, etc.), or any unexpected changes in the host itself. They mostly use the audit logs as their primary data source and only focus on detecting insider threats. Their work is a mixture of custom handcrafted rules and machine learning models relying on database logsâa clear advantage of these systems is to locate intrusions precisely [19].

	Advantages	**Disadvantages**
HIDS	▪ No extra hardware required. ▪ can check end-to-end encrypted communication behavior. ▪ every packet is reassembled. ▪ checks host's file system, network events, and systems calls.	▪ Consumes host resources. ▪ installation on each host machine. ▪ delays in reporting attacks. ▪ can't identify network traffic.
NIDS	▪ monitors network packets. ▪ installations are not required. ▪ capable for a wide range of network protocols. ▪ work on multiple hosts.	▪ can't identify encrypted packets. ▪ dedicate hardware is required. ▪ supports the only detection of network packets. ▪ challenging to analyze the high-speed network.

Tabelle 2.2.: Comparison of data source-based architectures

2.2. Classifiers in NIDS

As discussed in Section 2.1.2, an anomaly-based NIDS is widely used in the cybersecurity industry due to its advantages like detecting zero-day attacks, robust customization to any architecture, and mainly learning patterns in network traffic flows. Despite these advantages, often hybrid techniques that combine AIDS and MIDS are used to minimize the high false alarm rates and provide faster detection. In the following sub sections, some ML algorithms that have been employed in NIDS are discussed, and [22] provides a detailed study on the applications of machine learning in network security aspects. The Figure 2.2 describes a conceptual working of machine learning-based IDS.

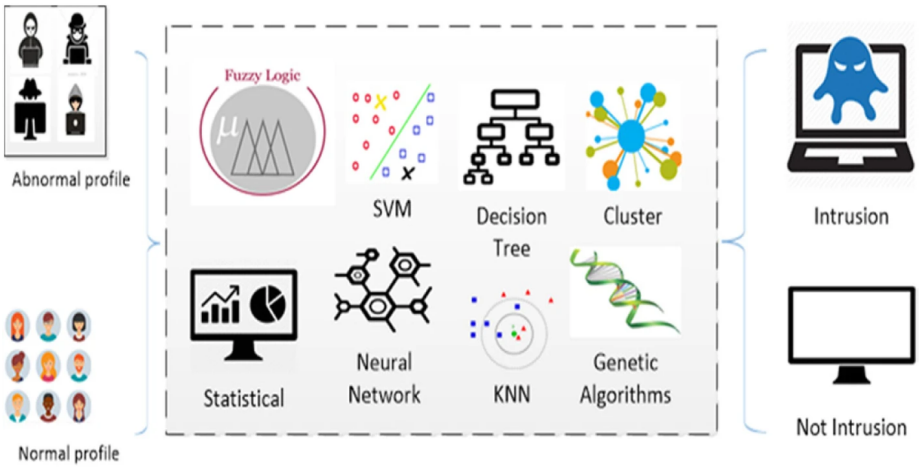


Abbildung 2.2.: Conceptual working of machine learning-based methods. [2]

Prior studies from [23, 24] provides a comprehensive overview of the recent work on supervised and unsupervised algorithms in network intrusion scenarios. Supervised methods consider *ground truth* of the output label being predicted and update the model's mapping parameters to learn the relationship with the input feature values (e.g., Decision Tree (DT), Support Vector Machine (SVM), Neural Network (NN), etc.). On the other hand, unsupervised methods do not have accurate label information and make clusters based on distance metric (e.g., k-means clustering, k-Nearest Neighbours (KNN), variational autoencoders, etc.). In this work, the primary focus is on supervised machine learning algorithms and the classifiers are categorized in the following tribes based on their working principle.

[2] Source: Survey of intrusion detection systems: techniques, datasets, and challenges

- **Bayesian:** It is a generative model that outputs the probability of a class label conditional on the feature values. An Naive Bayes (NB) classifier, based on Bayes' principle [3] assumes a robust independence among the features. However, in intrusion datasets, features are highly co-related, and based on the evaluation results from [25], these algorithms do not work very well with IDS. Mathematically stated in Equation 2.1,

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{2.1}$$

  where, $P(A|B)$ is a conditional probability of event A given B is true, is termed as the *posterior probability*, $P(B|A)$ is the *class likelihood model*, P(A), and P(B) are termed as the *marginal probabilities*

- **Evolutionaries:** The basic principle in this set of algorithms is based on the evolution process in genomes and DNA to process the data. A possible solution is represented as a series of bits, and their quality improves over time through selections, also referred to as *performance of data*. These approaches are always based on heuristics, where the survival and offspring of units are essentially considered data performance. The algorithm starts with a set of individuals, and the objective is to retain or favor fitter solutions. However, these algorithms are often challenging to converge and are considered NP-hard problems [26]. For nontrivial reasons, the internal working of genetic algorithms in intrusion scenarios are omitted, and interested readers can refer to [27].

- **Analogizers:** are discriminative models with the fundamental idea that nearby elements in an N-dimensional space are strongly related. They mostly use a distance metric (Euclidean distance, cosine angle, KL divergence, etc.) to understand how two samples are distant from each other. Some widely used algorithms in intrusion scenarios include KNN and SVM. The first uses a distance function to find k-nearest neighbors in a fixed feature space and predicts the class label based on the distribution mode of closest neighbors. The second algorithm works in a binary classification task, where a hyperplane and margin are built to separate the data into two halves. The objective function is to maximize the margin and allow better generalization. Both of these algorithms are supervised methods of learning and have been widely studied in the research community. However, these methods are not scalable with multiple class labels and face limitations when minor changes are performed in the packet structure (like flipping bits, adding malicious information in the payload, etc.).

[3] In probability theory, Bayes' theorem states the probability of any event is dependent on prior knowledge of conditions related to that event.

- **Symbolists:** The core working principle of these algorithms is inverse deduction, i.e., they start with some premises and then fill the gaps (e.g., decision tree and *random forest*). A decision tree is a rule-based *tree-structure* where the root node represents a test to an attribute and branch splits are all the possible outcomes. A Random Forest (RF) is merely an ensemble of *n* independent decision trees. The main advantages of these classification models are their easy implementation, high classification accuracy, and interpretability. The main idea is to traverse from a root node to a leaf node (which represents class label) based on *information gain* [28, 29]. Further, features with increasing information gain are preferred when traversing during a classification task.

  Equation 2.2 is the mathematical notation for calculating information gain, where T and A represents *target* and *feature attribute* respectively.

$$Gain(T, A) = Entropy(T) - \sum_{v \in A} \frac{|T_v|}{T}.Entropy(T_v) \tag{2.2}$$

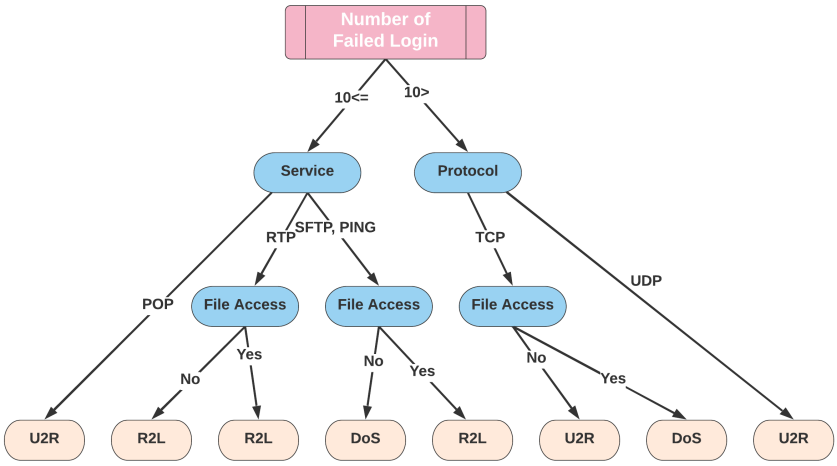


Abbildung 2.3.: Path traversal in a decision tree for intrusion detection.

From the intrusion detection perspective, network logs are inspected for classifying malicious and normal labels. However, these trained models need to be installed in each network, and researchers use several choices to ensure security (e.g., J48, ID3, C4.5, etc.). [17] demonstrated the C4.5 DT algorithm to perform the best on the NSL-KDD dataset. Figure. 2.3 provides an illustration of the path traversal in a decision tree for a classification problem.

- **Connectionists:** This tribe of algorithms focuses on reverse-engineering the neural connections in a human brain. The most common architecture involves connecting artificial neurons (non-linear) in a network consisting of several layers. Much to our expectation, their application is broad and is very successful in detecting intrusions. The most common learning technique in these algorithms is backpropagation, where gradients of network error are calculated concerning its trainable parameters. However, a data-driven approach does not perform well with low-frequency labels (e.g., User to Root (U2R) in the NSL-KDD dataset). Much work has been done exploring different variants (Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), Variational Auto Encoders (VAE)) under this algorithm category [16, 19].

  Their work involves passing the feature values from the input layer through the hidden layer(s) and an output layer that provides each class label's probability. Objective functions are designed to calculate the error, and the model undergoes training via backpropagation. A neural network consisting of multiple hidden layers is termed as *deep neural network* and is capable of solving complex tasks, including inputs it has never been trained on before. Figure 2.4 shows the internal architecture of a deep neural network for classification task scenarios on the NSL-KDD dataset.

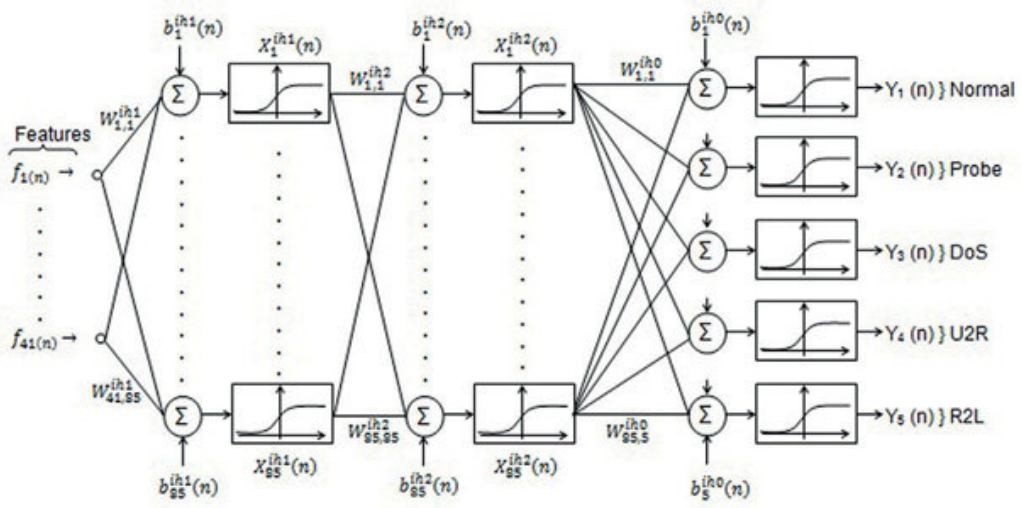


Abbildung 2.4.: Working of deep neural network on NSL-KDD. [4]

[4] Source: Network Intrusion Detection: Usability Comparison on Neural Networks [30]

2.3. Security Evaluation of Deep Neural Networks

Most machine learning algorithms [5] are overly dependent on the assumed statistics of the input data, i.e., a close attachment to the input makes them highly sensitive to a slight distribution shift [1, 3, 9]. In security-sensitive applications like malware detection, spam filtering, biometric authentication, and intrusion detection, the success of ML entirely depends on training and test data distributions. The classical design settings of these classifiers assume the underlying data distribution to be *stationary*. [31, 32] questions the classical design principles of the existing methodologies and discusses their dependency on the training data. They further expose the vulnerable areas, leading to various security menaces, where an attacker actively manipulates the data to make the model produce false negatives. These vulnerabilities of machine learning have become a serious concern regarding safety-, privacy-, and especially security, as attacks on these deployed models can cause severe damage.

2.3.1. Existing Vulnerabilities

[9] first showed that slight perturbations in the input data distribution could deceive deep neural networks. They discuss two counter-intuitive properties of neural networks that expose the intrinsic blind spots (relating to the semantic information in the higher dimensional space and their stability for additional noise). These carefully perturbed input samples obtained via optimization are termed as *AE* and are capable of fooling any machine learning algorithm by maximizing the prediction error.

1. **Why do adversarial examples exist?**

   Unit-level inspection of a neural network gives us an idea of its complexity, as the output distribution represents a non-linear operation on the inputs. The high modeling power of the neural network (a dense stack of non-linear layers) to represent any arbitrary function leads to an irregular, noisy loss surface. Now, in a high dimensional space, there exists a likelihood that the gradient of the loss surface at any point in the input space is very steep in a specific dimension. An effect of this behavior causes the learned model parameters to change dramatically, i.e., cross a decision boundary. Thus it is also justified why carefully chosen small perturbations easily misclassify.

   In simple words, an adversarial example is a perturbation or noise added to the original input feature set. The key idea is to carefully compute this perturbation, ensuring its distance from the original input sample stays minimum but crosses the discriminative boundary of the target classifier. A common approach is to formulate this as a non-linear

[5] all ML algorithms are vulnerable to adversarial examples, except RBF and parse and density estimators [11]

optimization problem, where the input feature vector **x** is updated instead of the model parameters θ via gradient descent. The intuition is finding the direction that results in a disparity between the semantic information and a decision change.

2. **Are adversarial perturbations and noise the same?**

   [33] probed the robustness of well-known classifiers and proved them vulnerable against both random and well-crafted perturbed noise. It is important to note that neural networks are robust when noise is taken in random dimensions but not the worst-case ones. Thus, the perturbation in the adversarial sample is not noise. For example:

   a) If some noise is added to an adversarial example, it will still be adversarial.

   b) If some noise is added to a clean example, it will still be a clean sample.

[1] further proves how easily these adversarial samples are generated by traversing back in the higher dimensional latent space. Their work also claims that these AEs are model agnostic, i.e., can be transferred from one model architecture to another and across different sets of training data from the same domain. Some key observations are:

1. AE are not due to overfitting or non-linearity of complex models. If that was the case, the respective model should misclassify a single adversarial sample randomly and uniquely. However, a pattern was observed for their existence.

2. AE generalized across cross-model and cross dataset. Many models trained with different architectures misclassified a single adversarial sample and predicted the same target label.

3. A systematic effect is observed. For example, the offset of the original and perturbed input set gave a direction in the input space. This vector, when added to any clean input feature values, mostly gave an adversarial example.

2.3.2. Design Principles

As discussed, the nature of adversarial data towards ML models. At present, there is an arms race between the system designers and their adversaries, which can be acknowledged by the increasing complexity of modern attacks and defense mechanisms. Moreover, the classical design methods for building ML pipelines are not resilient against clever adaptive attacks, raising serious security concerns.

[34] proposed a robustness and security evaluation framework as either a reactive or proactive arms race. First, the adversary and the system designer aim to attain their goals by updating their actions concerning the other, i.e., learning from their past behaviors. However, one can

point out that this approach cannot guarantee against never-before-seen attack scenarios. In the latter, the proactive paradigm, the system designer anticipates an attacker and evaluates the vulnerable aspects of the model under the adversarial setting. Then appropriate defense mechanisms are built iteratively before their deployment.

In order to understand better the security properties of these systems in an adversarial setting. [11] proposes a paradigm of security engineering and cryptography and is summarized as:

1. Investigate potential vulnerabilities of the model before an adversary is exploiting them.
2. Evaluate classifier security and the impact of corresponding attacks.
3. Develop appropriate countermeasures if an attack can cause the classifier to drop the performance significantly.

3. Adversarial Machine Learning in Constrained Domains

This chapter describes the closely related work under the scope of this dissertation, specifically, AML in the context of intrusion datasets. The focus is on DoS and DDoS attacks as they are good examples for the initial studies and are also prevalent in network traffic intrusions. Since most of the work relating to adversarial attacks is in the image domain, search keyword was extended to all the intrusion and malware scenarios while conducting this literature review. Searches on well-known science databases were performed like Springer, arXiv, IEEE Xplore, and Research Gate. In addition to these general queries, keywords like 'Intrusion/Malware Detection", 'Machine learning-based intrusion detection systems,Adversarial machine learning, Adversarial examples on intrusion datasets,änd all relevant publications closely working on intrusion scenarios were selected.

[9] After the first notable discovery of adversarial attacks in computer vision, there has been much work in the research community to understand their applicability in other data domains like textual, tabular, and sequential. In response to the existing security and privacy threats, there has been a momentum in this research sub-field which is closely in line with ML and computer security work to find solutions for the problems discussed in Section 2.3.

AML is the study of ML in the presence of an adversary that aims to degrade the performance of a trained classifier/model on any specific task. A common approach is to modify the input data either during the training phase, termed as *poisoning attacks* or during the inference phase, termed as *evasion attacks* [35]. In the scope of this work, the evasion attack scenarios are studied as they are the most researched types of attacks. Moreover, they are practical as the adversary is seldom allowed access to training data and can only access the model after its deployment, which closely resembles the real scenarios. Figure 3.1 represents an overview of AML where a perturbation δ is added to the original test sample iteratively until the trained ML model misclassifies it.

To this end, let us closely follow the existing taxonomies from [11, 31] and summarize a cumulative framework for the evasion attack scenario. An optimal attack strategy during evasion (test-time) requires an adversary to precisely define the assumptions relating to its goals,

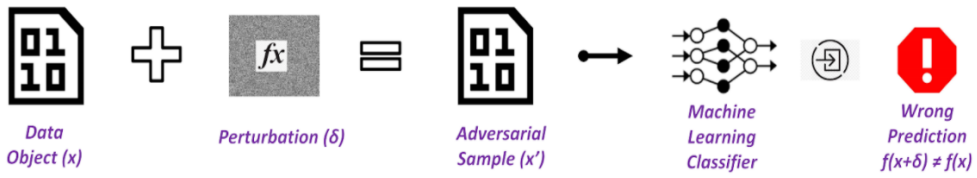


Abbildung 3.1.: Adversarial Machine Learning

knowledge, and capabilities. However, to make an informed decision, let us lay some groundwork for analyzing the adversary's knowledge about the target classifier. Formalizing some main components of the adversary's knowledge as the following:

- the training data or a proportion of it represented as $\mathcal{D}$.
- the type of learning algorithm $\mathbf{h}_\theta$.
- the model parameters and hyper-parameters $\mathbf{w}$.
- feedback from the classifier, i.e., the output labels or confidence scores.
- the feature representation set $\mathbb{X}$ for each sample vector $\mathbf{x}$, i.e., how network packets are represented into the classifier's feature space.

The information mentioned above can be available to the adversary at different levels and is categorized into a white-box, gray-box, or black-box attack.

1. **Perfect-Knowledge (PK) or White-Box Attack:** the adversary assumes to know everything about the target system and also its trainable parameters. This setting allows one to perform a worst-case security evaluation for any trained model and generate potent attacks. Formally, $\theta_{PK} = (\mathcal{D}, \mathbb{X}, \mathbf{h}_\theta, \mathbf{w})$.
2. **Limited-Knowledge (LK) or Gray-Box Attacks:** here the adversary has partial knowledge of the above-discussed components. In most cases, the adversary assumption of knowing the feature representation $\mathbb{X}$ and the classifier $\mathbf{h}_\theta$ is considered. Still, the training data of $\mathcal{D}$ or the parameters $\mathbf{w}$ are unknown. However, the adversary is allowed to create a surrogate model or dataset. Formally, $\theta_{LK} = (\hat{\mathcal{D}}, \mathbb{X}, \mathbf{h}_\theta, \hat{\mathbf{w}})$ [1].
3. **Zero-Knowledge (ZK) or Black-Box Attacks:** [36–38] claims that machine learning can be threatened without any substantial knowledge of the feature space, the learning algorithm, or the training data. An adversary can only leverage the prediction from past inputs to infer the model's vulnerability. [39] suggests black-box attacks are a more realistic threat; however, frequent querying of the model is required and is seldom studied in the intrusion scenarios. To characterize this setting mathematically, $\theta_{ZK} = (\hat{\mathcal{D}}, \hat{\mathbb{X}}, \hat{\mathbf{h}}_\theta, \hat{\mathbf{w}})$.

[1] The hat symbol to denotes limited knowledge of a given component

3.1. Attack Types

Based on [19], the taxonomy of different attack types is summarized across three main dimensions, the influence on the classifiers, the security violation type, and the attack specificity.

1. **Influence on the classifiers' perspective:** To exploit the inherent vulnerabilities, i.e., crack the system through spoofing, the ML security threats are classified into *causative attacks* and *exploratory attacks*.
   - **Causative attacks:** are also known as *poisoning* attacks, where the adversary can alter the training data [2] or the model parameters by injecting a small number of poisoned samples at the training phase. They can be performed either through input feature manipulations, changing the actual class labels, or altering model parameters - '*logic corruption*.' This behavior changes the model's bias towards malicious samples during the training and aims to drop the overall performance at test time. More details on this can be referred from [40, 41].
   - **Exploratory attacks:** are also known as *evasion* attacks, and they involve fabricating adversarial examples during test (inference) time. Because training data is not available or guarded for confidentiality in cyber-security domains, exploratory attacks are considered more practical as the adversary has no access to training data in real scenarios. [42] This category of attacks is extensively studied and proved effective in evading detection with a remarkable drop in the systems' overall security performance. Figure 3.2 shows the block diagram representation of an exploratory attack.
2. **Security violation perspective:** Another line of work suggests the ML security threats should be grouped considering the type of security violation on the targeted system and is sub-categorized into *integrity, availability, and privacy violation* attacks.
   - **Integrity attacks:** attempt to undermine the integrity of the target model during the inference phase, i.e., increase the false-negative rates or alter any other accuracy metric as discussed in Section 3.5. They do not aim at damaging the model's normal operations.
   - **Availability attacks:** target to compromise the system's expected functionalities for a group of genuine users, i.e., reduce the prediction confidence, or its speed, manipulate the performance, or increase false-positive rates for the benign samples. This category of attack denies legitimate access to its users, for example, DoS or DDoS attack.
   - **Privacy violation attacks:** attempt to steal private or confidential information about the training data and the model configuration by reverse-engineering methodologies. For example, in the financial market system, a trained model is an

intellectual property and must be hidden. Similarly, in cyber-security, the intrusion datasets are sensitive and should not be exposed to an adversary.

3. **Attack specificity perspective:** The third feature of this taxonomy is the attack's specificity, and the sub-categories are *non-targeted and targeted* attacks. Targeted attack aims to cause misclassifications to a specific source or target label, whereas a non-targeted also known as *indiscriminate* attack goals to produce a misclassification to any other class label except the ground truth.

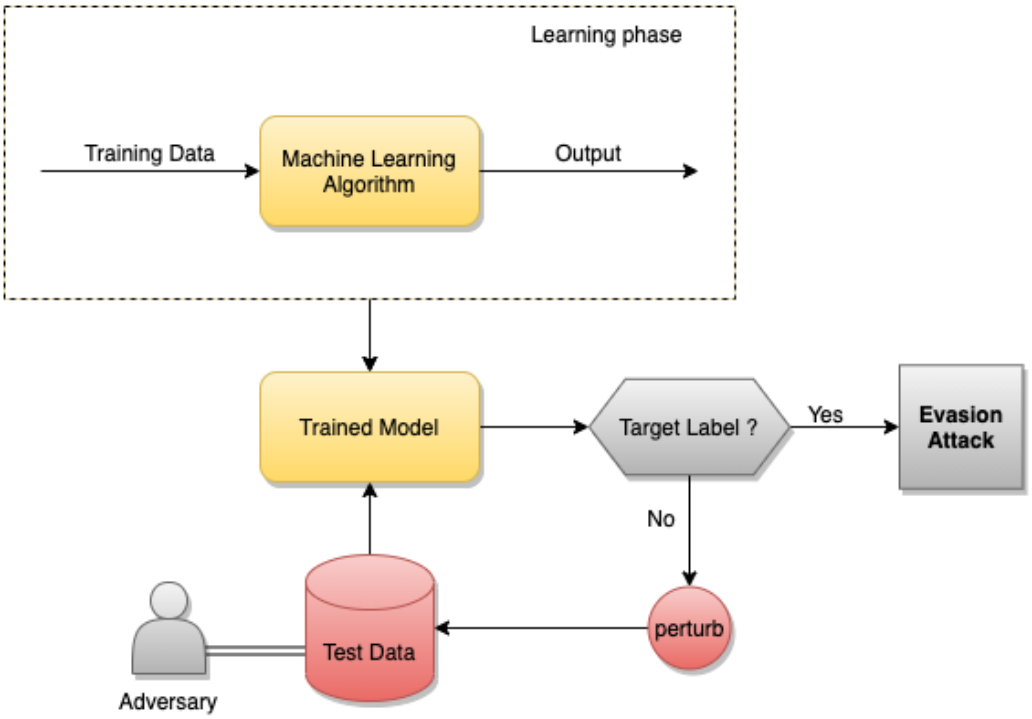


Abbildung 3.2.: Evasion Attacks during deployment

[2]Because training data contributes majorly in setting up the upper bound or baseline for the model's performance.

3.2. Threat Model

The first popular attack taxonomies had been proposed in [43, 44] and their subsequent extended versions in [39, 45]. In this work, the existing frameworks are exploited to deceive a previously trained classifier, i.e., manipulate data during the deployment or test phase by adding imperceptible perturbations. To motivate an optimal attack under an evasion scenario, let us consolidate the design principles from [11, 46] and formalize a precise definition of the threat model.

Why is it essential to define a threat model?

A threat model is an integral component of the defense for any deployed ML model, as it outlines the conditions under which such a system is considered secure. In other words, a threat model precisely specifies what type of actual attacker the defense intends to defend itself. In order to guarantee the robustness of any system, there must exist an experiment that contradicts its claims. Validating the robustness claims or an empirical defense evaluation requires an adversary to experiment with attack strategies to deceive the model. [11] A threat model defines the fundamental decisions for the adversary to formalize an attack strategy. Thus, there is an obvious need for a precise formalization that clarifies the threat scenarios of any deployed ML model for risk evaluation.

Assumption: the adversary acts rationally to obtain its goals, as per the knowledge about the classifier (victim model) and its capabilities to manipulate the input data distribution under a set of constraints.

3.2.1. Adversary Goals

As per [47] the adversary's goals should be formulated to optimize a utility (loss) function. The adversary's goal is to maximize or minimize this objective function. Theoretically, the adversary must force the victim model to make a false prediction on perturbed input samples. Nevertheless, this can be a broad objective and needs further discussion on the type of output label the adversary desires (specificity). As discussed in the related work section 3.1 that this function can be based on:

- the type of desired security violation (availability, integrity or privacy)
- the type of desired attack specificity (targeted or indiscriminate)

Additionally, the scope of this perturbation needs to be defined, also known as *perturbation space* to be successfully called an adversarial sample, i.e., it should be imperceptible or incon-

spicuous. For example, after adding some slight perturbation, the sample should hold its proper functionality. As imperceptibility can be abstract by definition, *perturbation imperceptibility* is considered for tabular intrusion datasets.

3.2.2. Adversary Knowledge

Previous work has focussed chiefly on a qualitative discussion of adversary knowledge. A systematic scheme of the adversary's knowledge about the target classifier and data may vary depending on the application scenario. This may include:

- the training dataset or a proportion of it.
- the feature representation of samples; i.e., how network packets are mapped into the target classifier's feature space.
- the type of learning algorithm or its decision function (e.g., decision tree, SVM, etc.)
- the feedback available from the victim model, i.e., confidence scores or only predicted labels for an input query.
- the trained model parameter weight values.

3.2.3. Adversary Capabilities

The adversary is limited to only manipulating the test set after model training in the evasion scenario. Some further restrictions on the adversary's power need to be enforced, and they include:

- the primary being the attack influence (causative or exploratory).
- which the input feature vectors can be manipulated and their extent.
- application-specific constraints (e.g., correlated features can not be modified independently or the functionality of the original traffic sample should not change)

3.3. Existing Attack Strategies

Now that a brief idea of different attack taxonomies is provided, in this section, few algorithms that have been widely studied for crafting AE on IDSs are discussed. Defining an optimal attack strategy requires the adversary to understand the data distribution and how it must be quantitatively modified to optimize an objective function. In the scope of this work, gradient-based approaches are explored that introduce perturbations optimized for a specific distance metric between the original and perturbed feature sets. Some commonly studied loss functions:

1. L_0 - minimizes the count of perturbed features.
2. L_2 - minimizes the euclidean distance between original and adversarial vector.
3. L_{inf} - minimizes the maximum amount of perturbation δ added to original vector **x**.

The first gradient-based method to craft adversarial samples in computer vision using a constrained Limited-memory Broyden Fletcher Goldfarb Shanno (L-BFGS) algorithm was proposed [9]. The objective function minimized the perturbations δ added to the original input image under the L_2 distance metric. Despite the effective technique, it is not practical due to its high computational requirements for optimal solutions. Since then, several attack strategies have been proposed, and the selection depends on varying factors like performance, computational complexity, and application feasibility.

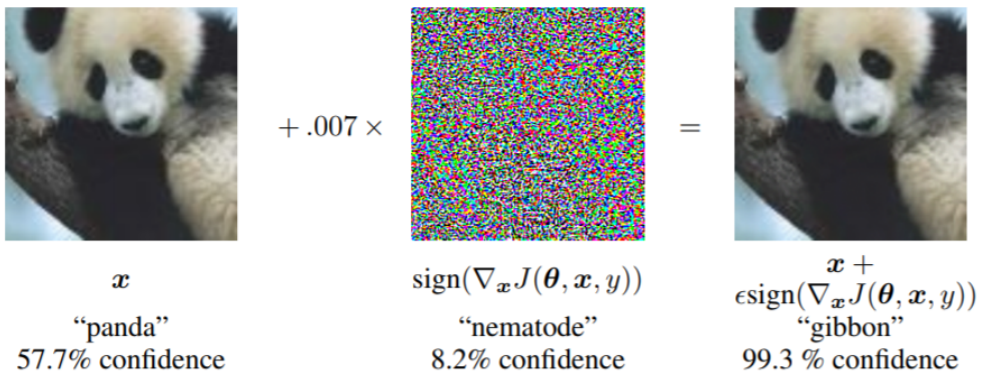


Abbildung 3.3.: Fast Gradient Sign Method [1]

.

- **Fast Gradient Sign Method (FGSM):** When generating an AE, it is crucial to ensure that the perturbation δ does not change the input to the extent that it is perceptible to human factors. This is provided by taking a small step size ϵ iteratively to find an AE, usually a prolonged process. [1] proposed a simple and faster method called FGSM that

minimizes the maximum amount of perturbation δ added to any pixel value. The key idea here is to calculate the gradient ∇ of the loss function concerning the input (via backpropagation) and then add it to the original feature vector along with the sign of its gradient. This allows the FGSM to be most efficient at computing time and faster generation of AEs. The same is formulated in the Equation 3.1.

$$\delta = \epsilon * sign(\nabla_x \mathbf{J}(h_\theta(\mathbf{x}, y))) \qquad \text{s.t.} \quad ||x' - x||_\infty \leq \epsilon \tag{3.1}$$

where:

$\mathbf{x}$	= the original feature vector.
y	= target labels.
δ	= perturbation
h_θ	= the classifier.
ϵ	= step size for δ being added to each feature vector.
J $(\mathrm{h}_\theta(\mathbf{x}, y))$	= the cost function for training the classifier.

Figure 3.3 describes how FGSM is implemented over an image (initially labeled as a panda) by adding noise δ leading to the model misclassifying it as gibbon with a higher confidence score.

Disadvantages: Despite FGSM being widely studied in image domain, they have some clear disadvantages of introducing more perturbations on the entire length of the feature vector. Due to this, its applicability in intrusion scenarios is limited as it does not consider the functionality of the network traffic under its actual label and manipulates all the features.

- **Jacobian-based Saliency Map Attack (JSMA):** is a greedy iterative approach proposed by [2]. The technique aims to minimize the number of features perturbed based on the L_0 distance metric. Saliency maps are computed for an input sample, and the features' saliency value signifies its impact on prediction if changed. This set of values are sorted in reverse order, and top K features are perturbed. The applicability of this algorithm lies in targeted attack scenarios where the Jacobin of the output logits concerning inputs is calculated to understand which features represent a higher variation when added perturbations. Figure 3.4 shows an implementation of JSMA on the MNIST dataset.

  Disadvantages: Despite their advantages of perturbing the least number of features, they assume to have more knowledge about the target system than real scenarios. For this reason, experiments are not conducted on this technique.

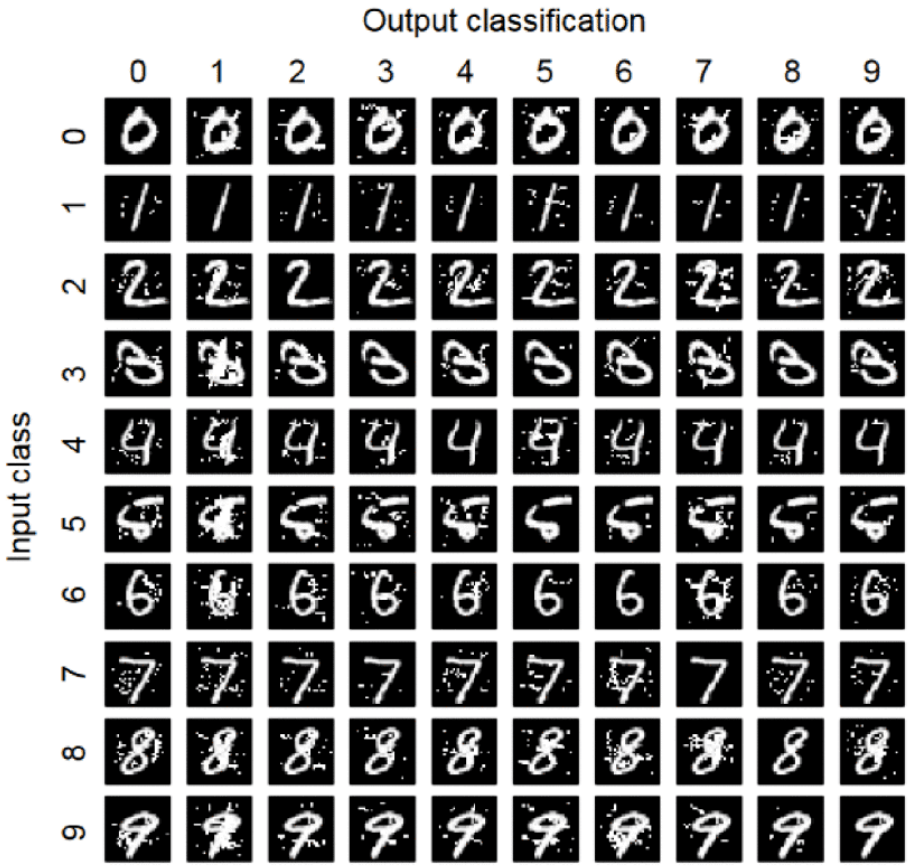


Abbildung 3.4.: JSMA on MNIST. The original samples are along the diagonal, and all other cells contain adversarial examples with the different classes as shown along the axis [2].

- **DeepFool:** initially proposed by [48] computes an AE by minimizing the euclidean distance between perturbed vector $\mathbf{x}'$ and original vector $\mathbf{x}$. This further involves an analytical calculation of a linear decision boundary, discriminating the different classes when a perturbation is added perpendicular to that decision boundary. Since neural networks are non-linear structures, the perturbations δ are added iteratively until the AE is found successfully. The approach crafts samples with fewer perturbations than FGSM and JSMA with higher misclassification rates. However, on the downside, it is computationally expensive. Figure 3.5 provides a conceptual understanding of how these perturbations are added.

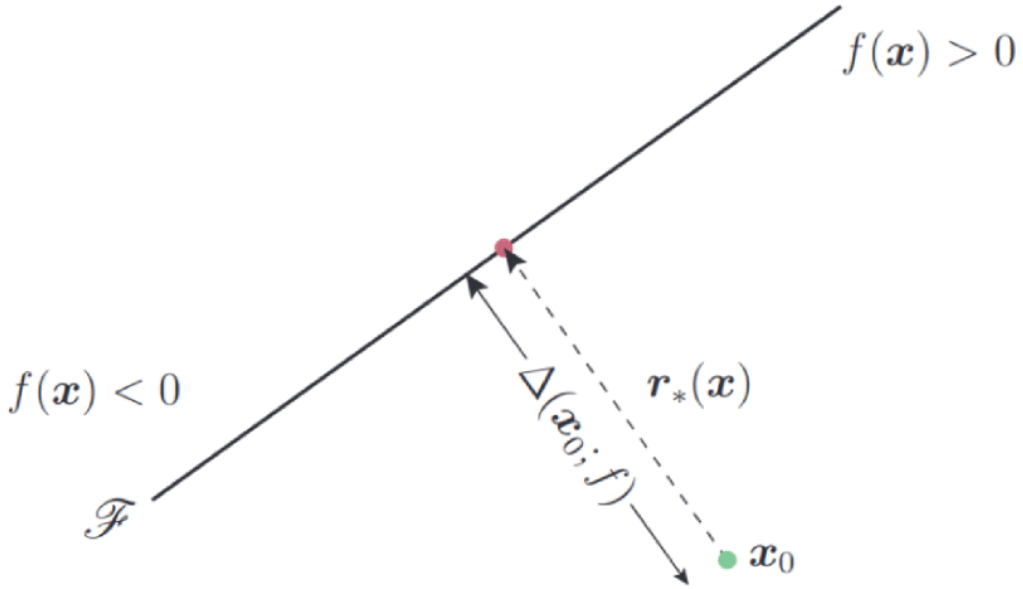


Abbildung 3.5.: Deepfool - perturbation **r** is added to a sample x_0 in the perpendicular direction of the hyperplane separating two class labels [3].

- **LowProFool**: is another approach proposed by [4] that generates an AE on tabular datasets. The evaluation results were compared on four financial datasets against the attacks from state-of-the-art methods while maintaining the lowest perceptibility of the perturbed samples. The key idea of the paper is to emphasize the importance of measuring *imperceptibility* and design attacks that rely on the intuition that only a subset of features are critical for an expert eye to provide a prediction. For example, in a bank loan application *income, age* are important features, and the adversary should minimize manipulations on such important features. However, their formalization is defined for an unconstrained domain where they do not have varying data types, perturbation bounds, and correlations among the features.

3.4. Image versus Tabular Domain

Much of our discussion has been around the machine learning models' brittleness to slight changes in their input. The primary research field studies this issue with an active focus on computer vision-related problems. However, not all applications in the industry work with image datasets; for example, the financial sector relies on the tabular dataset, and similar is the scenario with intrusion datasets where traffic flows are extracted in CSV formats. These AEs exist in all domains; however, the attack strategies are not directly transferable to the intrusion domain. For example, an FGSM manipulates all the features. Still, it does not make sense to do so on a network traffic sample, as one may not want to change the protocol of the User Datagram Pro-

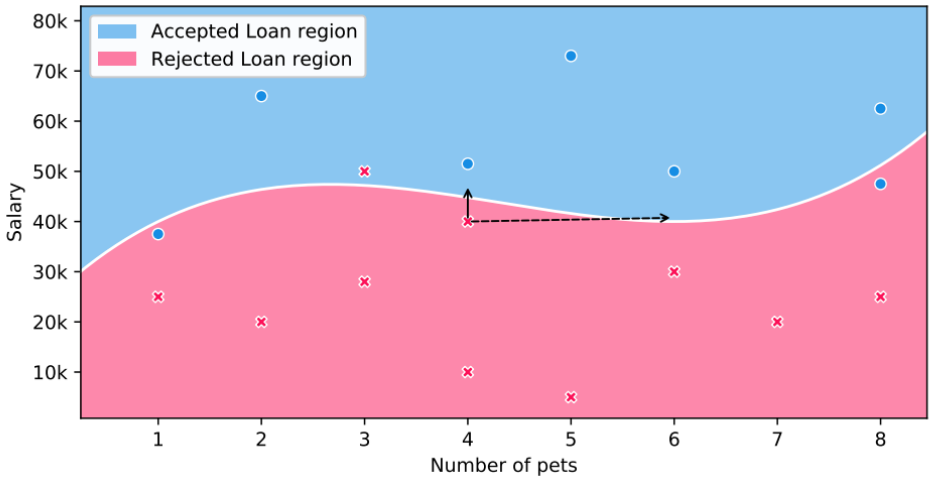


Abbildung 3.6.: LowProFool - The straight arrow is a 'naive' perceptible example, whereas the 'dashed' arrow is an imperceptible perturbation relying on a feature *number of pets*. In both scenarios, the perturbation causes the sample to cross the decision boundary [4].

tocol (UDP) Flood attack to Transmission Control Protocol (TCP) or Internet Control Message Protocol (ICMP) as this would not hold the functionality of the malicious attack.

To the best of our knowledge, [49] was the first to work on testing the effectiveness of adversarial attacks over an IDSs. The author used the NSL-KDD dataset to generate targeted attacks against five different ML models. Their adversarial attack techniques involved FGSM and JSMA. The results match our discussion in Section 3.3 where FGSM modified all the features for each test sample and JSMA a more realistic attack in intrusion detection modified on an average six percent of the features.

[50] also worked on testing adversarial attacks against a multi-layer perceptron using FGSM, JSMA, Deepfool, and C&W. The results effectively decreased the original AUC, with C&W being the least and FGSM the most effective. This paper marked JSMA to be the most realistic attack. [16,19] provides a comprehensive list of the evaluation datasets used in intrusion scenarios under an adversarial setting. Additionally, to understand the whole arms race of attack strategies, interested readers can review a systematic survey [23] to understand all the related work in this domain, providing a concise description of the timeline.

How are images different from intrusion datasets?

Unlike image data dealing with tabular datasets in the adversarial setting is complicated as each feature varies across various values and data types. In addition, the functionality of our perturbed example has to be ensured and make it imperceptible for an IDS model. Some significant differences in both these data structures are summarized as:

1. Image pixel values are a vector with a single data type (float or int). On the contrary, intrusion datasets can have feature types (continuous, discrete, categorical, and binary). This makes the attack strategies like FGSM disturb the input coherence of the test sample.

2. Functionality Constraints: In the case of tabular intrusion datasets, one needs to ensure the adversarial counterpart of an original sample retains its functional nature. This means to validate; further, the manipulations agree with the correlations among the features. On the contrary, images do not have these constraints, and all pixel values are independent.

3. Imperceptibility: Images use l_{inf} norms to make the perturbed images less susceptible and are controlled by a single hyperparameter. However, in the case of intrusion datasets measuring imperceptibility is a complex problem as tabular datasets are less readable and expert knowledge is required, unlike images. Let us discuss this in detail in Section4.6.1.

4. Image have fixed bounds of ([0-1] or [0-255]) for all the features, and intrusion datasets have different upper and lower bounds for each feature. Formally,

$$h_\theta(\mathbf{x}+\mathbf{r}) \neq h_\theta(\mathbf{x}) \qquad \text{s.t.} \quad ||r||_p \leq \epsilon \tag{3.2}$$

   where:

   $\mathbf{x}$ = the original input image.
   δ = perturbation noise.
   $\mathbf{x} + \delta$ = the adversarial counter part.
   δ_p = l_p norm for the vector δ.
   ϵ = upper bound of the norm ball r_p.
   h_θ = the classifier.

   On the contrary, intrusion datasets do not have straightforward formalization (no single upper bound value). It is complicated that their actual class label needs to be considered, i.e., a *SYN Flood* labeled traffic sample would have a much smaller upper bound for feature 'Flow Duration' when compared to a *benign* labeled sample.

3.5. Performance Metrics

Now that the necessary background information on the security evaluation has been discussed let us formalize some core metrics relating to the robustness of IDS models. In this section, the performance metrics are divided into two sub-sections where; the first discusses the widely adopted metric for evaluating ML models under classical design principles, and then some advanced metrics in the adversarial setting.

3.5.1. Metrics for baseline models

There are several primary metrics proposed for a classification model [51], with each having its own trade-offs. Classification is the task of categorizing an input sample to a specific label, i.e., if network traffic is *malicious* or *benign* (can also either be a binary classification or a multi-class problem). Let us define some relevant performance metrics used in the experimental Section 5.

True Positive and True Negative is when a model correctly predicts the positive and negative class instance. On the other hand, False Positive and False Negative corresponds to the model incorrectly classifying an instance to the positive and the negative class, respectively. Confusion matrices are built to describe the overall performance of a classification model, with the diagonal containing all correct predictions and all other cells representing incorrect predictions.

	Malicious	**Normal**
Malicious	TP	FN
Normal	FP	TN

Tabelle 3.1.: Confusion Matrix (rows correspond to actual labels and columns are the model predictions)

where:

TP = True Positive
FP = False Positive
TN = True Negative
FN = False Negative

In addition, some primary metrics:

1. **Accuracy**: also known as *classification rate*, measures how accurately the IDS detects normal or malicious traffic behavior. It is formulated as the sum of correctly predicted instances to all instances.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

2. **Precision**: measure the total correct classification penalized by the number of incorrect classifications.

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

3. **Recall**: measure the total correct classification penalized by the number of missed instances.

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

4. **F1 Score**: In scenarios involving imbalanced data distribution, FP & FN are crucial. F1 score is taken as the harmonic mean of precision and recall.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.6}$$

5. **Area Under Curve (AUC)**: stands for Area under the Receiver Operating Characteristic (ROC) curve. As ROC plots work in a binary classification setting, AUC is used to aggregate the performance across all classification thresholds.

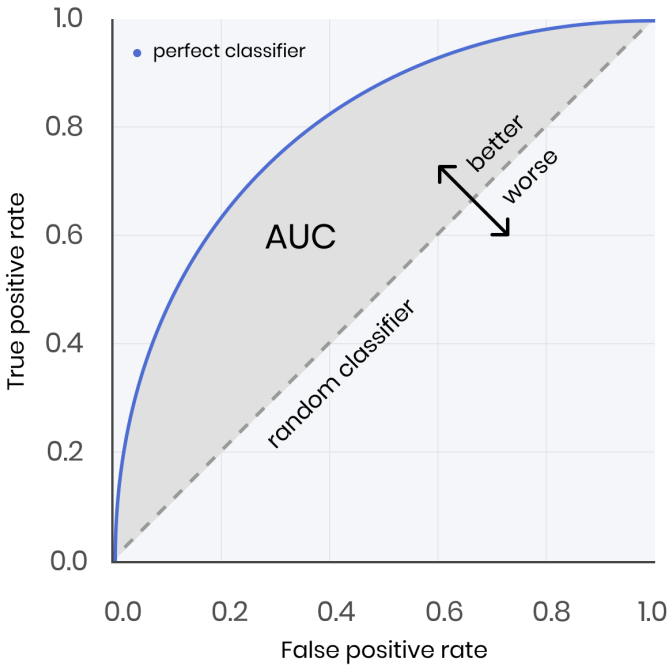


Abbildung 3.7.: Area under the ROC Curve

3.5.2. Metrics for adversarial robustness

1. **Success Rate:** The success rate, also known as *fooling rate* is an important metric to measure the robustness of a trained classifier against an adversarial attack. For the test set $\mathbb{X}$ and the adversarial test set $\mathbb{X}'$. With a given number of total iterations mentioned in Algorithm 1 success rate σ_{max_iters} is defined as:

$$\sigma_{max\_iters} = \frac{|\mathbb{X}'|}{|\mathbb{X}|} \tag{3.7}$$

2. **Perturbation Norm:** Is defined as the measure the l_2 norm of the difference of original vector **x** and adversarial vector **x′** or the perturbation δ. Next, the mean average is computed for all the pairs of **x** and **x′** for all the samples. Formally,

$$||\delta||_2 = \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{x}'_i||_2^2 \tag{3.8}$$

3. **Attack Score:** Equation 3.9 provides a mathematical notation of the metric, where the core idea is to reward the attacks that add perturbations δ, which modifies the prediction confidence to a higher degree. For example, a δ that shifts the original confidence of 52% for *benign* label to 49% for *DoS* label is not very impressive. However, the confidence prediction has shifted from 99% in favor of *benign* to 49% for *DoS* is a stronger attack. The acceptable values of the attack score can be in a range of (0, ∞) for this thesis, and these scores are used to compare adversarial instances.

$$S = \sum_{i=1}^{N} \frac{\Delta C_i}{w_i(f_i - f'_i)^2} \tag{3.9}$$

where

ΔC = difference in prediction confidence of original and adversarial sample.
w_i = weight vector
f_i = original feature vector
f'_i = perturbed feature vector

4. Proposed Architecture

From a methodological perspective, the paper follows a two-step process involving training a base classifier on an intrusion dataset and later querying this model to generate adversarial counterparts for the test set. The generation of these perturbed samples is an iterative process, and the objective is to analyze LowproFool and DeepFool in constrained domains and their suitability in the NIDS domain. In Section 3.5 some metrics were defined to evaluate the performance for different models on the original and adversarial sets. These experiments were repeated for five iterations, and the mean average of these results is reported in the next chapter. As depicted in Figure 4.1, an experimental setup is prepared to answer the questions proposed in section 1.1. The foremost step of the proposed architecture consists of:

1. Selection of three intrusion datasets and its preprocessing submodule.
2. Training five ML models on these clean datasets and provide decent baseline results on their respective test sets.
3. Define a threat model for crafting the adversarial attacks.
4. Adversary leverages two attack algorithms to query the saved ML-based IDS models from Step 2.
5. Prepare an adversarial test set and also perform the transferability evaluations.

As discussed in the Section 3.3 that despite FGSM being faster, it is not considered in the scope of this work because the results produced by this method cannot be used to evaluate intrusion detection problems in constrained domains (detailed in section 4.5). Next, JSMA has been deemed a more suitable method; however, prior work has already studied them [49, 52]. Thus, the focus is on new attack algorithms that focus on the imperceptibility of the adversarial counterparts, namely LowProFool and DeepFool. The details of these techniques are explained in the following sections.

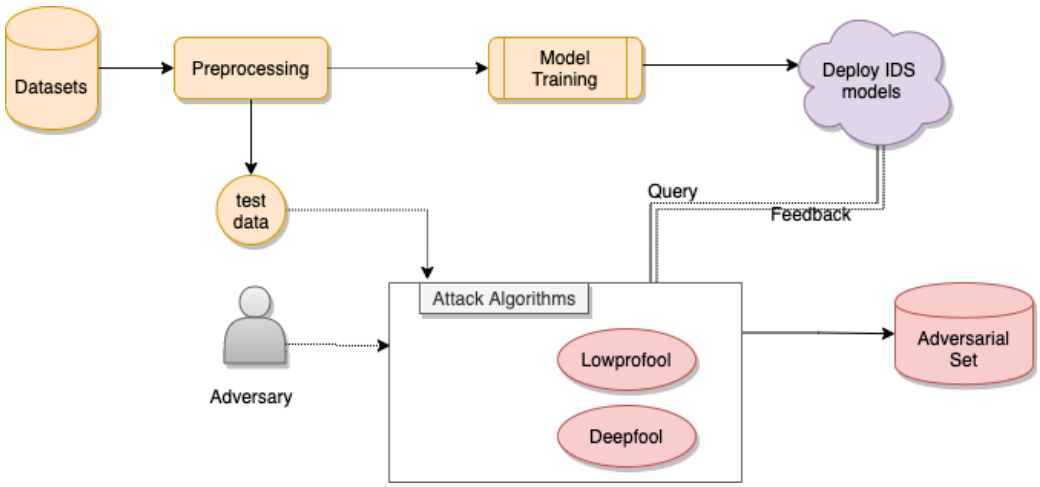


Abbildung 4.1.: A pipeline of the Proposed Architecture

Technologies

The experiments were performed on CPU Intel i5 with 8 GB of RAM in a virtual environment with Python 3.7 as the base interpreter. In order to facilitate the baseline model training (victim models), scikit learn, and python libraries are used like NumPy, scipy, pandas, matplotlib, absl, etc. Unlike prior works crafting the adversarial samples with the mentioned algorithms were developed from scratch using PyTorch 1.8. The usage of the cleverhans library is avoided as non of them define adversarial examples for tabular datasets in a constrained environment.

4.1. Dataset Selection

In the following, let us discuss the importance of a comprehensive and reliable intrusion dataset. The lack of a representative dataset is one of the biggest bottlenecks in designing machine learning-based IDSs [53]. An ideal choice would be to use network packets from the cyber security industry or commercial products, which are seldom available due to security concerns and privacy issues. These statements have been further supported by work from [54, 55].

The research community specifically, Defense Advanced Research Project (DARPA) [56], Knowledge Discovery and Data Mining (KDD), Canadian Institute for Cybersecurity (CIC), Center for Applied Internet Data Analysis (CAIDA), Australian Cyber Security Centre (ACSC), etc., have been extensively working to solve this problem by generating several publicly available intrusion datasets over the last years. As ML-based IDS models are data-driven, synthetic datasets are used today as benchmarks to evaluate the performance of these systems.

How are these synthetic datasets collected?

The testbed architecture comprises multiple host machines connected like on the Internet, and they generate realistic background traffic (based on benign and malicious simulations). These generated network packets are recorded using a flow meter for a specific time. The extracted data is a series of sessions having started and end timestamps, which further closely resembles the real-world data in *Packet Capture (PCAP)s*. It is essential to understand that they are additionally engineered with the help of network analysis tools to perform feature extractions and are mostly saved as raw Comma Separated Files (CSV) files. In the end, these extracted representations in the tabular domain are used to study intrusions in a network system.

To the best of our knowledge, NSL-KDD is the most widely studied dataset in the existing literature and remains a benchmark for evaluating machine learning-based IDSs. However, there has been much discussion regarding their limitations, such as incomplete traffic, duplicate records, and outdated attack types (over 20 years). [57] evaluated several attacks spanning 2007 - 2018, and they list down the shortcomings for existing intrusion datasets. Common demerits include anonymized traffic, removal of payload information (necessary to analyze packet details), lack of attack diversity, etc. Thus, these intrusion scenarios are old, and much advanced IDS models are already robust against such malicious traffics. Evaluating a deployed model against these old categories of the dataset is a rising concern in the industry. Based on these concerns, new DoS/DDoS datasets have been proposed in the last few years by CIC, which remedies the limitations of the previous contributions.

Detection of novel intrusion attacks is a priority for all network operators. Among all different types of existing threats, DDoS is a menace to network security. The hacker aims to exhaust the networked target system with many malicious network traffic requests. DDoS attacks are most common at layers [3-7] of the Open System Interconnection (OSI) model. DDoS Attack Handbook [1] discusses in detail the severity of DDoS and various methods of generating a flood of packets to overwhelm a target machine.

As shown in Figure 4.3, an attacker uses multiple malware-infected devices (computers, routers, or any IoT device) in a distributed fashion, also known as *botnet*, which is further controlled by a few *master nodes*.

[1] https://www.allot.com/docs/ddos-attack-handbook.pdf

To match our results with other baseline work, NSL-KDD is taken into account and to answer the research questions proposed in Section 1.1, an updated dataset based on modern DoS and DDoS attacks has been taken into account under the scope of this work, specifically, CICDDoS [2] and CICIDS [3] from CIC. Let us summarize the nature of different malicious class labels present under these datasets and a detailed data description is provided in Appendix A.

1. **NSL-KDD:** In 1999, KDD cup [4] tools competition was held to collect network traffic records. The main idea was to build an intrusion detector to distinguish *malicious* and *normal* network connection requests. A mass amount of internet traffic was recorded and bundled as KDD'99. Later, NSL-KDD was brought into existence in 2009 (as a cleaned-up version). The dataset contains 43 features per record, where 41 features refer to the input traffic, a score (the severity of traffic input), and a class label. The different class labels in the dataset can be binned in four main categories based on their functionality as described in Table A.2:

Abbildung 4.2.: An attacker executes a DoS attack against a target from a single source.

   a) **DoS:** It is the most common intrusion type where an attacker attempts to shut down the traffic flow altogether to a host machine. As shown in Figure,4.2 the basic idea is to flood the victim with many connection requests, which the target system can not handle and shuts down to protect itself. As a consequence, a legitimate user can no longer utilize the services offered by the target system. [58] provides a detailed taxonomy of the DoS attacks and their implementation details.

   b) **Probe:** is also known as *surveillance*, is an unauthorized probing/scan inside a network with the goal of theft or stealing some critical information about the system. Probing is usually the first step for an intruder, and it checks for the vulnerable ports and gathers other meta-information (e.g., port scanning).

   c) **U2R:** is a privilege escalation scenario where a regular user account attempts to gain access to the target system as a super-user (root).

[2] https://www.unb.ca/cic/datasets/ddos-2019.html
[3] https://www.unb.ca/cic/datasets/ids-2017.html
[4] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

d) **Root to Local (R2L):** it involves unauthorized access to a remote machine. An attacker attempts to "hack"their way into a network to gain local access to the system.

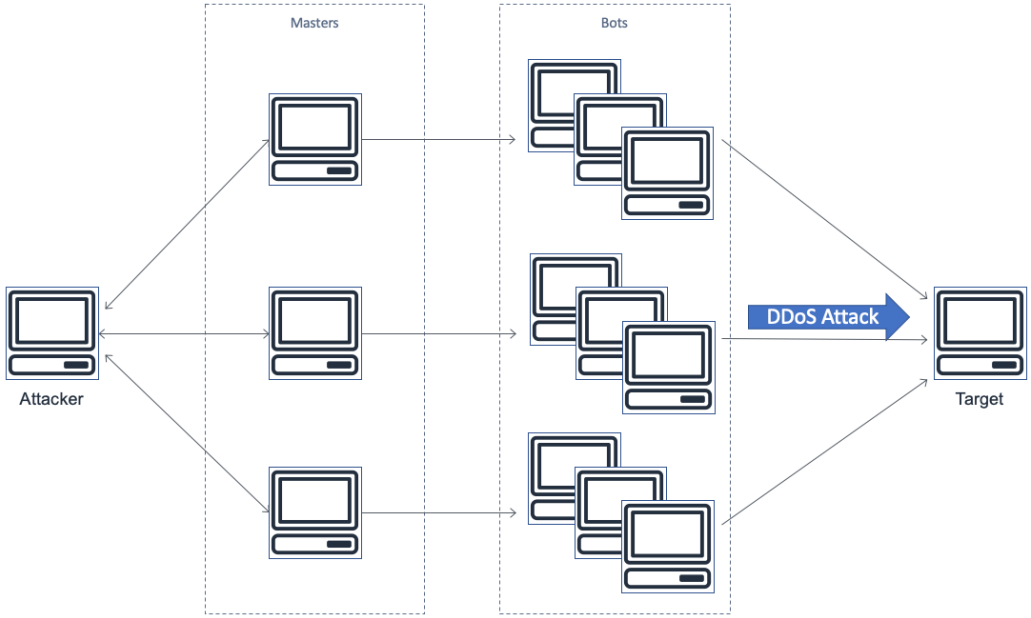


Abbildung 4.3.: Attacker orchestrates DDoS attack against the target with multiple sources. [5]

2. **CICDDoS:** It is developed by CIC in 2019 and consists of a diverse set of updated DDoS attack techniques and scenarios. As per [15] the dataset remedies present shortcomings and is labeled with 80 best feature sets to detect different types of DDoS intrusions. The features are extracted for benign and DDoS flow using a publicly available tool CICFlowMeter [6]. Let us discuss some relevant class labels under the scope of this work, and a detailed data description is provided in the Appendix A.1.

   a) **SYN Flood:** is also known as *neptune* or *half opened* TCP-SYN attack, where an attacker exploits the three-way-handshake TCP protocol required while establishing a connection. Many connection requests (spoofed SYN packets) are directed towards the server, but this client (attacker) never responds with an ACK packet. The process continues until the attacker exhausts the server's connection table memory and utilizes all the resources. The outcome of this process is that the server becomes unavailable to legitimate user requests.

   b) **UDP Flood:** here, the victim server is flooded with small spoofed UDP packets at random ports. Many source IPs are used to spoof the UDP packet, and they do not have any consistent pattern. These attacks never attempt to establish a connection

[5] Source: AWS Best Practices for DDoS Resiliency
[6] https://github.com/ahlashkari/CICFlowMeter

between the server and the client; they are challenging to be detected. This attack aims to utilize all the available bandwidth in the network until the victim goes offline.

c) **Lightweight Directory Access Protocol (LDAP):** attacks involve sending small requests with spoofed source IP address to publically available vulnerable LDAP servers, such that the response is directed to spoofed addresses (intended victims) instead of the actual sender. This is further extended with amplification attacks by selecting queries that yield the maximum responses from internet servers, which floods the spoofed source IP address (victim).

d) **Network Basic Input/Output System (NetBIOS):** the attacker aims to allow applications located on different devices to establish connections and communicate to share resources over a local area network. They can be carried out using either TCP or UDP protocols. The response queries are almost three times the initial requests sent by the attacker.

e) **Microsoft SQL Server Resolution Protocol (MC-SQLR):** these attacks manifest in the form of Microsoft SQL Server responses to a client query or illegal requests through the MC-SQLR listening on UDP port 1434. Precisely, attackers abuse internet available SQL servers with manipulated queries and spoofed source IP. More details on this new attack vector has been studied by Akamai [7].

f) **UDP-Lag:** is a UDP protocol attack that aims to disrupt the established connection between the host and the server. Most of the application has been seen in the online gaming industry, where attackers intend to slow down the movements of other players. The implementations can be carried out by either using software that utilizes the bandwidth of the target network or using a hardware device known as *lag switches*.

4.2. Dataset Preprocessing

The initial steps involved data understanding, cleaning raw datasets, feature selection & engineering, and dealing with class imbalance problems.

Dataset Selection: As discussed in Section 4.1, to the best of our knowledge NSL-KDD and CICIDS have been used in the context of adversarial sample generation. Other datasets have also been used to expose this vulnerability concept. Three multi-class datasets [8] where a *benign* label represents normal traffic flow and all other class labels represent *malicious* traffic flow. The reasons for this selection are:

[7] https://www.akamai.com/fr/fr/multimedia/documents/state-of-the-internet/ms-sql-server-reflection-ddos-mc-sqlr-threat-advisory.pdf

[8] All datasets are used in tabular formats, which represents the network traffic in PCAP formats

- **NSL-KDD**: widely used in almost all the studies relating to ML-based IDS and also in an adversarial setting.
- **CICIDS**: has been studied in constrained scenarios to generate adversarial samples by [52].
- **CICDDoS**: Our aim is to use modern DDoS attack types as they represent complex intrusion scenarios as proposed by [57].

A technical description of features in NSL-KDD is provided in Table, A.3 and for CICIDS & CICDDoS is given in the Table A.1.

Data Cleaning Several preprocessing steps are performed on the raw datasets, and they are commonly summarized as below:

1. All records with null and infinity values are dropped.
2. Rename a few class labels as they were not unique between the train and test set. For example: 'DrDoS_DNS' was replaced with 'DNS'.
3. Dropped records where features $FwdHeaderLength$, $BwdHeaderLength$, and $min\_seg\_size\_forward$ had negative values. These features, by definition, cannot be less than zero and are assumed to be errors during the packet capture process.
4. For NSL-KDD, the attack labels are binned in five main class labels. This is further detailed in the Table A.2.
5. Removed the malicious samples having Port 0 as they are less than 0.01% and more details discussed in Appendix A.1.

4.2.1. Feature Selection

The next step was to perform feature selection to remove misleading data and improve training times. Feature extraction methods like factor analysis, principal component analysis, etc., are not utilized in this pipeline. The assumption keeps the adversarial attacks as realistic as possible, thus maintaining the input coherence and keeping the feature vector undisturbed.

- Let us not drop non-relevant features as these features would bring bias on model training. Most of these features do not hold the payload information and describes source/destination addresses, and they are also dropped seldom available due to security concerns. Specifically:
  - NSL-KDD: $difficulty, label$ (different from class label)
  - CICIDS & CICDDoS: $Unnamed: 0, FlowID, SourceIP, DestinationIP, Timestamp, SimillarHTTP, DestinationPort$.

- Next, all features with a single unique value throughout the dataset do not need to be used. This does not add any additional variance to the dataset and would increase the computational cost. For example:
  - NSL-KDD: $num\_outbound\_cmds$
  - CICIDS & CICDDoS: $BwdPSHFlags$, $FwdURGFlags$, $BwdURGFlags$, $FwdAvgBytes/Bulk$, $FwdAvgPackets/Bulk$, $FwdAvgBulkRate$, $BwdAvgBytes/Bulk$, $BwdAvgPackets/Bulk$, $BwdAvgBulkRate$, $FINFlagCount$, $PSHFlagCount$, $ECEFlagCount$.
- Finally, all secondary features are dropped which are direct formulations of other primary features. The details are provided in Appendix A.1.
  - CICIDS & CICDDoS: $FwdHeaderLength.1$, $FwdPacketLengthMean$, $BwdPacketLengthMean$, $FlowBytes/s$, $FlowPackets/s$, $FwdPackets/s$, $BwdPackets/s$, $MinPacketLength$, $MaxPacketLength$, $AvgFwdSegmentSize$, $AvgBwdSegmentSize$.

4.2.2. Sampling

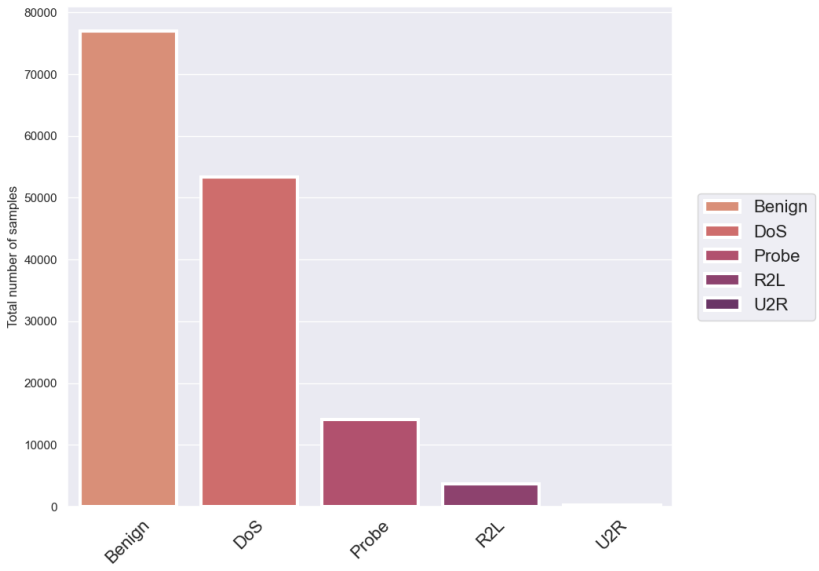


Abbildung 4.4.: NSL-KDD: Data Distribution

All the datasets suffer from a high-class imbalance problem, as shown in Figure 4.4 and A.1. In NSL-KDD and CICIDS, one may observe a very high ratio of Benign samples; however, this is our most minor concern as the focus is on deceiving a trained IDS model for malicious samples

to look benign. Thus, the better model is trained with benign samples, and the stricter is our evaluation scheme.

Statistical Evaluation: The dataset is provided as separate train and test subsets, thus no changes are done to this ratio and evaluate its actual counts as provided in tables [4.1 - 4.3]. The training data is used to train the machine learning model, while the testing data is used to evaluate the model's classification performance. Furthermore, for hyperparameter tuning, the training data is split into subsets of training and validation sets in the ratio of 80% and 20%, respectively.

	# samples
train size	125974
test size	22544
Benign	77053
DoS	53385
Probe	14077
R2L	3749
U2R	252

Tabelle 4.1.: Sample frequency for different labels for NSL-KDD

	# samples
train set	2264594
test set	566148
Benign	2273097
DoS	252661
DDoS	128027
Portscan	158930
Patator	13835
Web Attack	2180
Bot	1966
Others	47

Tabelle 4.2.: Sample frequency for different labels for CICIDS

However, for CICDDoS, the number of benign traffic samples was limited ($\tilde{1}$12k), and malicious samples contribute a vast number. It is vital to perform undersampling for this specific dataset as the trained ML model should not be biased towards intrusion samples due to their higher ratio. All the available *benign* samples are considered and randomly sampled the same number of other class labels. An exception to this would be the *UDPLag* label, which comprises fewer records. As shown in Table 4.3 seven different class labels are selected, which are commonly provided in the training and test files by the authors [57].

	# samples
train set	619348
test set	154837
Benign	112312
Syn	120000
UDP	120000
LDAP	120000
MSSQL	120000
NetBIOS	120000
UDPLag	61873

Tabelle 4.3.: Sample frequency for different labels for CICDDoS

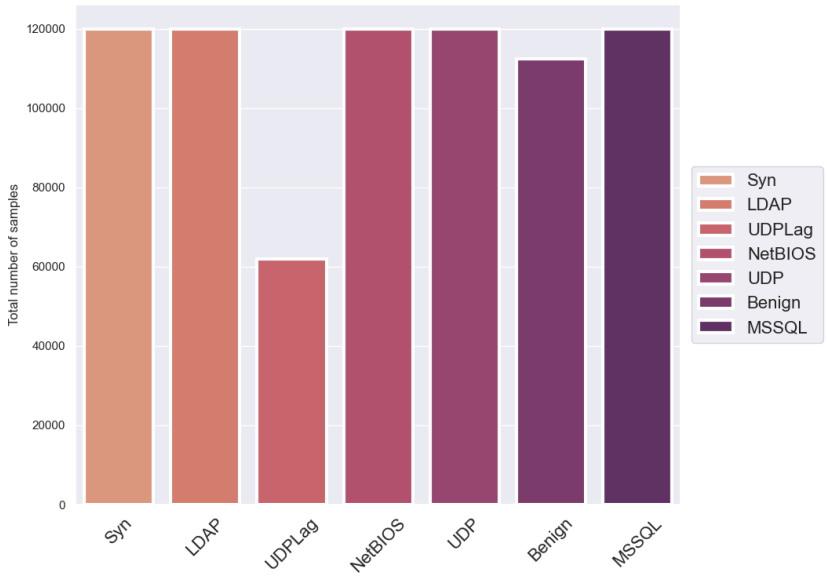


Abbildung 4.5.: CICDDoS2019: Data Distribution

Figure 4.5 shows the uniform sampled distribution from the whole dataset for CICDDoS2019 considered for the experiments performed in the scope of this work.

4.2.3. Categorical Encoding

Next, to deal with categorical features, one needs to encode them as numbers for the ML models to process them. There are several ways, like one-hot encoding, label encoding, or train categorical embedding representations [9]. For simplicity reasons, one-hot and label encoding are considered. When considering the one-hot representations, the input feature space would expand by a large number. For example, in NSL-KDD, 122 features are obtained after one-hot encoding, which gets complicated during adversarial attacks. Moreover, it did not contribute to better performance while evaluating the baseline models over test data. Label encoding is opted, iterating on all categorical features during the preprocessing pipeline and save the encoded values as a JSON object in the utils directory. This utility file is later used during model training and also while crafting adversarial examples.

Assumption: The process is inherent to an IDS after deployment for the reason that the adversary now does need to be informed of this encoded dictionary, and its knowledge is kept to the bare minimum.

4.2.4. Normalization

In the last step, all the datasets are scaled. Applying a normalization scheme (Min-Max or Standard) to all the features between a specific interval ([0-1] or [-1, 1]) was a critical step to improve the training process and thus the evaluation of the test set. All the features have huge range differences, preventing the classifiers from converging, as the features with higher values are preferred over the binary features. Min-Max-Scalar is used from the scikit-learn library to overcome this scenario and saved the scalar object as a pickle. Selecting Min-Max over standard scaler is motivated by better performance based on evaluation metrics discussed in the section 3.5and the training time of these base models. Equation 4.1 formally describes how the min-max normalization was performed.

$$\mathbf{x}' = \frac{\mathbf{x} - \mathbf{x}_{min}}{\mathbf{x}_{max} - \mathbf{x}_{min}} \tag{4.1}$$

[9] https://github.com/Shivanandroy/CategoricalEmbedder

4.3. Baseline Models

Five machine learning-based models are trained for the experimental phase, specifically random forest, naive Bayes, linear support vector classifier, gradient boost classifier, and deep neural network. This choice was selected to have at least one for each tribe of algorithms, as mentioned in the above sections. Furthermore, to keep things simple, an exception of not using unsupervised and evolutionary algorithms is not to be implemented for classification tasks and can be a potential future work. To optimize the hyper-parameter of the algorithms, a cross-validation grid search was performed, using three folds mainly to optimize the performance of these IDS models. The optimized settings are provided in Table 4.4.

Models	**Configuration**
Random Forest	max_depth = 15 $\|min\_samples\_split = 5\|n\_estimators = 16$
Linear SVC	tol = 0.01 $\|c = 3\|max\_iter = 2000$
Bernoulli Naive Bayes	default
Gradient Boost	n_estimators = 50 $\|lr = 0.1\|max\_depth = 1$
Neural Network	batch_size = 256 $\|lr = 0.001\|n\_epochs = 20$

Tabelle 4.4.: Experimental Settings

As per the evaluation results shown in Table 5.1, the Neural network and random forest outperformed other models on all the metrics. As most research on adversarial input attacks is performed on Deep Neural Network (DNN)s, the attack implementations in this work are designed for the neural network.

Note: In the experimental section, these trained models are considered as the victim models, and a neural network is specifically used to generate adversarial examples in a white-box setting.

4.4. Attack Surface

In the following sub-sections, let us define the decomposition of our threat model in three main dimensions: adversary's goals, knowledge, and capabilities.

Goals: Different aspects are considered in the adversary's goals under the scope of this work.

(g.i) Integrity violation by maximizing the number of malicious test samples and normal traffic samples.

(g.ii) Availability violation by maximizing the number of normal traffic flow samples flagged as DoS label.

(g.iii) Targeted attack scenario where all *malicious* samples are targeted to *benign* label and all *benign* are targeted to *DoS* label, subject to functionality constraints (Section 4.5.2).

Knowledge: For an IDS, it is worth keeping these assumptions realistic and to the minimum. First experiments were conducted with complete knowledge of the system and slowly reduced it in an iterative manner until excellent performance was received. All the information except the ones below is kept entirely secret and was not available to the adversary while crafting successful adversarial samples.

(k.i) training data is kept hidden from the adversary under all circumstances.

(k.ii) due to the access to test data, one may assume to know the feature preprocessing and engineering required to pipe an input batch across the model for the classification task. This ensures to at least query the victim model.

(k.iii to craft adversarial samples, a gradient-based approach has been used, so the learning algorithm is kept known. However, the model parameters are kept unknown. When performing these attacks on the shallow models (random forest, naive-Bayes, etc.), the adversary only passes the adversarial set, and the type of learning algorithm does not matter.

(k.iv) in feedback, the adversary is informed about the output label, and the confidence value is not shared.

(k.v) for lowprofool, the adversary builds a feature importance vector from the test set in the following two ways. Selections were made based on performance, as mentioned in the experimental section 4.6.1.

  (a) train a random forest and save the vector as a JSON object.

  (b) uses Pearson's correlation coefficient formula to build a co-relation vector.

Capability: Focus is on maintaining the input coherence of the benign and malicious samples during the process. These constraints account for the definition of an optimal attack strategy where the original samples are perturbed only in an allowable set of perturbation space relating to its original class label.

(c.i) exploratory or evasion scenario only.

(c.ii) to maintain the input coherence, the feature manipulation are restricted based on following heuristics:

  (a) do not perturb the functional features;

  (b) do not perturb the binary features; e.g., flag counts

(c.iii) each sample is perturbed in small step sizes and the bounds are decided based on the values observed in the test data. These bounds are saved for each class label as a JSON object, ensuring the original class's proper functionality.

[10]Source from [2]

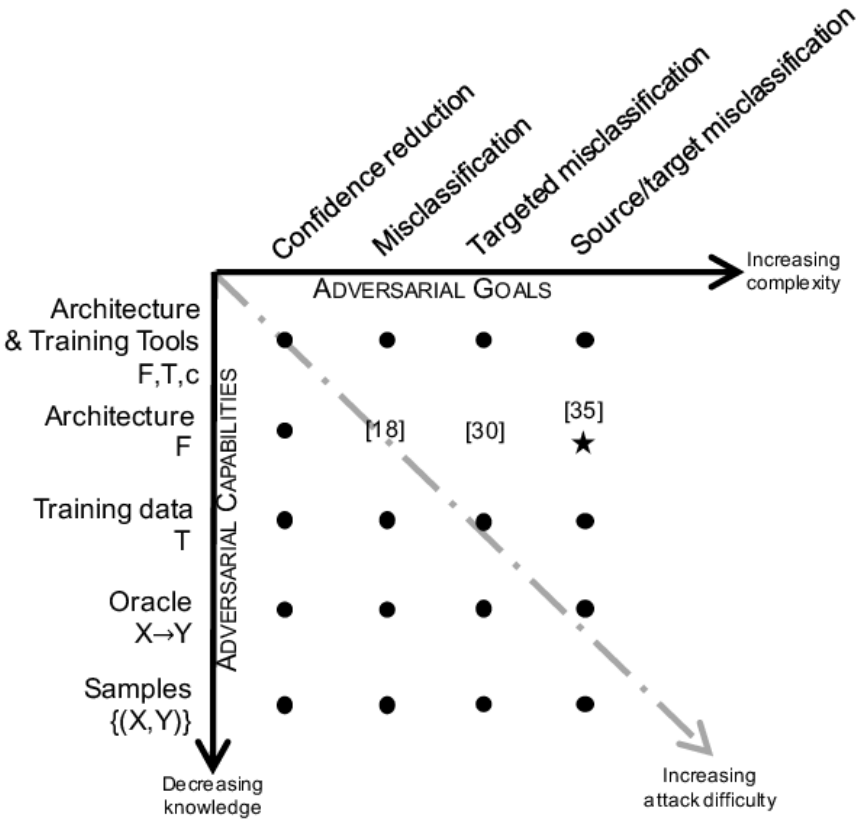


Abbildung 4.6.: Threat Model Taxonomy [10]

(c.iv) discrete and nominal features are perturbed, and their multiplicity is considered a constraint.

(c.v) No operation is done to maintain the correlation features, as no direct formulation is available.

(c.vi) The maximum number of attempts to manipulate the input data is controlled with the help of hyper-parameter *max_iters*.

The Table 4.5 summarizes the defined threat model based on the points mentioned earlier.

4.5. Constrained Domains

In this subsection, the importance of considering constraints when generating adversarial samples in intrusion scenarios are mentioned. Most of the previous work in evasion attacks assumes the adversary can arbitrarily manipulate all the features. This is not true in some domains where

Goals	Influence	Exploratory (evasion) scenarios
	Security Violation	Integrity & Availability
	Specificity	source/target misclassifcation
Knowledge	Training data	No access (makes attacks more realistic)
	Feature Set	Yes (as the synthetic datasets are publicly available)
	Learning algorithm	Yes (neural network)
	Model Parameters	No access
	Model Feedback	Yes (only predicted labels)
Capability	Dataset	limited to test set only
	Input coherence	feature perturbations with constraints
	Perturbation space	each feature has different bounds
	Constraints	multiplicity and functionality constraints
	No. of query requests	controlled by max_iters

Tabelle 4.5.: Crosstable categorization for our Threat Model

stricter restrictions are imposed with a limited degree of freedom for perturbing. Assuming a NIDS classifies the feature representation [11] of real network traffic flows as *usual or malicious*. When tweaking these feature vectors, the adversary must be careful and obey the TCP/IP protocol so that the network traffic sample manipulation is realizable. For example, it would be illegal to assign negative integers to port numbers.

Most research in adversarial machine learning deals with unconstrained scenarios, and those methods are not directly applicable in NIDS. In constrained domains, all the features do not represent the same kind of information (unlike pixel vs. pixel information), nor do they have a common data type. These fundamental reasons require us to re-define the threat surface for each application scenario and make the existing attack algorithms unsuitable.

1. Unconstrained domains have algorithms that consider minimizing the human perception of the perturbation. Such metrics do not have any meaning on tabular datasets, as humans can not readily perceive them. Thus the approaches have limited utility in constrained scenarios.

2. Adversary assumes complete control over the feature space. This is likely an unreasonable assumption; for example, in NIDS, features may represent broad network behaviors that happen to be outside the adversary's control, such as round-trip times, backward packet size, etc.

[11] These features are created using a feature extraction algorithm, taking network traffic flows as their inputs.

3. Domain limitations always apply when considering the functionality of the traffic sample. For example, the NSL-KDD dataset has protocols and service flag features. To maintain its legitimacy as a network traffic sample, a specific set of service flags can belong only to a single protocol ID.

What constraints should be learned for intrusion datasets?

To capture all the functionality constraints on intrusion datasets would require an expert's opinion to follow a rule-based approach. The adversary should follow this handcrafted set of rules while crafting perturbations. Moreover, this does not guarantee that all the constraints observed in the tabular represented format of the actual traffic are captured. To the best of our knowledge, it [52] is the only work that has considered these constraints in an adversarial setting. However, their implementations are limited to considering primary and secondary features only for NSL-KDD and UNSWB-15 datasets. The intuition behind their methodology is to model these constraints as first-order-logics and is limited to deal with only categorical features.

In the scope of this work, let us scale up by considering newer datasets like CICIDS and CICDDoS, where most features are continuous. Thus, the functionality constraints needed to be captured dynamically by creating bins and computing each feature's upper and lower bounds for each class label. This information is saved as a JSON object before crafting attacks. Since this information is seldom available, it was decided to infer it from directly the dataset available and further categorized them into:

1. Primary features: Independent features which can be manipulated independently of other features.

2. Secondary features: They can be categorized as natural formulations of two other primary features or having a significant correlation value with any other feature in the dataset.

4.5.1. Feature Bounds

After the pre-processing data module, CIC datasets and NSL-KDD has 58 and 40 features, respectively, in mixed data types, including binary, categorical, ordinal, nominal, and continuous. Unlike in images where perturbations are bounded between (0, 1) or (0, 255), the adversary faces a challenging problem on these datasets. It is crucial to keep each feature's upper and lower bound limits along with its data type. For example, SYNFlagCount can only have ordinal values and not decimals.

Since these approaches are gradient-based, a derivative is calculated for the loss function with respect to the input feature vector. Perturbation space is created for each dataset while creating our threat model. During the attack process for each iteration, the adversary ensures to pass the perturbed data batch through a clipping module that bounds the samples to their perturbation bounds based on its actual label.

4.5.2. Input Coherence

A different scheme to learn these constraints was required and is accomplished in the following steps:

- As mentioned in Appendix,A.1 re-compute the secondary features iteratively if they happen to be formulations of other primary features.
- Do not perturb the categorical and binary features flags, as, during our experimental phase, it was observed that continuous features were sufficient to expose the vulnerabilities.
- Create a multiplicity vector which holds the information about each feature being (discrete, continuous, or binary) while iterating; ensure to hold this constraint before getting the prediction from the model.

The steps mentioned above prove simpler and robust to all datasets and do not require any extensive expressions. However, there might be constraints due to correlation values, and that stays one of the future work to be handled with constraint resolution problems.

Another aspect of these attacks involves input coherence. Prior work involved using extensive expressions (i.e., propositions) as a constraint while adding perturbations that ignored the multiplicity factor, re-computation of secondary features based on primary features, and the correlation among features. However, handling correlation stays one of the complex tasks; for example, FwdIATStd and FwdIATMean are correlated, and handling such scenarios stays one of the future works of this work. If the adversarial counterparts do not satisfy these constraints, they must be bounded and rounded to fit the context and maintain their proper functionality

based on their original label. For instance: an SYN label representing a DoS attack cannot have src_bytes representing the amount of data transferred from source to destination greater than zero. The connection is never established in an SYN attack label.

4.6. Attack Algorithms

Multiple tests for two attack algorithms, LowProfool and DeepFool, were performed. Additionally, these methods are extended from their base implementations to fit in constrained domains, and thus the naming conventions as *C-LowProFool* and *C-DeepFool* are used. In C-LowProFool, a feature importance vector (generated from a random forest model based on test data) is used to guarantee the concept of imperceptibility. Similarly, in DeepFool, few steps are modified for faster convergence. The details are discussed in the following sections:

4.6.1. C-LowProFool

The exact properties of an AE are a topic of discussion in the research community. When dealing with images, measuring the perturbation is natural for a human to check if the victim model was deceived based on the input and predicted label. Measuring this perturbation is a complex task. [4] provides a formalization for measuring *imperceptibility* of an attack under the most commonly used l_p norms in a tabular domain. This work is extended for intrusion datasets where functionality constraints are present (Section 4.5).

Key Idea: In intrusion datasets, an expert is likely to focus on the critical features (total connection requests, failed attempts, number of half-open connections) among the whole features space to predict if a traffic sample is malicious or benign. Assuming our ML-based IDS models are the expert, the aim is to bypass this by perturbing the less critical features (UrgentFlag, BwdIATMean, etc..) and avoiding the important ones. Adapting to the current work, let us now translate the formalization for perceptibility of an adversarial attack as the l_p norm of perturbation weighted by a feature importance vector.

Notation: For the test set **X** let us denote each sample as $\mathbf{x}^{(i)}$ where $i \in \{1 - N\}$ and their true labels as $y^{(i)}$. For the set **X** there exists a set of features $j \in \mathbf{J} = \{1 - D\}$. Next, a multiclass function $h_\theta : \mathbb{R}^D \rightarrow$ *l* mapping inputs to a discrete label *l* $\in \{0, L\}$ and $d : \mathbb{R}^D \rightarrow \{0, 1\}$ a mapping of the perturbation vector $\mathbf{r} \in \mathbb{R}^D$ to its perceptibility value. As per (Section 4.5) a set of valid coherent samples is defined as $\mathcal{C} \subseteq \mathbb{R}^D$.

$N =$ total number of test samples
$D =$ total number of features
$L =$ total number of class labels

For a given sample $\mathbf{x} \in \mathbb{R}^D$ with a true label $s = h_\theta(x)$ it is required to generate its adversarial counterpart with a target label $t = h_\theta(x + r)$ such that $\mathbf{r} \in \mathbb{R}^D$.

$$\mathbf{r} =_r d(r) \qquad \text{s.t. } h_\theta(x) \neq h_\theta(x + r) \text{ and } (x + r) \in \mathcal{C}. \tag{4.2}$$

Additionally, each feature j is associated with a feature importance vector $\mathbf{v}$ such that each coeffieciet $v_j \in \mathbb{R}^D$. Formally,

$$\mathbf{v} = \{v_1, ..., v_j, ..., v_D\} \qquad \text{s.t. } v_j > 0, \ \forall j \in \mathcal{J} \tag{4.3}$$

Equation 4.2 ensures the input coherence of the generated adversarial sample. Finally, to measure the perceptibility as the l_p norm of the element-wise product of perturbation vector $\mathbf{r}$ and feature importance vector $\mathbf{v}$.

$$d_v(r) = ||\mathbf{r} \bullet \mathbf{v}||_p^2 \qquad \text{where } \bullet \text{ is the Hadamard product} \tag{4.4}$$

Minimization Problem: The objective is to perform higher perturbations on more minor relevant features, and this can be formulated as the aggregation of the above equations with the final utility loss function and the minimization of d_v .

$$g(r) = \mathcal{L}(\mathbf{x} + \mathbf{r}, t) + \lambda * d_v(r) \tag{4.5}$$

To provide an intuition of the equation 4.5 $\mathcal{L}(\mathbf{x}', t)$ is the cross-entropy loss of feature representation concerning the target label; thus, the input feature space is adjusted with the derivative of this loss. The second term $||\mathbf{r} \bullet \mathbf{v}||_p$ allows to minimize the perturbation $\mathbf{r}$ w.r.t to the perceptibility. The overall minimization problem encapsulates the whole algorithm in a hybrid loss function. Additionally, λ is used as a hyperparameter to control the penalty of using the important features to achieve the adversarial goal. The algorithm 1is a modified version of the *low profile* algorithm in the constrained setting for multi-class models. It is a gradient-based iterative approach where the least perceptible example is selected as the final adversarial representation.

Algorithm 1: C-LowProFool

Input: Classifier h_θ, test data $\mathbb{X}$, target label t, loss function $\mathcal{L}$, feature importance vector $\mathbf{v}$, mask vector $\mathbf{m}$, perturbation *bounds*, hyper-parameters: trade-off factor λ, step size α, *max_iters*.

Output: Adversarial data $\mathbb{X}'$

1 $\mathbf{r} \leftarrow$ random vector of length D. `// values in a range of 0 - 0.0001`

2 $\mathbf{x}' \leftarrow \mathbf{x} + (\mathbf{r} * \mathbf{m})$

3 **for** *i in range [0-max_iters]* **do**

4 $\quad logits = h_\theta(x')$

5 $\quad \hat{y} = \mathcal{L}(logits, t)$ `// Cross entropy loss`

6 $\quad d_v(r) = imperceptibility(\mathbf{v}, \mathbf{r})$

7 $\quad \mathbf{r}_i \leftarrow -\vec{\nabla}_r(\mathcal{L}(\mathbf{x}', t) + \lambda * d_v(r)$

8 $\quad \mathbf{r} \leftarrow \mathbf{r} + \alpha * \mathbf{r}_i$

9 $\quad \mathbf{x}'_{i+1} \leftarrow \mathbf{x} + (\mathbf{r} * \mathbf{m})$

10 $\quad \mathbf{x}'_{i+1} \leftarrow clip\_tensor(\mathbf{x}'_{i+1}, bounds, true\_label)$

$\quad$ `// ensures functionality`

$\quad$ `// The coherence checks involves multiple constraints and for non-trivial reasons ommitted .`

11 $\quad \mathbf{x}'_{i+1} \leftarrow coherence(\mathbf{x}'_{i+1}, h_\theta, *) \qquad s.t.\ \mathbf{x}' \in \mathcal{C}$

$\quad$ `// takes input some additional utility objects`

12 $\mathbf{x}' \leftarrow_{x_i} d_v(\mathbf{x}'_i) \qquad \forall i \in [0, ..., N] \quad s.t.\ h_\theta(\mathbf{x}'_i) \neq h_\theta(\mathbf{x})$

13 **return** $\mathbf{x}'$

4.6.2. C-DeepFool

[33] proposed a white-box approach for crafting adversarial samples against neural networks. The key idea is based on an iterative linearization of the victim model. It generates perturbations such that the distance between the decision boundary and a given input vector is minimal. To overcome the non-linearity in high dimensional space, they leveraged linear approximations iteratively. The results from the image domain show them to produce fewer perturbations than JSMA, and thus this approach is extended in constrained scenarios.

Note that imperceptibility for this approach is not considered, and the goal is to perform a source/target misclassification of test samples. The same notation is followed as in Section 4.6.1, and for a complete formal description of this algorithm, one may refer to the original paper [33]. The extended version for this methodology is provided in the Algorithm 2 (line 9-11).

Algorithm 2: C-DeepFool

Input: Classifier h_θ, test data $\mathbb{X}$, original label *o*, target label *t*, feature importance vector **v**, mask vector **m**, perturbation *bounds*, hyper-parameters: overshoot λ, step size α, *max_iters*.

Output: Adversarial data $\mathbb{X}'$

$\mathbf{x}_0 \leftarrow$ **x**, **i** $\leftarrow$ **0.**

while $h_\theta(x_i) = h_\theta(x_0)$ **do**

  for $t \neq h_\theta(x_0)$ **do**

    $w' \leftarrow \vec{\nabla}_t h_\theta(x_i) - \vec{\nabla}_o h_\theta(x_i)$ `// target gradient - source gradient`

    $\vec{\nabla}_{net\_grad} \leftarrow h_\theta(x_i)[o] - h_\theta(x_i)[t]$ `// difference in output latent space`

  $\delta \leftarrow_{t \neq o} \frac{\vec{\nabla}_{net\_grad}}{||w'||_2}$

  $\delta \leftarrow \delta * \alpha$

  $\delta \leftarrow \frac{\delta}{\boldsymbol{v}+1e-6}$

  $x_{i+1} \leftarrow x_i + (((1+\lambda) * \delta) * \boldsymbol{m})$

  $\boldsymbol{x}_{i+1} \leftarrow clip\_tensor(\boldsymbol{x}_{i+1}, bounds, \mathsf{o})$

  `// ensures functionality`

  `// The coherence checks involves multiple constraints and for non-trivial reasons ommitted .`

  $\boldsymbol{x}_{i+1} \leftarrow coherence(\boldsymbol{x}_{i+1}, h_\theta, *) \qquad s.t. \ \boldsymbol{x}_{i+1} \in \mathcal{C}$

  `// takes input some additional utility objects`

  $i \leftarrow i + 1$

return $\boldsymbol{x}_{i+1}$

5. Evaluation

In this chapter, our experimental results are discussed in detail on all three datasets. Begin with the performance of the baseline models whose input is the clean data. The frequency of samples belonging to each label is summarized in Table 4.3. Next, the attack surface for the adversary is built, and the performance of two algorithms is investigated as mentioned in Section 4.6. Our experiments validate the hypothesis if adversaries can deteriorate the performance of classifiers in constrained domains.

Note: Our aim is not to achieve a 100% success rate in these experiments but to find a trade-off between the computational difficulty to find such samples and their imperceptibility. One can execute these methods for a more significant number of epochs on powerful GPUs to further find more potent adversarial samples. However, the results presented are sufficient enough to expose the vulnerabilities of ML-based IDS. In the end, an ablation study is provided regarding which features were manipulated and the transferability of the adversarial set on other models. Five experiments are conducted, and the mean results are reported for both methodologies on all datasets.

5.1. Baseline Performance

Several different classifiers were trained and tested using all three datasets to set the baseline for the performance of ML-based IDS models. The results on the test sets are presented in Table 5.1. The best values are represented in bold and second best with an underline. Based on the observations from the results, Random Forest and Neural Networks outperform the SVC, Naive Bayes, and Gradient Boost classifier. This essentially also means that they perform better to classify the labels for discriminating network traffic samples.

These baseline models are saved and later used to check out the hypothesis of the transferability of AE. We can see that Naive Bayes' do not provide good results; however, in order to represent one algorithm from each tribe of ML algorithm as explained in Section 2.2

[1] Gradient Boost Classifier

	Random Forest	SVC	Naive Bayes	GB [1]	Neural Net
	NSL-KDD				
Accuracy	0.763	0.748	0.611	0.717	**0.772**
Precision	**0.82**	0.747	0.732	0.748	0.717
Recall	**0.763**	0.748	0.611	0.717	0.75
F-Score	**0.719**	0.7	0.586	0.668	0.7
AU-ROC	**0.942**	0.887	0.921	0.892	NA
Time (in seconds)	2.8	15.7	0.3	28.6	77.5
	CICIDS				
Accuracy	**0.995**	0.939	0.663	0.959	0.977
Precision	**0.995**	0.941	0.901	0.954	0.974
Recall	**0.994**	0.939	0.663	0.959	0.972
F-Score	**0.995**	0.935	0.736	0.953	0.972
AU-ROC	**1**	0.979	0.886	0.989	NA
Time (in seconds)	240	1364	9.7	3706	1708
	CICDDoS				
Accuracy	0.934	0.884	0.437	0.824	**0.972**
Precision	0.948	0.916	0.385	0.969	**0.982**
Recall	0.934	0.884	0.437	0.824	**0.972**
F-Score	0.933	0.883	0.366	0.824	**0.976**
AU-ROC	0.998	0.987	0.77	0.997	NA
Time (in seconds)	21.5	213	3.3	300	353

Tabelle 5.1.: ML-based IDS model performance on intrusion datasets

5.2. Adversarial Attacks

The objective of the evasion attacks is two-fold. First, misclassify all *benign* samples to *malicious* samples and vice versa. Both these methods are implemented to generate adversarial sets from the original test set. The pre-trained neural network model was used to query the target model and utilize the predicted outputs at each iteration.

5.2.1. Hyperparameters

The core idea of LowProFool is to maintain the imperceptibility of the perturbed samples, and DeepFool generates faster and more potent attacks. Some hyper-parameters used to guarantee the convergence of multiple loss functions are shown in Table 5.2.

1. **batch_size:** It is observed that small batch sizes give to higher success rates; i.e., if one finds AEs one at a time, then we would achieve the best results. However, let us consider batch processing for faster computation of these experiments.

2. **cor_weights:** is a boolean parameter that, if set to *True*, consider the Pearson's correlation coefficient as the feature importance vector. A *False* value uses the feature importance vector generated from a random forest on the test set only.

3. **perturb_threshold:** the idea is not to manipulate the features which are significant for making a prediction. A threshold is set, i.e., less than x% significant features should be manipulated.

4. **max_iters:** The experiments are iterated for a fixed number of iterations as a trade-off between the perceptibility and computational time. However, one can experiment with higher significant numbers to generate stronger adversarial samples.

5. **alpha α:** this decides the step size for updating the original vector with adversarial noise.

6. **lambda λ:** this is a regularizing hyper-parameter for controlling the effect of each loss component in the final hybrid cost function. A higher value of λ makes the generated set closer to the original set; however, the success rate would decrease.

7. **overshoot:** this shifts the input tensors in the following ratio (1+overshoot) * δ; where δ is the perturbed noise.

8. **epsilon ϵ:** this is a cautionary measure to prevent gradients from turning into infinity. When a zero division happens while crafting attacks, a small ϵ value is added to prevent NaN in the backpropagation calculation.

	C-LowProFool	**C-DeepFool**
batch_size	64	64
cor_weights	FALSE	FALSE
max_iters	2000	2000
perturb_threshold	0.05	0.05
alpha	0.01	0.001
lambda	4.5	NA
overshoot	NA	0.02
epsilon	NA	0.00001

Tabelle 5.2.: Hyperparameters for the best performance

Table 5.3 provides the results of a drop in performance (in terms of Accuracy, Precision, Recall, and F1-Score) for neural network on the adversarial test set generated by the C-LowProFool & C-DeepFool method. We can see a significant drop in the values in terms of overall performance, with C-DeepFool having the highest success rate. These experiments run for five iterations and report the mean values.

Metrics	Original Samples	Adversarial Samples	
		C-Lowprofool	C-Deepfool
	NSL-KDD		
Accuracy	0.772	0.114	0
Precision	0.717	0.103	0
Recall	0.75	0.113	0
F1-Score	0.7	0.108	0
	CICDDoS		
Accuracy	0.972	0.593	0.516
Precision	0.982	0.694	0.469
Recall	0.972	0.591	0.513
F1-Score	0.976	0.585	0.488
	CICIDS		
Accuracy	0.977	0.271	0
Precision	0.974	0.492	0.002
Recall	0.972	0.269	0
F1-Score	0.972	0.347	0.001

Tabelle 5.3.: Performance of ML-based IDS on the original and adversarial test sets

Both the algorithms are successful in deteriorating the performance of baseline results on all three datasets. The first column in Table 5.3 is the results from the neural network on the best configurations. Finally, two sets of adversarial samples are generated, one for each algorithm and later used to evaluate the performance. We can see that for NSL-KDD, there is a significant drop in all metrics for C-LowProFool, and C-DeepFool reduces the metrics to absolute zero. The newer datasets CICIDS and CICDDoS, are comparatively robust and less vulnerable in adversarial space, and the confusion matrix of all class labels is reviewed in Section 5.2.2.

It is important to note that these drops in performance are observed, ensuring the functionality of perturbed samples. Validating the input coherence for a specific ground truth class label often limits manipulation in the adversarial direction, i.e., the constraint space is too tight to fool a classifier into misclassifying.

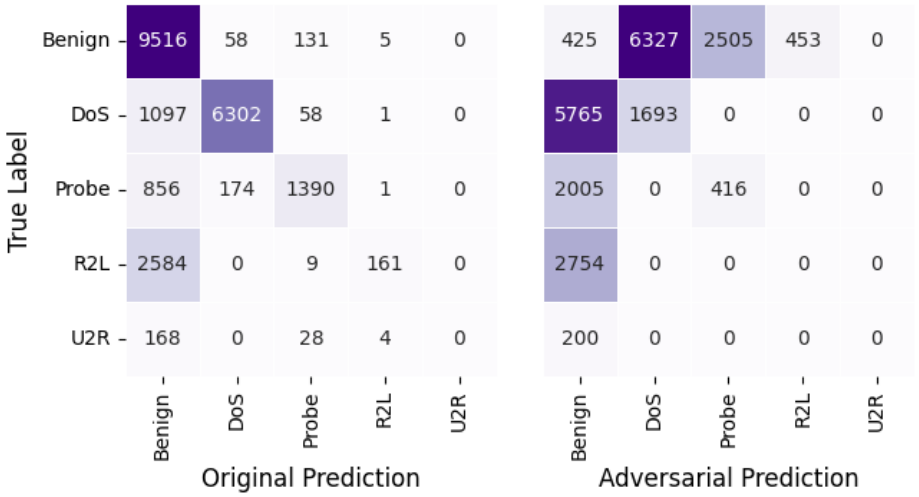


Abbildung 5.1.: NSL-KDD confusion matrix for C-LowProFool

5.2.2. Confusion Matrices

Figures 5.1 & 5.2 show the confusion matrices for the NSL-KDD dataset on both these algorithms. C-LowProFool has entries on its diagonal, which represents the labels that could not be perturbed, and *DoS* has the highest frequency. A justification for such examples is values in the critical features that are not manipulated or fall under the perceptibility constraint. In contrast, C-DeepFool does not have this constraint and thus can generate stronger attacks by misclassifying all the class labels (diagonal has no elements).

For space constraints, plots for CICIDS are not provided and only report the mean results in Tables 5.3. Figures 5.3 & 5.4 justifies the robustness of the CICDDoS dataset; as we can see, the main diagonal has high frequencies. Specifically, class labels *LDAP, MSSQL, UDP* are not at all vulnerable in adversarial space, and to some extent, other labels too cannot get misclassified easily. However, when the constraint space is relaxed by not considering the imperceptibility factor in C-DeepFool, we see the confusion matrix gets rid of the partial cases like *NetBIOS, Syn, UDPLag* and misclassifies them completely.

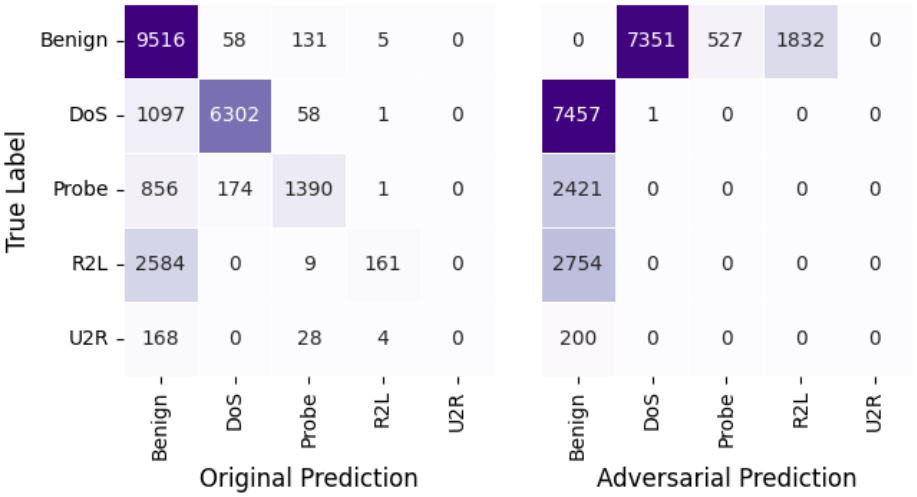


Abbildung 5.2.: NSL-KDD confusion matrix for C-DeepFool

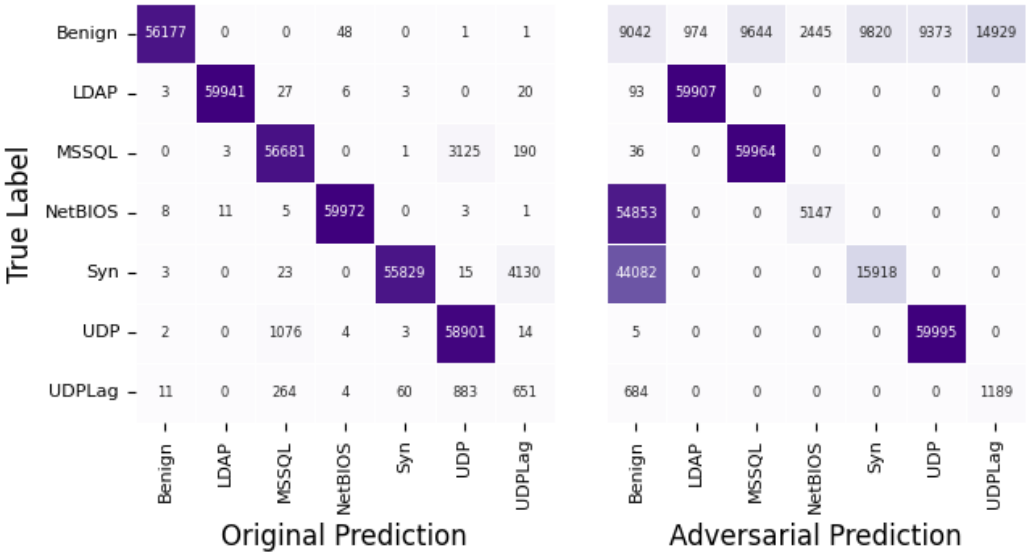


Abbildung 5.3.: CICDDoS confusion matrix for C-LowProFool

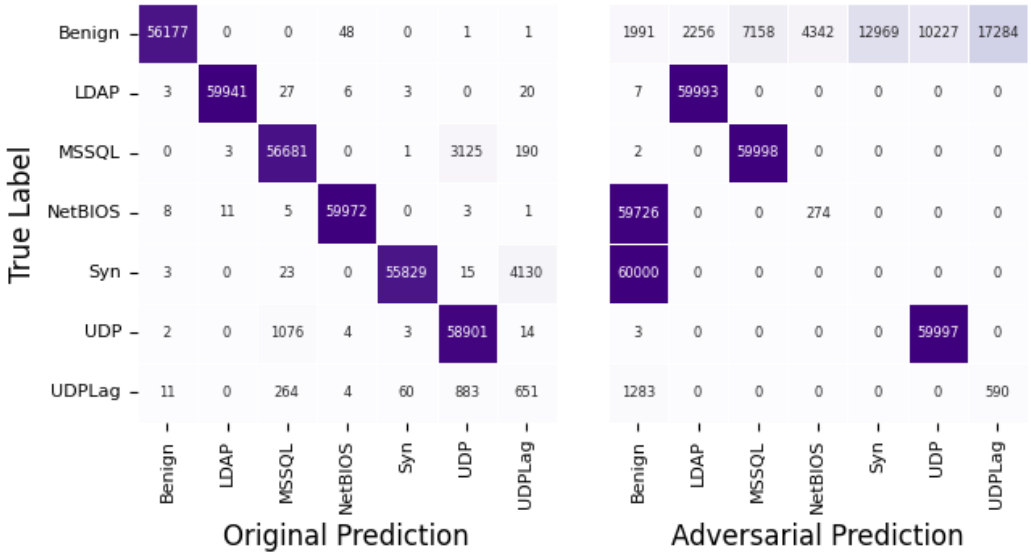


Abbildung 5.4.: CICDDoS confusion matrix for C-DeepFool

5.3. Ablation Study

In this section, the results from both the attack algorithms are discussed, and why C-LowProFool has a lower success rate is reasoned. Let us start with comparing these methods based on metrics defined in Section 3.5.2 and plot the perturbed features for the entire adversarial set. The following subsections describe the concept of imperceptibility for C-LowProFool and the transferability of these samples across other model classes.

5.3.1. Comparison

C-DeepFool has a higher success rate than C-LowProFool, and it is essential to note that C-DeepFool does not guarantee imperceptibility. It can generate stronger attacks as its objective function minimizes the l_2 norm of the perturbation δ under no constraint concerning the perceptibility of the manipulated feature vector. In contrast, in C-LowProFool, the overall optimization is a combination of cross-entropy loss and perceptibility loss. The ratio of these losses is controlled by a hyper-parameter λ.

Table 5.4 provides a comparison of both methods and looking at the values, and we see that C-DeepFool always has a higher value of attack score. These strong attacks are balanced with a higher value for perturbation distance which provides a penalty on the sample being less realistic than C-LowProFool. In order to best compare these methods, the number of featured allowed for manipulation are fixed on all three datasets.

	C-Lowprofool	C-Deepfool
	NSL-KDD	
Success Rate	0.906	1
Attack Score	80.7	94.3
Perturbation Distance	0.327	0.576
# Features Perturbed	17 / 40	17 / 40
	CICDDoS	
Success Rate	0.437	0.493
Attack Score	68.9	199.4
Perturbation Distance	0.336	0.755
# Features Perturbed	31 / 58	31 / 58
	CICIDS	
Success Rate	0.729	0.1
Attack Score	45.2	91.9
Perturbation Distance	0.13	1.26
# Features Perturbed	31 / 58	31 / 58

Tabelle 5.4.: Comparison of C-LowProFool & C- DeepFool

Next, the perturbed features are analyzed for each method, as shown in Figure 5.5 & 5.6. The data values are normalized between (0 and 1) for meaningful plots. The interpretation of the following plots in two halves:

1. The first half of each subfigure shows a line plot of the original and adversarial values for the features (in the x-axis). The values are on the y-axis represent the mean values for all the samples in each dataset. We can observe that for C-LowProFool, these lines are always close or comparatively closer when compared with C-DeepFool plots. This provides a visual interpretation of the metric *perturbation distance*.

2. Looking at the second half of each plot, we observe a line plot representing the feature importance value in percentage and a bar plot representing the perturbation δ introduced over the original feature vector. The values on the y-axis are taken as a mean of all samples in the dataset. It is essential to mention that the weight vector represented as a line plot shows the euclidian norm of the original weight values. One can get an idea of which features are most vulnerable from the x-axis, and a bar with a higher magnitude represents the vulnerability for the model.

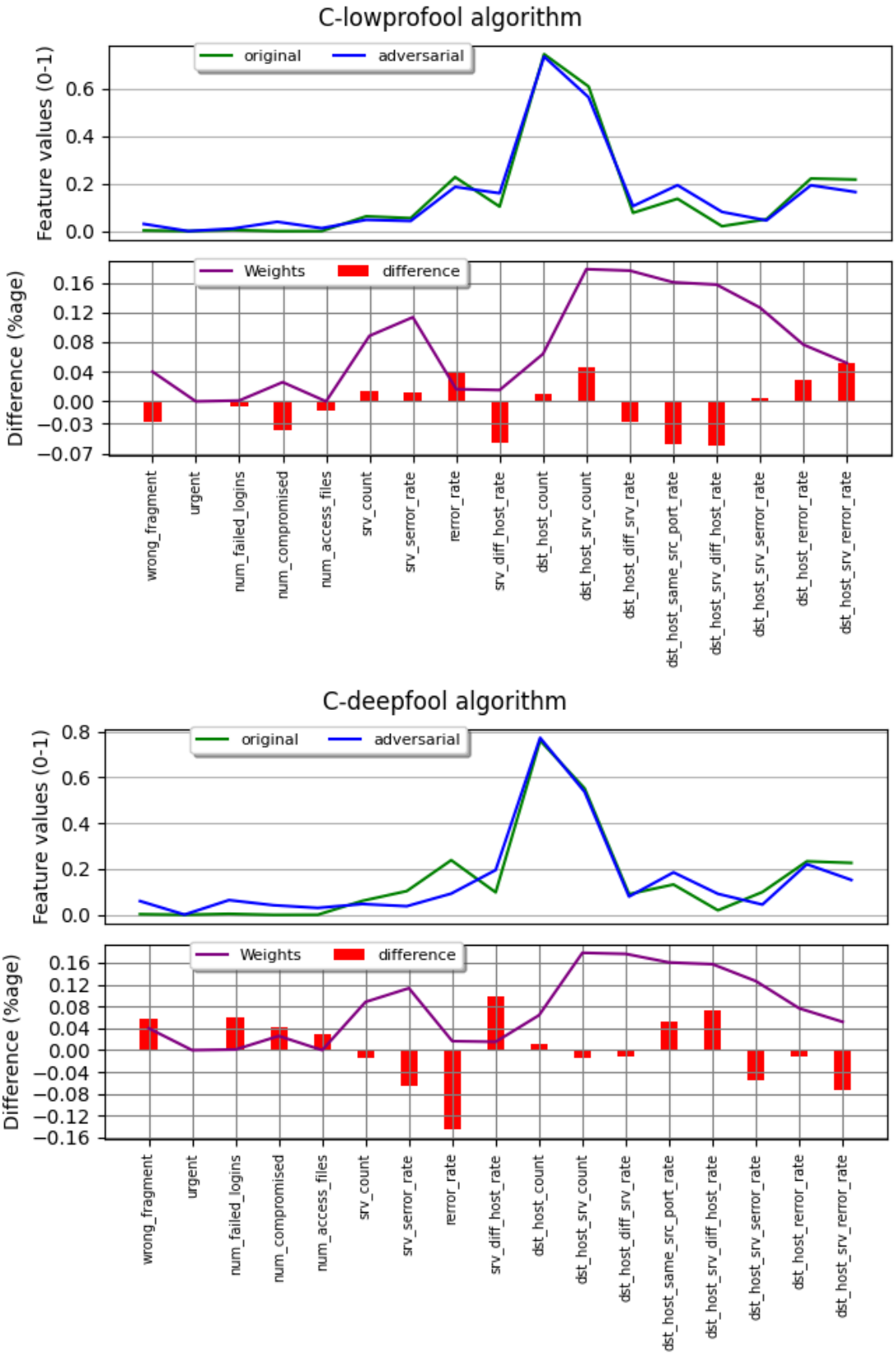


Abbildung 5.5.: NSL-KDD Perturbed Features Analysis

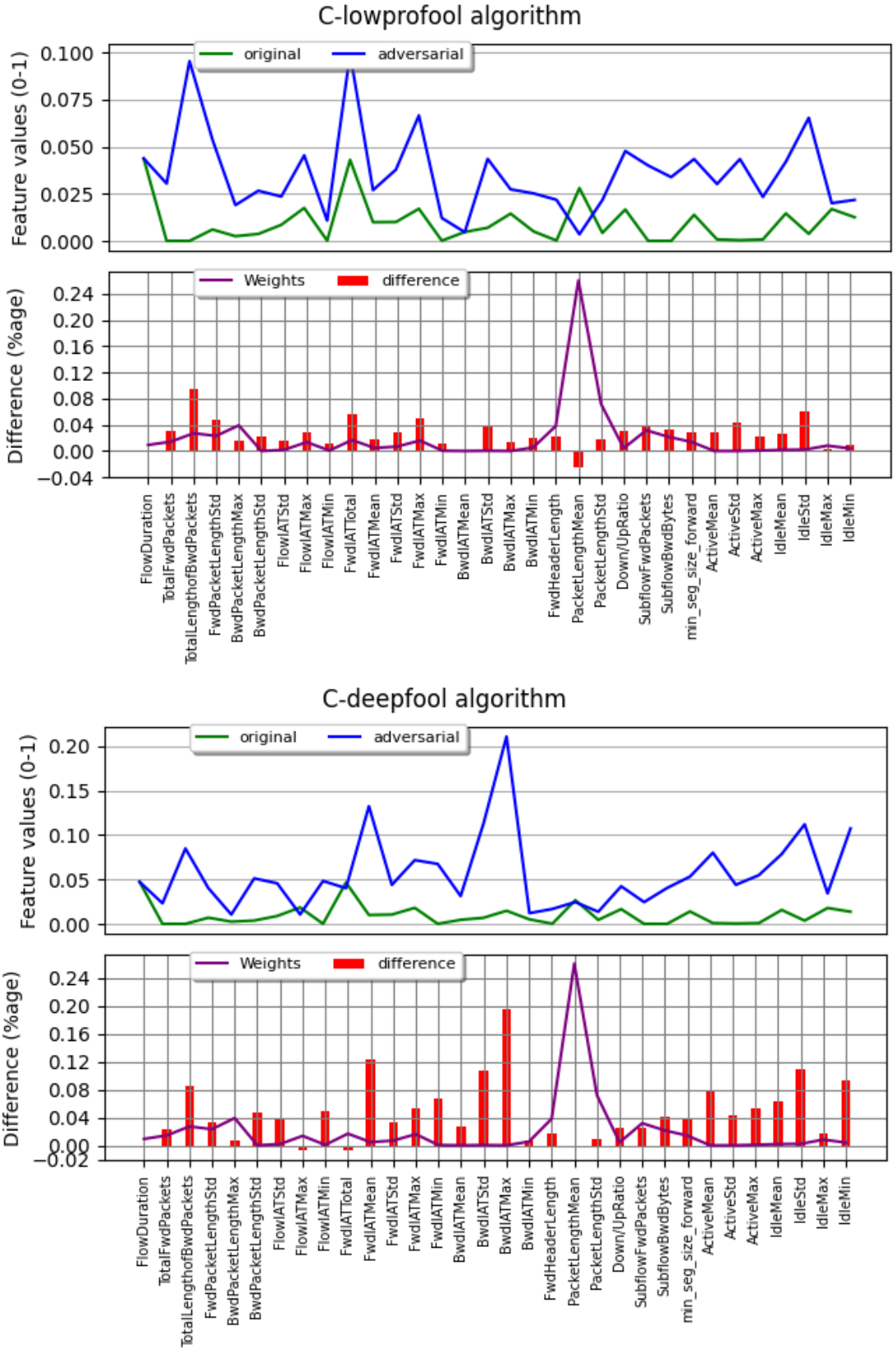


Abbildung 5.6.: CICDDoS Perturbed Features Analysis

In Figure 5.5 *rerror_rate* has a low weight value and thus maximum perturbation. Similarly, *dst_host_srv_count* is the most critical feature, and C-LowProFool manipulated this feature to a meager extent. These features inform that an attacker needs to perturb more on insignificant features when compared to significant ones.

The C-DeepFool algorithm minimizes the l_2 norm of δ under no perceptibility constraint, which naturally makes it easier to generate adversarial examples. This is also justified with the first half of each subplot in Fig,5.6 where the adversarial line is too away from the original line.

5.3.2. Imperceptibility

In order to measure imperceptibility on a quantitative level as defined in the equation 3.8. In this sub-section, let us discuss the qualitative visualization of each dataset's highest & lowest perceptible samples. These plots are only applicable for C-LowProFool, and each sample vector is represented using a triplet of grayscale images. Each image is further split onto *n* shades based on *n* features of the visualized instance. A darker tone represents a higher feature value of the instance. For simplicity, all the feature values were initially normalized in a range of (0, 1).

Figure 5.7 which shows the highest and lowest perceptible sample from the generated adversarial set. The first axis in each figure is the original feature vector $\mathbf{x}$, the second axis denotes the added perturbation δ, and the third axis is the adversarial counterpart $\mathbf{x}'$. The dark tone represents a higher value of 1, and the lighter tone represents a 0. This visualization aims to justify the factor of imperceptibility produced and compare how similar the adversarial sample is to its original form.

An interpretation of this visualization on the NSL-KDD dataset would be:

1. Highest perceptible: The first sub-figure shows the highest perceptible adversarial sample. The model originally predicted it to be a *DoS* label with 100 % confidence, a perturbation δ as shown in second axis when added to $\mathbf{x}$ yeilding $\mathbf{x}\prime$. The adversarial sample is predicted *Benign* with confidence of 52%.

2. Lowest perceptible: The second sub-figure shows the least perceptible adversarial sample. The original label predicted is *Probe* label with 100% confidence and after adding the perturbation δ the model predicted *Benign* label with 66% confidence. As we see, the perturbation is barely visible in the plot, and the original and adversarial vectors look almost similar.

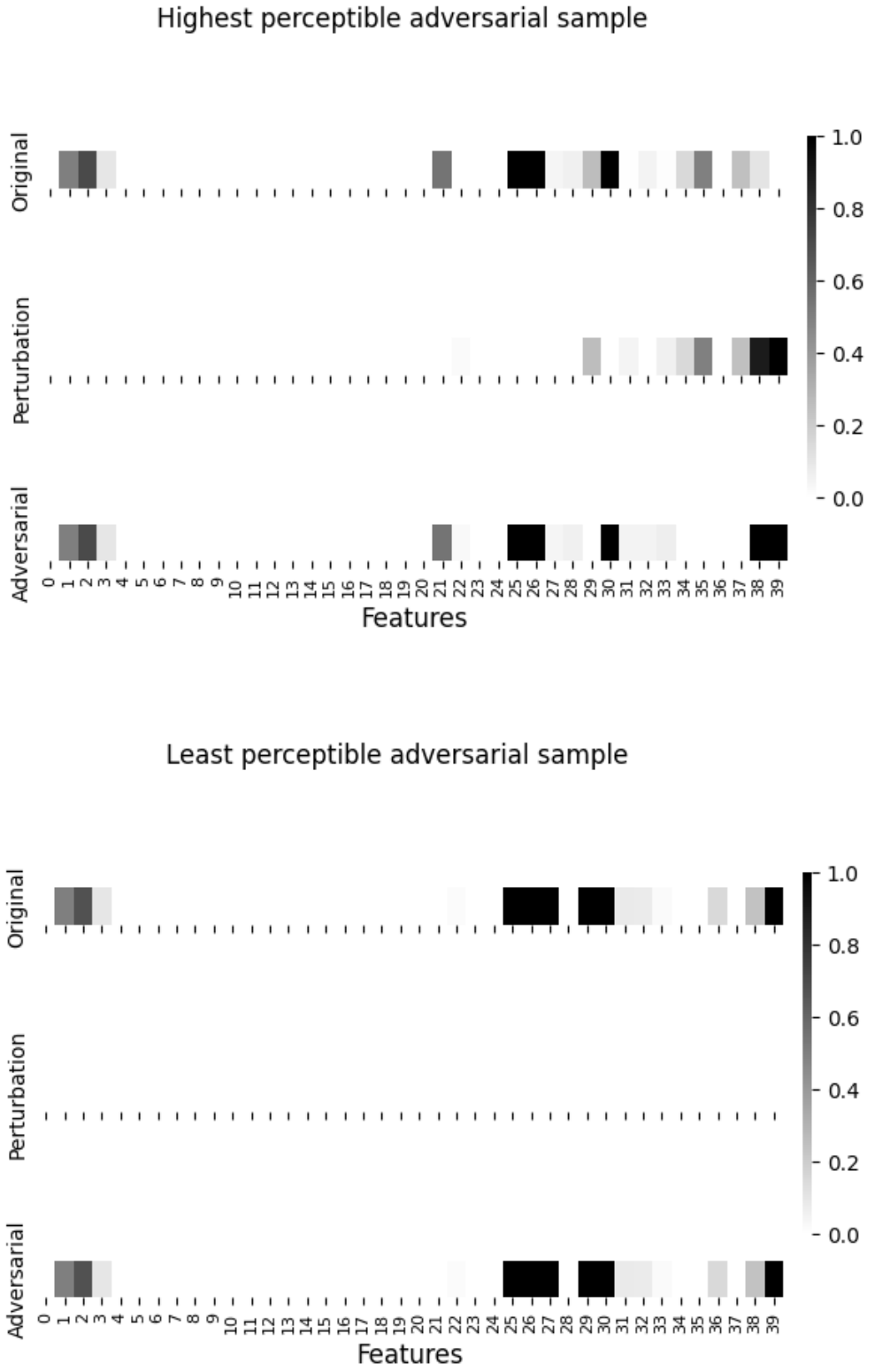


Abbildung 5.7.: Samples from NSL-KDD representing *highest* & *lowest* perceptibility value.

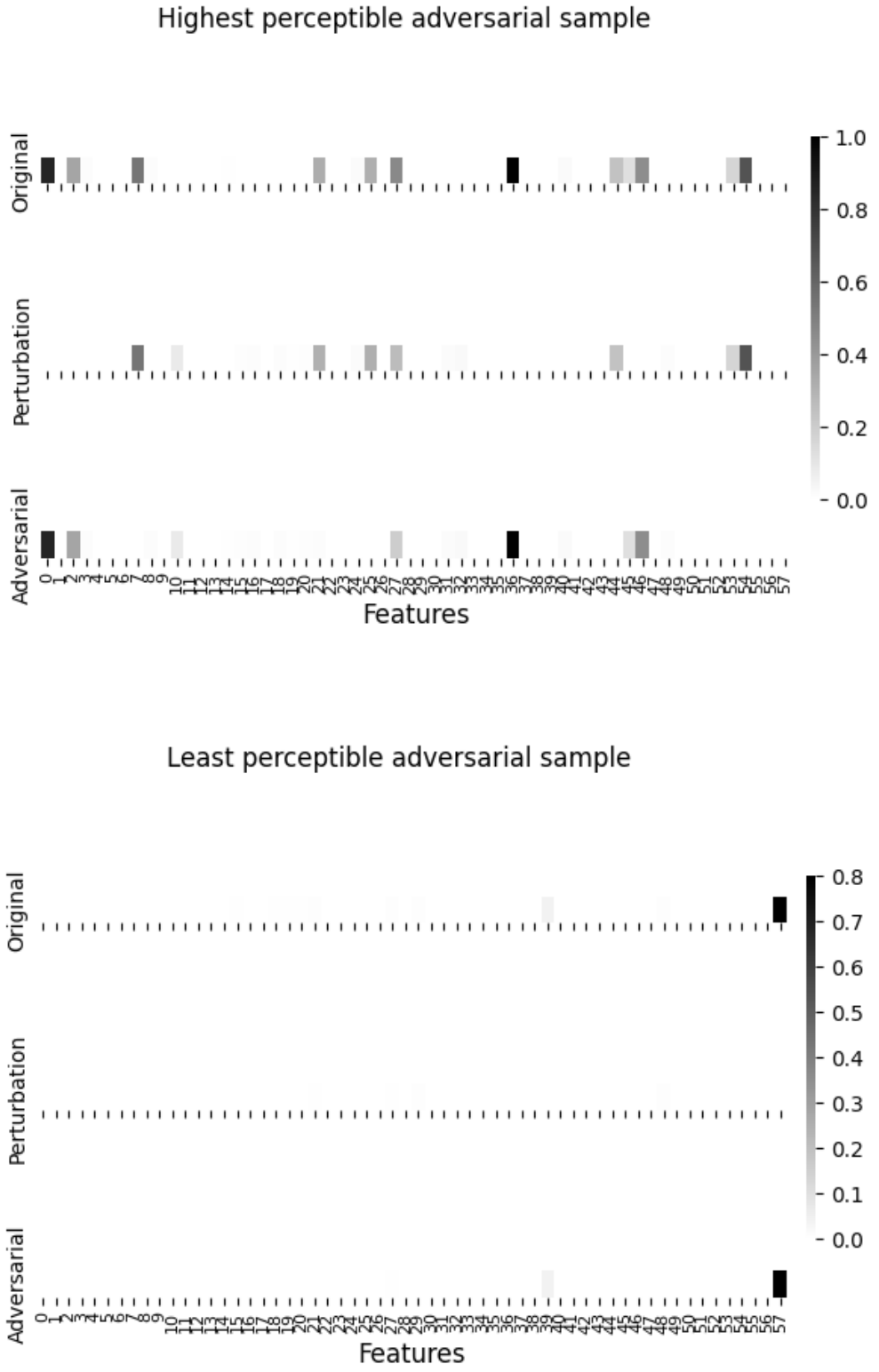


Abbildung 5.8.: Samples from CICDDoS representing *highest* & *lowest* perceptibility value.

Another interpretation of Figure 5.8 on CICDDoS dataset would be:

1. Highest perceptible: The first sub-figure shows the highest perceptible adversarial sample. The model originally predicted it to be a *Benign* label with 100 % confidence, a perturbation δ as shown in second axis when added to $\mathbf{x}$ yeilding $\mathbf{x}\prime$. The adversarial sample is predicted *UDPLag* with confidence of 55%.

2. Lowest perceptible: The second sub-figure shows the least perceptible adversarial sample. The original label predicted is *UDP* label with 100% confidence, and after adding the perturbation δ, the model predicted *Benign* label with 100% confidence. Despite the perturbation being invisible, it is also a potent attack as the prediction shifted with confidence 100% for each.

6. Conclusion & Future Work

Adversarial machine learning is an emerging research field raising concern for the usage of machine learning across various domains. In this dissertation, we studied the effectiveness of neural networks for intrusion detection with a focus on DDoS attacks and exposed the brittleness of machine learning models. The adversarial attacks generated by LowProFool outperform DeepFool significantly in terms of perceptibility; however, DeepFool samples have a higher attack score and success rate. The adversary had no information about the training data distribution except the feature importance vector. With the results obtained from the experiments, we can answer the five questions proposed in Chapter 1:

1. **How effective are the attack algorithms against the newer intrusion datasets?**

   - CICDDoS2019 proved to be the most robust dataset for each algorithm, specifically class labels *MSSQL, UDP, and UDPLag* showed high resilience towards getting misclassified. However, for the other two datasets, NSL-KDD and CICIDS2017, we observed the baseline performance metrics drop to 0% in each case.

2. **Which attack methods are applicable in tabular domains, where features vary in their data types (e.g., binary, discrete, or continuous)?**

   - Based on the discussion in the above sections, it is justifiable to say JSMA, LowProFool, and DeepFool are applicable in tabular domains as they do not modify all the features. However, for the experiments performed, DeepFool outperforms the LowProFool in terms of attack score, and both maintain the input coherence of the network traffic sample.

3. **Do adversarial examples exist in constrained scenarios maintaining the input coherence and the functionality of a network traffic instance?**

   - Yes, results show that with enough iterations, adversarial examples exist for most samples in constrained domains. However, we also observed some exceptions in the CICDDoS2019 dataset where it was impossible to manipulate some specific class labels and also maintain their functional nature.

4. **Is it possible to measure (as a qualitative/quantitative value) the imperceptibility of perturbation without a domain expertise?**

   - Yes, it is possible to measure the imperceptibility of the perturbation as the l_p norm of the noise vector or as the euclidian distance between the original and adversarial counterpart. The plots also represent the least and most perceptible sample on each dataset with grayscale heatmap representations.

5. **Are these adversarial counterparts transferable across models and datasets?**

   - Due to the amount of work and time constraints, it was not verified under the scope of this work. However, in theory, it is possible to transfer these perturbations across models but not datasets as the feature representation is unique for each of them and remains one of the future work to be implemented.

Future Work

1. Investigate and reason on why some class labels in CICDDoS2019 showed high robustness towards the attack algorithms. Based on the unit-level analysis, a few features for some class labels prevent such network traffic samples from getting misclassified along with maintaining their input coherence.

2. [1] explored the transferability concept for adversarial examples. The key observation was that when a model successfully misclassifies an input, it is often misclassified by any other model, even if they have a different architecture or are trained on a diverse training set. [2] Supports these conclusions and further reasons them to be successful only when these models are trained via gradient descent. We suggest conducting more experiments to verify the concept of transferability as an extension to this research work.

3. Currently, there is no single defense mechanism that is entirely satisfactory. Although adversarial training shows some promising results, it does not provide a guarantee for robustness. Performing attacks was the first phase to motivate the need for ML security in IDS. A promising line of work is to find relevant defense algorithms that can prevent these attacks and build a reliable networked system.

Literaturverzeichnis

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[2] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," 2015.

[3] M.-I. 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, "Adversarial robustness toolbox v1.0.0," 2019.

[4] V. Ballet, X. Renard, J. Aigrain, T. Laugel, P. Frossard, and M. Detyniecki, "Imperceptible adversarial attacks on tabular data," 2019.

[5] R. Kemmerer, "Cybersecurity," in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 705–715.

[6] J. P. Anderson, "Computer security threat monitoring and surveillance," *Technical Report, James P. Anderson Company*, 1980. [Online]. Available: https://ci.nii.ac.jp/naid/10014688737/en/

[7] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[8] M. Zamani and M. Movahedi, "Machine learning techniques for intrusion detection," 2015.

[9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014.

[10] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," 2018.

[11] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," 2019.

[12] K. Dvijotham, S. Gowal, R. Stanforth, R. Arandjelovic, B. O'Donoghue, J. Uesato, and P. Kohli, "Training verified learners with learned verifiers," 2018.

[13] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, p. 828â841, Oct 2019. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2019.2890858

[14] A. Raghunathan, J. Steinhardt, and P. Liang, "Semidefinite relaxations for certifying robustness to adversarial examples," 2018.

[15] M. Sharif, L. Bauer, and M. K. Reiter, "On the suitability of l_p-norms for creating and preventing adversarial examples," 2018.

[16] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, 2019. [Online]. Available: https://doi.org/10.1186/s42400-019-0038-7

[17] N. G. Relan and D. R. Patil, "Implementation of network intrusion detection system using variant of decision tree algorithm," in *2015 International Conference on Nascent Technologies in the Engineering Field (ICNTE)*, 2015, pp. 1–5.

[18] H. Liu and B. Lang, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey," *Applied Sciences*, vol. 9, no. 20, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4396

[19] D. Dasgupta, Z. Akhtar, and S. Sen, "Machine learning in cybersecurity: a comprehensive survey," *The Journal of Defense Modeling and Simulation*, 2020. [Online]. Available: https://doi.org/10.1177/1548512920951275

[20] P. Casas Hernandez, J. Mazel, and P. Owezarski, "Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, Apr. 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00736278

[21] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers Security*, vol. 86, pp. 147–167, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481930118X

[22] O. Ibitoye, R. Abou-Khamis, A. Matrawy, and M. O. Shafiq, "The threat of adversarial attacks on machine learning in network security – a survey," 2020.

[23] N. Martins, J. M. Cruz, T. Cruz, and P. Henriques Abreu, "Adversarial machine learning applied to intrusion and malware scenarios: A systematic review," *IEEE Access*, vol. 8, pp. 35 403–35 419, 2020.

[24] A. Drewek-Ossowicka, M. PietroÅaj, and J. Ruminski, "A survey of neural networks usage for intrusion detection systems," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, 01 2021.

[25] L. Koc, T. Mazzuchi, and S. Sarkani, "A network intrusion detection system based on a hidden naÃ¯ve bayes multiclass classifier," *Expert Systems with Applications*, vol. 39, p. 13492â13500, 12 2012.

[26] D. Tong, A. Murray, and N. Xiao, "Heuristics in spatial analysis: A genetic algorithm for coverage maximization," *Annals of the Association of American Geographers*, vol. 99, pp. 698–711, 10 2009.

[27] G. Parry and S. Kumar, "Genetic algorithms in intrusion detection systems: A survey," *International Journal of Innovation and Applied Studies*, vol. 5, pp. 233–240, 01 2014.

[28] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, vol. 41, no. 4, Part 2, pp. 1690–1700, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417413006878

[29] K. Goeschel, "Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive bayes for off-line analysis," in *SoutheastCon 2016*, 2016, pp. 1–6.

[30] O. Er, "Network intrusion detection: A usability comparison study on neural networks," *ELECTRONICS WORLD*, vol. 120, pp. 26–32, 11 2014.

[31] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ser. AISec '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 43â58. [Online]. Available: https://doi.org/10.1145/2046684.2046692

[32] Y. Vorobeychik, M. Kantarcioglu, R. Brachman, P. Stone, and F. Rossi, 2018.

[33] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, "Robustness of classifiers: from adversarial to random noise," 2016.

[34] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, p. 984â996, Apr 2014. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2013.57

[35] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Å rndiÄ, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," *Lecture Notes in Computer Science*, p. 387â402, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40994-3_25

[36] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," 2017.

[37] F. TramÃ¨r, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," 2016.

[38] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo," *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, Nov 2017. [Online]. Available: http://dx.doi.org/10.1145/3128572.3140448

[39] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, p. 317â331, Dec 2018. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2018.07.023

[40] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" 2018.

[41] A. Paudice and E. C. Lupu, "Label sanitization against label flipping poisoning attacks," 2018.

[42] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on pdf malware classifiers," 01 2016.

[43] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 16â25. [Online]. Available: https://doi.org/10.1145/1128817.1128824

[44] M. Barreno, B. Nelson, A. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, pp. 121–148, 11 2010.

[45] B. Biggio, I. Corona, B. Nelson, B. I. P. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, , and F. Roli, "Security evaluation of support vector machines in adversarial environments," 2014.

[46] F. Assion, P. Schlicht, F. GreÃner, W. GÃ¼nther, F. HÃ¼ger, N. Schmidt, and U. Rasheed, "The attack generator: A systematic approach towards constructing adversarial attacks," 2019.

[47] P. Laskov and M. Kloft, "A framework for quantitative security analysis of machine learning," in *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*, ser. AISec '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1â4. [Online]. Available: https://doi.org/10.1145/1654988.1654990

[48] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," 2016.

[49] M. Rigaki, "Adversarial deep learning against intrusion detection classifiers," Master's thesis, LuleÃ¥ University of Technology, Computer Science, 2017.

[50] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38 367–38 384, 2018.

[51] S. M. Hossin M, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining Knowledge Management Process*, vol. 5, no. 2, pp. 01–11, 2015.

[52] R. Sheatsley, N. Papernot, M. Weisman, G. Verma, and P. McDaniel, "Adversarial examples in constrained domains," 2020.

[53] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 305–316.

[54] M. Małowidzki, P. Berezinski, and M. Mazur, "Network Intrusion Detection: Half a Kingdom for a Good Dataset," 2015.

[55] W. Haider, J. Hu, J. Slay, B. P. Turnbull, and Y. Xie, "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling," *Journal of Network and Computer Applications*, vol. 87, pp. 185–192, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804517301273

[56] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, p. 262â294, Nov. 2000. [Online]. Available: https://doi.org/10.1145/382912.382923

[57] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.

[58] M. Haddadi and R. Beghdad, "DoS-DDoS: TAXONOMIES OF ATTACKS, COUNTERMEASURES, AND WELL-KNOWN DEFENSE MECHANISMS IN CLOUD ENVIRONMENT," *EDPACS*, vol. 57, no. 5, pp. 1–26, 2018. [Online]. Available: https://doi.org/10.1080/07366981.2018.1453101

Appendices

A. Datasets

A.1. CICIDS & CICDDoS

The features in CICIDS2017 and CICDDoS2019 are both developed by CIC and CICFLowMeter was was used commonly. Due to the this reason their descriptions stay similar and is commonly described in table A.1. In the following subsection, we provide some additional information regarding a subset of these features. This information was collected from multiple sources as part of data understanding at the very early stages of this thesis work.

Feature	Description
Source Port	Source Port
Destination Port	Destination Port
Protocol	Protocol Type
Flow duration	Duration of the flow in Microsecond
total Fwd Packet	Total packets in the forward direction
total Bwd packets	Total packets in the backward direction
total Length of Fwd Packet	Total size of packet in forward direction (bytes)
total Length of Bwd Packet	Total size of packet in backward direction (bytes)
Fwd Packet Length Max	Maximum size of packet in forward direction
Fwd Packet Length Min	Minimum size of packet in forward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Fwd Packet Length Mean	Mean size of packet in forward direction
Fwd Packet Length Std	Standard deviation size of packet in forward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Bwd Packet Length Std	Standard deviation size of packet in backward direction
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second
Flow IAT Mean	Mean time between two packets sent in the flow
Flow IAT Std	Standard deviation time between two packets sent in the flow (in microseconds)

Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Total	Total time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the backward direction
2***FWD Packets/s**	2*Number of forward packets per second

Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a packet
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWR Flag Count	Number of packets with CWR
ECE Flag Count	Number of packets with ECE
down/Up Ratio	Download and upload ratio
Average Packet Size	Average size of packet
2***Fwd Segment Size Avg**	2*Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in the backward direction
Fwd Bytes/Bulk Avg	Average number of bytes bulk rate in the forward direction
Fwd Packet/Bulk Avg	Average number of packets bulk rate in the forward direction
Fwd Bulk Rate Avg	Average number of bulk rate in the forward direction
Bwd Bytes/Bulk Avg	Average number of bytes bulk rate in the backward direction
Bwd Packet/Bulk Avg	Average number of packets bulk rate in the backward direction
Bwd Bulk Rate Avg	Average number of bulk rate in the backward direction
Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction

Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
Fwd Init Win bytes	The total number of bytes sent in initial window in the forward direction
Bwd Init Win bytes	The total number of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Mean	Mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Min	Minimum time a flow was active before becoming idle
Idle Mean	Mean time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Min	Minimum time a flow was idle before becoming active
Inbound	Direction of traffic

Tabelle A.1.: Data description for CICIDS & CICDDoS

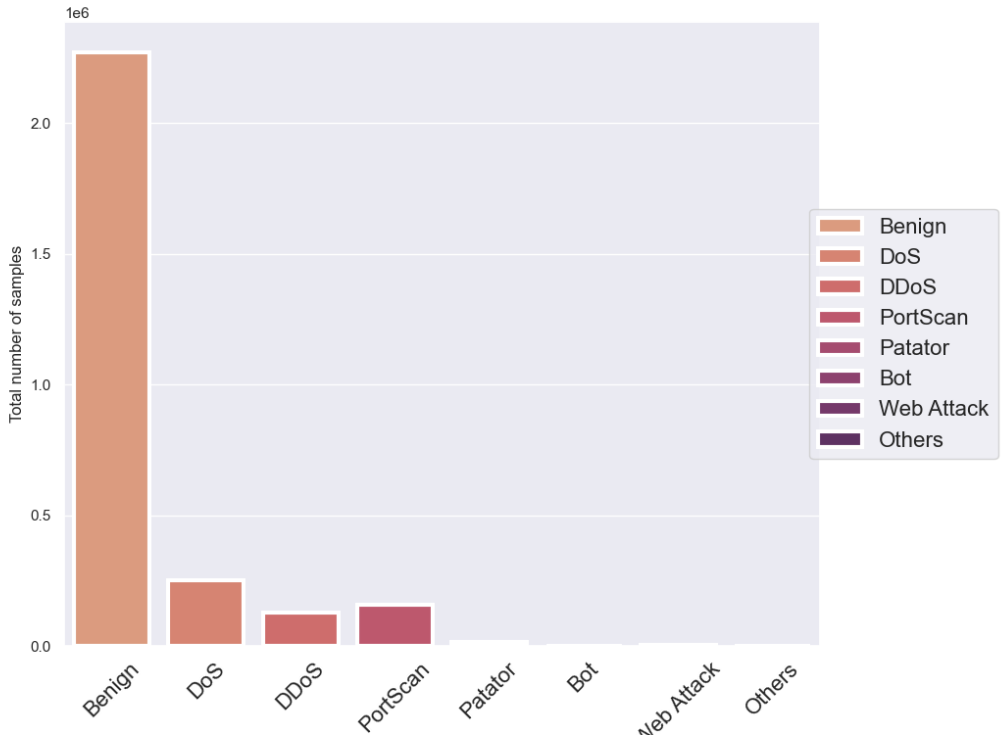


Abbildung A.1.: Data distribution for CICIDS

Feature Description

1. **Source Port:** As per [1] Port 0 is a system port used in programming APIs (not in communication between hosts), it requests a system-allocated dynamic port. Many TCP and UDP network applications do not have their own system port and must obtain from respective operating systems each time they run. Usually, Port 0 is assigned, which further triggers an automatic search provides a suitable available port. Note: Port 0 is never assigned for communication itself. Their exact functionality varies from OS, but this general idea stays common. We do not drop malicious records if port number is 0 as it can be instantiated to trick an IDS.

2. **Destination Port:** Additional to above information, many Internet Service Providers automatically drop packets with Port number 0 and an observation the dataset observed is: whenever Source Port is 0 Destination Port is also 0. Such behavior is mainly observed for normal traffic samples and less than 0.01% in case of malicious sample.

3. **Protocol:** A value of 17 denotes UDP, 6 denotes TCP and 0 represents IPv6 Hop-by-Hop

[1] https://en.wikipedia.org/wiki/List$_of_TCP_and_UDP_port_numbers$

option [2]. We should drop records with protocol 0 in all malicious samples, as they can only be TCP or UDP protocol (authors did not consider any other protocol to generate these malicious traffic). Observation if Protocol is 0 then Source and Destination Port are also 0.

4. **Total Packet length:** Total packets and size of total packets do not have any relationship between them. The size of packets fall under a range of [0-65535] [3] [4].

5. **Fwd/Bwd Packet Length Mean:** All features with means happen to be direct formulation of other two primary columns. We happen to drop such columns in the preprocessing pipeline to avoid the correlation. This method is trivial in most downstream machine learning tasks.
$$PacketLengthMean = \frac{TotalPacketLength}{TotalPacket} \tag{A.1}$$

6. **Flow Bytes/s:** Same explanation as above.
$$FLowBytes/s = \frac{totallengthfwdpackets + totallengthbwdpackets}{FlowDuration} * 1000000 \tag{A.2}$$

7. **Flow Packets/s:** Same explanation as above.
$$FLowPackets/s = \frac{totalfwdpackets + totalbwdpackets}{FlowDuration} * 1000000 \tag{A.3}$$

8. **Fwd/Bwd PSH Flags:** Informs the HOST that the data should be pushed to the upper layers immediately [5].

9. **Fwd/Bwd Header Length:** Negative values were observed in the dataset. We could not drop records based on this filter due to large proportion of records. This needs further investigation on the reason for negative values for length, and for simplicity we decided to drop these columns.

10. **Fwd/Bwd Packets /s:** Secondary features as they are direct formulations of some primary features.
$$Fwd/BwdPackets/s = \frac{totalfwd/bwdpacket}{FlowDuration} \tag{A.4}$$

11. **Packet Length Min/Max:** The formulation can be taken from the following: [min(e1, e2) if e1 !=0 else e1
for e1, e2 in zip(data['FwdPacketLengthMin'], data['BwdPacketLengthMin']).

    Similarly for max() operation.

[2] https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
[3] https://jvns.ca/blog/2017/02/07/mtu/
[4] https://arxiv.org/pdf/1809.03486.pdf
[5] https://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/

12. **URG Flag:** Informs HOST that certain segment in the data is urgent and should be prioritized. However, they are not used by modern protocols [6].

13. **Fwd/Bwd Init Win Bytes:** UDP protocols should have a value of -1. If 0 is observed this means the TCP buffer in layer4 is full, i.e., it has crossed 65535 and the data transmission halted. Normal procedure is a wait for the buffer to get clear [7]. We decided to drop the erroneous records where UDP protocols is not equal to -1.

A.2. NSL-KDD

NSl-KDD contains records of internet traffic recorded by a simple intrusion detection network. As discussed in Section 4.1 about different labels, DoS acts differently from other three attacks. DoS attempts to utilize all the resource or bandwidth of the target system, whereas the remaining quietly infiltrate the system undetected. Table A.2 below, provides a breakdown of different subclasses observed in the dataset under each category type. A detailed data description is provided here [8].

Class label	Subclass
DoS	apache2; back; buffer_overflow; land; mailbomb; neptune; pod; processtable; smurf; teardrop; udpstorm
U2R	xterm; sqlattack; rootkit; ps; perl; loadmodule; httptunnel; buffer_overflow
R2L	ftp_write; guess_passwd; imap; multihop; named; phf; sendmail; snmpgetattack; snmpguess; spy; warezclient; warezmaster; worm; xlock; xsnoop
Probe	satan; saint; portsweep; nmap; mscan; ipsweep

Tabelle A.2.: Sub classes under each attack label

Additionally, the feature information for all the traffic records can be broken under four main categories depending on the type of information they carry:

1. **Intrinsic features**: taken from packet headers without the information about the payload itself. They carry some basic information about a individual packet, feature sequence [1-9] fall under this category.

2. **Content features**: usually packets are sent in multiple pieces instead of a single flow. These feature sequences [10-22] contain information about the original packet structure and also the payload.

3. **Time-based features**: mostly contain the analysis behaviour of network traffic over a two-second interval (e.g. how many connection requests to a host). Feature sequence

[6] https://datatracker.ietf.org/doc/html/rfc3168
[7] https://accedian.com/blog/tcp-receive-window-everything-need-know/
[8] https://docs.google.com/spreadsheets/d/1oAx320Vo9Z6HrBrL6BcfLH6sh2zIk9EKCv2OIaMGmwY/edit

[23-31] are mostly about counts and rates rather with no information about the content of network traffic.

4. **Host-based features**: similar to time-based features, except the time window interval is longer than two seconds (e.g. 100 seconds or total connection requests to the same host). Feature sequence [32-41] fall under this category.

These features have different data types, i.e., 4 categorical, 6 binary, 23 discrete and 10 continuous. Categorical features are *protocol type, service, flag and class label*, and they are mostly self explanatory except *flags*. Table A.3 provides a description of the status of the connection for each value.

Flag	**Description**
SF	Normal establishment and termination. Note that this is the same symbol as for the state S1. You can tell the two apart because for S1 there will not be any byte counts in the summary, while SF there will be.
REJ	Connection attempt rejected.
S0	Connection attempt seen, no reply.
S1	Connection established, not terminated.
S2	Connection established and close attempt by originator seen (but no reply from responder.
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO	Connection reset by the originator.
RSTR	Connection reset by the responder.
OTH	No SYN seen, just midstream traffic (a âpartial connectionâ that was not later closed).
RSTOS0	Originator sent a SYN followed by a RST, we never say a SYN-ACK from the responder.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection was âhalfâ open).

Tabelle A.3.: Flag feature description