An Adversarial Attack Framework for Deep Learning-Based NIDS

by

Hesamodin Mohammadian

M.Sc. in Software Engineering, Kharazmi University, 2015
B.Sc. in Software Engineering, Iran University of Science and Technology, 2012

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Ali A. Ghorbani, Ph.D., Faculty of Computer Science Arash Habibi Lashkari, Ph.D., Faculty of Computer Science
Examining Board:	Ronxing Lu, Ph.D., Faculty of Computer Science Sajjad Dadkhah, Ph.D. Faculty of Computer Science Hamed Asgari, Ph.D. Department of Mechanical Engineering
External Examiner:	Khalil El-Khatib, Ph.D. Faculty of Business and Information Technology, Ontario Tech University

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 2023

Abstract

Intrusion detection systems are essential to any cybersecurity architecture, as they play a critical role in defending networks against various security threats. In recent years, deep neural networks have demonstrated remarkable performance in numerous machine learning tasks, including intrusion detection. However, it has been observed that deep learning models are highly susceptible to a wide range of attacks during both the training and testing phases. These attacks can compromise the privacy and security of deep learning models, such as model inversion, membership inference, poisoning, and evasion attacks.

Numerous studies have been conducted to understand and mitigate these attacks to propose more efficient techniques with higher success rates and accuracy in various tasks utilizing deep learning models, such as image classification, face recognition, network intrusion detection, and healthcare applications. Despite the considerable efforts in this area, the network domain still lacks sufficient attention to these attacks and vulnerabilities.

This thesis aims to address this gap by proposing a framework for adversarial attacks against network intrusion detection systems. The proposed framework focuses on both poisoning and evasion attacks. For poisoning, we present a label flipping-based attack, and for evasion, we propose two attacks: FGSM-based and saliency map-based attacks. These attacks are designed by considering the distinct characteristics of network data and flows.

Furthermore, we introduce an evaluation model for the evasion attack based on several carefully selected criteria. To assess the effectiveness of the proposed techniques, we utilize three network datasets that cover a wide range of network attack categories: CIC-IDS2017, CIC-IDS2018, and CIC-UNSW. Through extensive evaluations and analysis, we demonstrate that the proposed methods are highly effective against deep learning-based NIDS and can significantly degrade their performance. With the proposed evasion attacks, we show that each feature has a varying impact on the adversarial sample generation process, and it is possible to execute a successful attack even with a few features involved.

In conclusion, this thesis contributes to network intrusion detection by providing novel and effective approaches for adversarial attacks, shedding light on the vulnerabilities of deep learning-based NIDS, and emphasizing the importance of enhancing their robustness to such attacks.

Dedication

To my lovely wife, Fatemeh, who has always helped and supported me during this long journey;

To my parents, without whom I would never have had the opportunity to follow my studies and reach where I am right now;

And to my home country, Iran, which offered the essential foundations and resources that paved the way for my education and personal growth.

Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my esteemed supervisors, Prof. Ali A. Ghorbani and Dr. Arash Habibi Lashkari. Their unwavering dedication is truly commendable, as they devoted numerous hours to support and guide me on this academic journey. I am grateful for their generosity in sharing their invaluable knowledge and vast experience, which significantly enriched my understanding and contributed to the success of my Ph.D.

My gratitude goes to my thesis advisory committee members, Dr. Ronxing Lu, Dr. Sajjad Dadkhah, Dr. Hamed Asgari, and Dr. Khalil El-Khatib, for patiently reading through my thesis, and providing valuable feedback that has served to improve this thesis.

Table of Contents

List of Tables

List of Figures

Abbreviations

FGSM	Fast Gradient Sign Method
NIDS	Network Intrusion Detection System
ML	Machine Learning
DNN	Deep Neural Network
CNN	Convolutional Neural Network
KNN	K Nearest Neighbour
SVM	Support Vector Machine
SSAE	Stacked Sparse Autoencoder
AE	Autoencoder
RNN	Recurrent Neural Network
MLP	Multi-layer Perceptron
DBN	Dep Belief Network
LSTM	Long Short-term Memory
SAE	Stacked Autoencoder
NLP	Natural Language Processing
JSMA	Jacobain Saliency Map Attack
CW	Carlini Wagner
SRBF	Success Rate of the Best Feature Set
ACAC	Average Confidence of Adversarial Class
TLS	Transport Layer Security
DoS	Denial of Service
DDoS	Distributed Denial of Service
DT	Decision Tree
NB	Naive Bayes
LR	Logistic Regression
RF	Random Forest
F1	F1-score
PC	Precision
RC	Recall

Chapter 1

Introduction

Recently, there has been a significant increase in the usage of machine learning for automated tasks and decision-making problems. ML applications have experienced tremendous growth and are now heavily relied upon in national critical infrastructures and vital sectors such as medicine, healthcare, computer security, spam and malware detection, autonomous driving vehicles, unmanned autonomous systems, and homeland security [27]. These machine-learning models have demonstrated the ability to achieve human-level or even above human-level accuracy in the tasks above. As a result, they are gradually replacing humans or assisting them in carrying out essential duties and making decisions that profoundly impact human lives. Since introducing deep learning, these methods have excelled in various machine learning tasks and garnered significant attention. Consequently, the security of machine learning and deep learning models has become one of the most intriguing research areas in recent years.

There are different kinds of attacks against deep learning models, which can occur both during the training and testing phases [15], such as membership inference attack [103, 49], model inversion attack [30], evasion attack [109], and poisoning attack [112]. In a membership inference attack, the attacker aims to compromise the

model's privacy and determine if a specific record is part of the model's training set. These attacks are particularly significant when dealing with sensitive data, such as health or financial-related information of individuals [115, 20]. The model inversion attack closely resembles the membership inference attack, as it aims to extract sensitive features of the training samples [134]. On the other hand, an evasion attack occurs during the test phase. Its objective is to deceive the trained deep learning model and evade detection by crafting adversarial examples through minor modifications to the original dataset samples [38]. Furthermore, attackers can also impact the model's performance during the training phase, using methods like poisoning attacks in which they try to inject contaminated data into the training set [122].

In current studies, deep learning has demonstrated great potential in enhancing security measures, particularly in areas like malware detection and intrusion detection systems (NIDS). The primary goal of NIDS is to identify and differentiate between attacks directed at a network and benign activities [17]. NIDS operate by continuously analyzing both incoming and outgoing network traffic.

There are two major types of NIDS: signature-based and anomaly-based. These approaches heavily rely on information from past attacks and can only detect known malicious patterns. Consequently, they struggle to keep up with the constantly changing and evolving attack strategies. As network traffic continues to increase exponentially, these traditional techniques have proven to need revision for greater practicality. In contrast, anomaly detection methods based on deep learning techniques offer a more flexible and efficient approach, particularly in networks with high-volume data. This aspect has garnered considerable attention from researchers, making it an attractive area for exploration [117, 33, 6].

As previously mentioned, deep learning models in the network domain are susceptible to various attacks, with poisoning and evasion attacks being the two major categories [91]. In a poisoning attack, the attacker's objective is to manipulate the

training data to compromise the model's performance. By injecting malicious or misleading data into the training set, the attacker aims to distort the model's learning process, leading to incorrect or biased results during inference.

On the other hand, an evasion attack occurs during the test phase. In this type of attack, the attacker tries to deceive the deep neural network (DNN) model by making subtle changes to the input samples. These changes are designed to evade the model's detection and classification mechanisms, potentially causing misclassifications or incorrect outputs [92].

One of the earliest attempts to explore this phenomenon was reported in [24], where Dalvi *et al.* investigated adversarial examples in the context of spam filtering. They observed that even a linear classifier could be easily deceived by making minor modifications to the content of spam emails while still maintaining the overall readability of the spam message. This work represented the first instance of adversarial examples against linear classifiers.

The interest in employing deep learning models surged after Krishevsky *et al.* [58] demonstrated the remarkable success of a Convolutional Neural Network (CNN) [63] in a large-scale visual recognition task [2]. In [109], Szegedy *et al.* explained the susceptibility of deep neural networks to adversarial examples, particularly in the computer vision domain. The term "adversarial attack" is widely used to describe attacks targeting deep learning models. However, it is crucial to acknowledge that it is also specifically used to denote evasion attacks.

In other words, "adversarial attack" is a general term encompassing various attacks that exploit vulnerabilities in deep learning models. Still, it can also be specifically used to emphasize attacks focused on evasion, where malicious inputs are crafted to deceive the model during testing.

While adversarial attacks on deep learning models in computer vision and image processing have been extensively studied and gained popularity in recent years, the

research on such attacks in the network domain has received more attention recently. There have been more attempts to conduct poisoning and evasion attacks against NIDS.

Initially, researchers attempted to employ the same adversarial attack techniques used in image classification models for network traffic classification and anomaly detection. There are several issues with adopting this approach. The primary objective of an adversarial attack is to create a sample that closely resembles the original one while preserving its inherent nature.

In the context of image classification, where inputs are raw images, humans can easily assess and verify whether the changes made to an image still preserve its nature. However, in the network domain, it is not feasible to use a human expert to verify the nature of the changes made to a flow's features. This lack of a human observer poses significant challenges in determining the effectiveness and potential consequences of adversarial attacks on network traffic.

Due to the complexity and uniqueness of network traffic data, devising effective and robust adversarial attacks and defenses in the network domain requires careful consideration and novel approaches that go beyond the methods used in computer vision. Researchers are actively exploring new techniques to address these challenges and enhance the security of deep learning models in the context of network intrusion detection and anomaly detection.

This research evaluates adversarial attacks in NIDS and proposes a comprehensive adversarial attack framework for deep learning-based NIDS. The framework comprises two evasion attacks based on FGSM (Fast Gradient Sign Method) and saliency map, as well as a poisoning attack based on label flipping. For the poisoning attack, stratified sampling is employed to select the target label for label flipping. On the other hand, the evasion attack introduces two methods to identify the best features for generating adversarial examples in network datasets. The proposed framework is

thoroughly evaluated using three well-known network datasets, and extensive analysis is conducted based on several carefully selected criteria.

1.1 Problem Statement

The increasing use of deep learning models in the cybersecurity domain, particularly in network intrusion detection systems, has raised concerns about the security and reliability of these models. One significant threat is the existence of adversarial examples, which are carefully crafted samples designed to deceive deep learning models during inference, leading to a decrease in their performance. Adversarial examples can exploit vulnerabilities in the model and cause it to misclassify inputs that it would otherwise correctly classify.

Apart from the inference-time attacks, there is another type of adversarial attack that occurs during the training phase: the poisoning attack. In this attack, the adversary attempts to manipulate the training data by injecting poisoned samples into the dataset. The poisoned samples are carefully chosen to influence the model's behavior during training, leading to compromised performance or even backdoor vulnerabilities in the resulting model.

Most of the researcher's endeavors related to adversarial attacks have been focused on the image processing and computer vision domain. While some published works explore adversarial attacks in the network security domain, researchers have often applied attack techniques specific to the image domain on network datasets without considering the intrinsic differences between network and image data features. In image domain features are single pixel which does not have independent meaning while in network data we have features such as number of packets, size of the packets, etc. Proposing new methods for adversarial attacks that take into account the characteristics of network data and preserve the integrity of network attacks would

contribute to a more realistic and feasible approach to adversarial attacks in the network security domain.

1.2 Motivations

The motivation for this thesis stems from the widespread utilization of deep learning models in cybersecurity, particularly in tasks related to malware and intrusion detection. The security of these models holds immense significance, as mishandling or neglecting their protection can lead to severe consequences and substantial financial losses. While there has been considerable interest in investigating the security of deep learning models in various domains, there is a pressing need for further attention dedicated to network security-related tasks. Consequently, the primary motivation of this thesis is to explore the deep learning models' security in the network domain and propose attack techniques specific to this domain. By conducting this research, we aim to contribute to the existing knowledge and improve the overall security of deep learning models in network security. Our findings will be valuable for cybersecurity professionals, researchers, and practitioners in developing strong defenses and protecting critical network infrastructure.

1.3 Contributions

This research focuses on exploring poisoning and evasion attacks in network intrusion detection systems. The main contributions of this thesis could be summarized as follows:

1. Proposing a comprehensive framework for adversarial attacks on deep learning-based NIDS. The framework comprises two main components: a poisoning attack and an evasion attack. The poisoning attack introduces a label-flipping

attack, while the evasion attack includes two methods based on Fast Gradient Sign Method (FGSM) and saliency maps.

2. Developing a label-flipping attack using stratified sampling and random label-flipping as the poisoning attack. This attack selects various percentages of the original training set and flips the labels of the selected samples, thereby poisoning the training set and reducing the target model's performance.

3. Developing an FGSM-based evasion attack that categorizes the extracted network features into six categories based on their characteristics, including *Forward, Backward, Flow-based, Time-based, Packet Header-based*, and *Packet Payload-based*. The attack modifies each of these feature categories to generate adversarial examples.

4. Developing an evasion attack based on the saliency map. This attack calculates the saliency map for different combinations of input features, considering varying numbers of members within the combinations. It identifies the combination that has the highest impact on the target model's decision and uses it to generate the adversarial samples.

5. Proposing a novel evaluation model based on meticulously selected criteria to evaluate the proposed saliency map-based evasion attack effectively. The selected criteria are the effect of the attack method's parameters, the analysis of the distance between the original and adversarial sample, the effect of the adversarial attack on the performance of the DNN model, best feature sets success rates, average confidence of adversarial class, and adversarial samples transferability.

6. Evaluating the proposed attacks using three widely recognized datasets, namely CIC-IDS2017, CIC-IDS2018, and CIC-UNSW. The CIC-UNSW dataset is

generated by performing feature extraction and labeling tasks on the original pcap files of the UNSW-NB15 using the CICFlowMeter. Importantly, this dataset contains the same set of features as the other two datasets, enabling a fair and consistent evaluation of the proposed attacks.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

- *Chapter 2:* In this chapter, the background of network intrusion detection systems (NIDS) is presented, along with an overview of various techniques used for implementing and improving these systems, including machine learning and deep learning. The history of deep learning is discussed, and some well-known deep neural networks utilized in the network intrusion detection domain are introduced. The chapter then delves into the attacks that pose threats to the privacy and security of deep learning models, with a specific focus on poisoning and evasion attacks. Furthermore, related works exploring poisoning and evasion attacks in the network domain are presented.

- *Chapter 3:* In this chapter, the thesis outlines the proposed framework, comprising two main components: Poisoning attacks and Evasion attacks. A label flipping-based attack is proposed for the poisoning attack component, while two FGSM-based and saliency map-based attacks are introduced for the evasion attack component. The chapter provides a detailed explanation of the proposed evaluation model tailored for the evasion attack.

- *Chapter 4:* Three datasets, including CIC-IDS2017, CIC-IDS2018, and CIC-UNSW are used to evaluate and analyze the proposed method. The description of these datasets and detailed information about the implementation are provided in this chapter.

- *Chapter 5:* This chapter presents the results and discussion of the experimental analysis for the three proposed attacks and the combined poisoning and evasion attacks. The proposed evaluation model is utilized to assess the outcomes of the saliency map-based attack.

- *Chapter 6:* The thesis concludes in this chapter, presenting the final findings derived from the research. Additionally, the chapter outlines possible directions for future work in this area of study.

Chapter 2

Background and Literature Review

In this section, first, a brief introduction to network intrusion detection systems is provided. Then we review the basic concepts of the deep neural network and common attacks against machine learning and deep learning models. After that, the works focusing on poisoning and adversarial cyber attacks are reviewed.

2.1 Network Intrusion Detection Systems (NIDS)

With the rapid expansion of computer networks and the increasing reliance on digital communication, the security of these networks has become a critical concern. Network intrusion detection systems (NIDS) safeguard computer networks against unauthorized access, malicious activities, and potential cyber threats. NIDS has evolved as a fundamental component of modern network security architectures, providing organizations with a proactive approach to identifying and responding to network attacks [83].

Malicious actors constantly seek vulnerabilities to exploit within networks, posing significant risks to individuals, organizations, and nations. To mitigate these risks, the concept of intrusion detection emerged. Intrusion detection aims to identify and respond to unauthorized activities within a computer network [69]. Traditional

security measures, such as firewalls and antivirus software, focus on preventing unauthorized access and detecting known threats. However, they may not be sufficient to protect against sophisticated attacks or zero-day vulnerabilities. This is where network intrusion detection systems play a crucial role.

Network intrusion detection systems are designed to monitor network traffic, analyze data packets, and detect suspicious or malicious activities [53]. These systems operate on the principle of analyzing network traffic patterns and comparing them against known attack signatures or abnormal behavior. The NIDS is placed at the network edge and monitors incoming or outgoing traffic. If there is any abnormal or malicious behavior, the NIDS raises the alarm. NIDS can operate in either a passive or active mode.

In passive mode, NIDS analyzes network traffic without interfering with the data flow. They monitor packets, extract relevant information, and generate alerts or log entries when potentially malicious activity is detected. In contrast, active NIDS can immediately block or mitigate attacks by modifying network configurations or sending alerts to administrators for further investigation [1].

Two main categories of NIDS based on detection techniques are signature-based and anomaly-based [47]. Signature-based NIDS, or misuse detection systems, rely on a database of known attack signatures to identify and match patterns within network traffic. They compare incoming traffic against a set of predefined signatures to determine if an attack is in progress. While signature-based NIDS are effective against known attacks and have a low false positive detection rate, they may struggle to detect new or zero-day threats [25, 93].

Anomaly-based NIDS, on the other hand, focuses on establishing a baseline of normal network behavior and flagging any deviations from this baseline as potential intrusions. These systems use machine learning algorithms and statistical analysis to identify anomalies in network traffic [76, 104]. Anomaly-based NIDS can be more

effective in detecting previously unseen attacks or zero-day vulnerabilities but can also generate more false positives, which means the possibility of classifying legitimate behavior as malicious. One of the most used methods in anomaly-based NIDS is deep learning models because they can learn and extract usable features from a high volume of data. The deep learning models applied in NIDS are usually deployed in the system's background, so the potential adversary may only be able to obtain the classification results and fail to access the internal information [129].

Over the past several years, many papers have been published using ML and DL algorithms in NIDS. In [34], they developed an IDS using an adaptive ensemble learning model and evaluated it using the NSL-KDD [111] dataset. They used four algorithms: Decision Tree, Random Forest, K-Nearest Neighbor, and Deep Neural Networks. Karatas *et al.* [54] compared several ML algorithms' performance, including Decision Tree, Random Forest, and KNN using the CIC-IDS2018 dataset. The authors in [35] used one-class and two-class SVMs with linear and non-linear kernels to build an IDS and compared the results with an unsupervised anomaly-based IDS. Yan *et al.* used a stacked sparse autoencoder (SSAE) to extract high-level features with SVM as the classifier in their proposed model [128]. In [130], a multilevel intrusion detection model framework is proposed to address issues such as the imbalance of network traffic in different categories and the nonidentical distribution between the training set and the test set.

Tang *et al.* [110] used a Deep Learning Model for flow-based anomaly detection in a Software Define Network environment and tested it on the NSL-KDD dataset. Six features from NSL-KDD that can be easily extracted for an SDN environment are selected to train the DNN model. Kim *et al.* [55] used a Long Short-term Memory (LSTM) network to classify network attacks in the KDD dataset. In [29], they used a DNN model to detect DoS attacks and DNN, CNN, and LSTM to detect XSS and SQL attacks in ad-hoc networks. KDD Cup 99 dataset and a DNN with four

hidden layers with 100 neurons in each layer were used to build an IDS model in [56]. An improved DBN is used to detect attacks in the NSL-KDD dataset [67]. They enhance the learning process of the DBN by applying an Extreme Learning Machine (ELM). Lopez-Martin *et al.* [68] used deep neural networks and RBF neural networks for classification tasks on CIC-IDS2017 [101] and CIC-DDoS2019 [102]. Potluri *et al.* [94] utilize an accelerated Deep Neural Network (DNN) architecture with three hidden layers to identify abnormalities in network data. They enhance the effectiveness of the DNN architecture by pre-training it using an Autoencoder (AE). Yin *et al.* [131] employ a Recurrent Neural Network (RNN) model to do the classification task in both binary and multi-class settings using the NSL-KDD dataset. Farahnakian *et al.* [28] propose a Deep Learning-based approach for intrusion detection. Their proposed technique includes a 4-layer AutoEncoder and a softmax classifier.

2.2 Deep Learning

Several complex real-world problems are difficult to model mathematically and solve even with the significant capabilities of computers. However, the human brain can easily understand these problems, such as speech recognition, image classification, and understanding natural languages. Researchers have long tried to model the human brain's behavior to solve these problems. Finally, in 1985, Frank Rosenblatt introduced perceptron as an artificial model of the human brain's biological neuron [98, 99]. Perceptron can only classify the linearly separable data and cannot solve some simple problems like XOR [75].

Multi-layer Perceptron (MLP) [48], which is the basis of all DNNs, is constructed from layers of perceptron neurons. MLP aims to find a non-linear mapping from its input layer features to its respective output. Each neuron is a simple function followed by an activation function such as Sigmoid, TanH, and ReLU. The neuron's

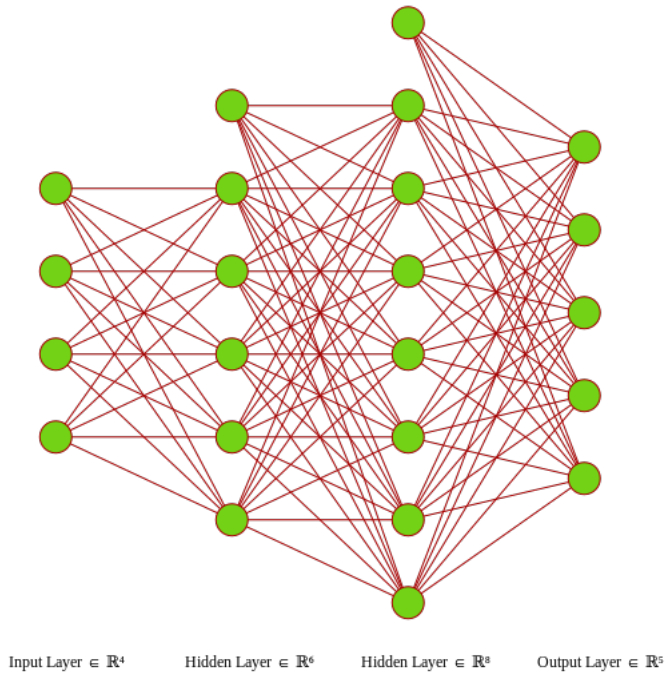


Figure 2.1: A Multi-layer Perceptron.

role is to compute the weighted sum of its parameters, which are the previous layer's outputs, and then generate the final outputs of its layer by using the activation function. The output of a neuron in layer j is calculated using the following formula:

$$y_j = f_i(\sum_{i=1}^{n} \theta_{ji} x_i + b_j), \tag{2.1}$$

where θ_j is the layer j weight matrix, b_j is the bias, and $f(.)$ is the activation function. There are three different layers in an MLP: the input layer, the hidden layer, and the output layer. Each MLP consists of one input layer, one output layer, and several numbers of hidden layers based on the respective task. Figure 2.1 shows the architecture of an MLP with two hidden layers.

An MLP model can be considered as the following function F:

$$F(\vec{x}) = f_n(\theta_n, f_{n-1}(\theta_{n-1}, ...f_2(\theta_2, f_1(\theta_1, \vec{x})))), \tag{2.2}$$

where θ_i is the weight vector of layer i, f_i is the activation function used in layer i, and $\vec{x}$ is the input vector.

Using a DNN consists of two phases: training and testing. In the training phase, we feed a large set of input-output pairs $(\vec{x}, \vec{y})$ to the DNN model. Then the model adjusts its weight parameters by minimizing the error between the predicted value for the output and the correct output $\vec{y}$. The commonly used technique for adjusting the DNN parameters is the backpropagation algorithm [26, 124] with optimization methods such as Stochastic Gradient Descent (SGD), and Adam optimizer [57]. The DNN will predict the output for unseen inputs during the test phase, using the weight parameters calculated during the training phase.

2.2.1 Deep Neural Networks (DNN)

There are several advanced deep learning models such as Convolutional Neural Networks (CNN) [63], Recurrent Neural Networks (RNN) [12, 45], Stacked AutoEncoder (SAE) [11, 96], and Deep Belief Networks (DBN) [43] with a wide range of applications [62] including natural language processing [23], bioinformatics [74], computer vision [119, 125], image recognition [106], speech recognition [85], and information retrieval [132]. In the following subsections, some of the famous DNN architectures that have considerable performance in complex problems are introduced.

2.2.1.1 Convolutional Neural Network (CNN)

CNNs are a category of deep learning models that are usually applied to image-related tasks, object detection and recognition, image segmentation, and image classification [62]. Like any DNN, CNN consists of an input layer, an output, and hidden layers, which may contain one or more convolutional, pooling, and fully connected layers. Figure 2.2 shows a CNN architecture that can be used in a classification problem.

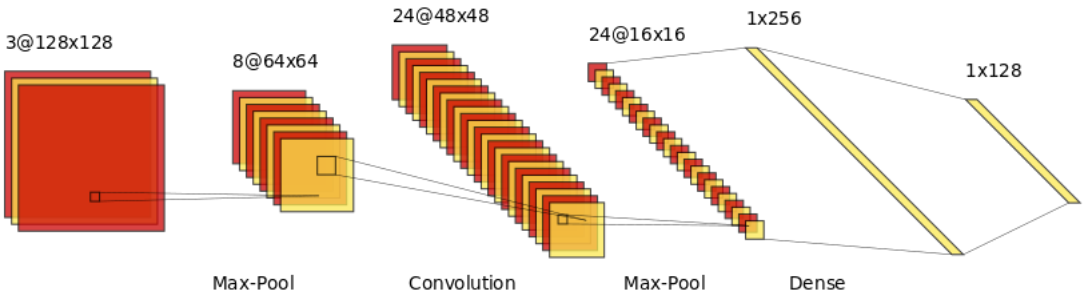


Figure 2.2: A Convolutional Neural Network architecture.

The raw pixel values of an image are fed to the input layer. In the convolutional layer, the dot product of the filter and a local region of the input image is calculated. These local regions are called receptive fields, and the filter slides across all the regions to represent the whole image. The sliding size of the filter is called the stride window. After each convolutional layer, a pooling layer is placed to reduce the dimension of the data and CNN computation costs [21]. Max Pooling and Average Pooling are the two most used pooling functions. After several convolutional and pooling layers, a fully connected layer performs the classification task based on the extracted features. A fully connected layer is the same as an MLP; the same function calculates its activation.

To capture the non-linearity of the input image, an activation function such as ReLU is placed after each convolutional layer [42]. In contrast, a softmax activation function is usually used after the fully connected layer to calculate a probability between 0 to 1 to complete the classification task.

2.2.1.2 Recurrent Neural Networks (RNN)

In traditional neural networks, there is an assumption that all the inputs and outputs are independent. This assumption is invalid in many tasks, such as natural language processing and speech recognition. RNN benefits from having a memory of the previously calculated information and can produce an output dependent on

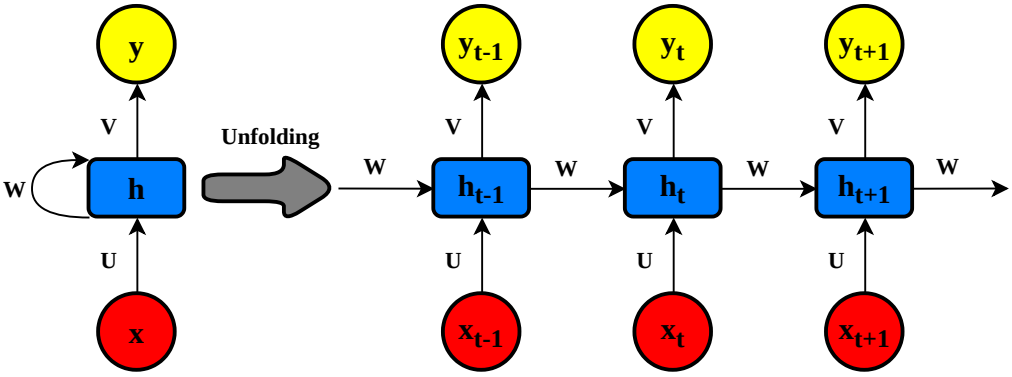


Figure 2.3: Compressed and unfolded basic RNN.

the previously computed state. Figure 2.3 shows the compressed and unfolded basic RNN.

Let x_t and h_t be the input and hidden state at the timestamp t, respectively. The output y_t at timestamp t is defined as:

$$y_t = softmax(Vh_t), \tag{2.3}$$

where V is the weight matrix of the output layer. The parameter h_t is assumed to be the network memory and is calculated based on the previous hidden state and the input at the current step.

$$h_t = f(Ux_t + Wh_{t-1}). \tag{2.4}$$

U and W are weight matrices for the input layer and hidden state. The activation function f usually is a non-linearity, such as sigmoid, tanh, or ReLU [71, 73].

One of the main problems of the traditional RNN is the vanishing or exploding training weight. As explained before, RNN uses the information from the previous state in its decision, and it can't remember information for a very long time because the backpropagation gradients increase or decrease at each time step. In [46], they introduced Long Short-Term Memory (LSTM) unit to overcome this problem. Figure 2.4 shows the architecture of an LSTM cell. An LSTM cell includes four different gates:

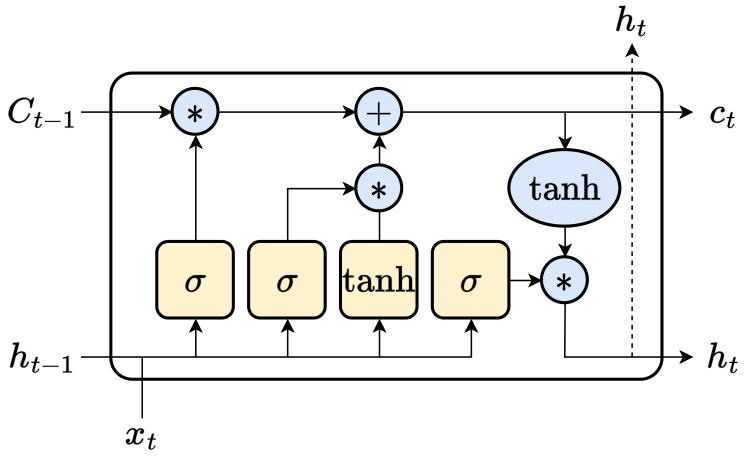


Figure 2.4: LSTM cell.

input gate, output gate, forget gate, and update gate. The input gate decides what to accept into the cell, the forget gate controls what to forget from previous memory units, the update gate updates the memory, and the output gate calculates the cell output.

2.2.1.3 Stacked AutoEncoder (SAE)

Stacked AutoEncoders consist of several AutoEncoders stacked on top of each other where the output of each layer is connected to the input of the next layer [118]. Each AutoEncoder is trained separately by minimizing the reconstruction error of its input, which is the output of its previous layer. After training all the AEs, a softmax layer is added as the last layer, and the network is fine-tuned for the desired classification task. Using a chain of encoders can help compress the data and extract more high-level features that represent the coded version of the input [16].

An AutoEncoder employs recognition weights to transform an input vector x into a representation vector h, known as the latent vector. Then it uses generative weights to decode the latent vector and produce a reconstruction of the input vector x' [10]. The objective of the AutoEncoder is to unsupervisedly reconstruct the input data without relying on any labels while ensuring that the input and output dimensions

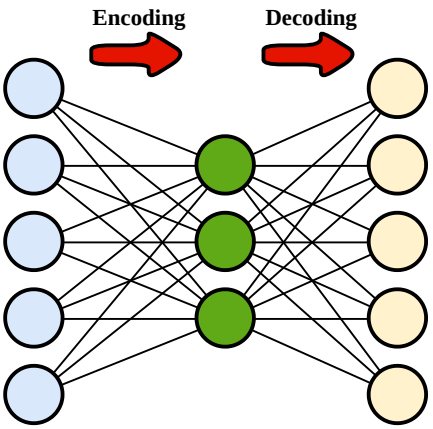


Figure 2.5: AutoEncoder.

remain identical.

Figure 2.5 shows the architecture of an AE. The following functions are used to do the encoding and decoding:

$$h = f(Wx + b), \tag{2.5}$$

$$x' = f'(W'h + b'), \tag{2.6}$$

where f and f' are activation functions, W and W' are weight matrices, and b and b' are bias matrices.

2.3 Attack against Machine Learning and Deep Learning Models

As we mentioned before, ML and DL models are vulnerable to a range of different attacks which can threaten their privacy and security. Some of these attacks are membership inference attack [44, 66, 100, 103, 116], model inversion attack [30, 31], model extraction attack [114], hyper-parameter stealing attack [86, 120], property inference attack [7, 32, 72], poisoning attack [113], and evasion attack (adversarial

attack) [18, 38, 64, 90, 89, 127].

- **Membership Inference Attack**

  The attacker's goal here is to access the private data used for the training. They are trying to predict if a specific record was part of the training set and, therefore, violated the training data's privacy. Shokri *et al.* [103] were the first to perform a membership inference attack against deep learning models. They designed a shadow training method to make the membership inference. They created several shadow models which imitate the target model's behavior. In [100], Salem *et al.* improved the membership inference attack by relaxing previous assumptions about the adversary's information. Through extensive experiments, they introduced three different adversaries and demonstrated that relaxing those assumptions does not significantly change membership inference attack performance.

- **Model Inversion Attack**

  In a model inversion attack, an adversary with access to a machine learning model confidence values tries to infer sensitive features of the training samples. Fredrikson *et al.* [30] performed the model inversion attack on a machine learning as a service model (MLaaSS) performing a face recognition task. They did their attack in a black-box setting and recovered the original image from the training set. In [31], they also made a model inversion attack in a biomedical setting.

- **Model Extraction Attack**

  Training sophisticated machine learning models requires a lot of time and computation power, making them more expensive and valuable. The model extraction attack aims to recover the model's parameters via black-box access. Tramer *et al.* [114] performed their attack against a Logistic Regression and a

Multi-layer Perceptron in binary and multi-class classification tasks.

- **Hyper-parameter Stealing Attack**

  A variant of model extraction attack is hyper-parameter stealing, which aims to know the hyper-parameters used during the training phase by the target machine learning model. The hyper-parameter stealing attack is not limited to neural networks; it can also be employed against other machine learning algorithms such as logistic regression and support vector machines [120].

- **Property Inference Attack**

  The attacker's goal is to acquire certain properties of the target model's training data. In [32], they perform their attack against a fully connected neural network to infer some global properties of the training dataset, such as the environment in which the data was produced or the fraction of the data that comes from a certain class.

Among the attacks mentioned above in this thesis, our focus is on poisoning and evasion attacks, and we propose a framework using these attacks against NIDS. In the following subsections, we will explore these attacks in more detail.

2.3.1 Poisoning Attack

In a poisoning attack, the attacker's goal is to decrease the machine learning model performance by interfering with the training phase [122]. In this attack, the main idea is to inject malicious samples into the training set of the target model [8, 9]. Each attacker may have different capabilities, and based on these capabilities, the poisoning attacks can be categorized into four groups:

- **Label Manipulation.** Here, the attacker only has access to the training data labels and can modify the label of any records in the training set.

- **Data Injection.** In this case, the attacker can inject malicious records into the training set.
- **Data Modification.** With this capability attacker can modify some of the records present in the training set.
- **Model Tampering.** Attackers with this capability can tamper with the training model parameters and algorithm and damage the output trained model.

In the training phase, the machine learning model learns from the sample-label pairs in the training set and predicts based on the learned knowledge. Suppose the samples in the training set are not paired with the correct label. In that case, it can significantly decrease the trained model's performance [112]. A simple way to do label manipulation is by flipping the labels of the training samples. In [13], Biggio *et al.* performed a label-flipping attack against an SVM classifier with two strategies: random label flips and adversarial label flips. In the random label flips strategy, they randomly selected a percentage of the training data and flipped their labels. But, in the adversarial label flips strategy, instead of randomly selecting the sample for label flipping, they tried to choose the samples that maximize the classification error of the test set. Zheng *et al.* [133], performed a label-flipping attack against famous deep learning models such as Inception and Alexnet on the CIFAR-10 dataset. The deep learning models were forced to learn the poisoned data and gained acceptable training errors, but the test error was close to zero. In [126], they also made a label-flipping attack against an SVM classifier. They used three strategies to find the combination of the flipped label, which led to the highest classification error. These strategies are randomly flipping the labels, selecting samples nearest to the decision hyper-plane first, and selecting samples furthest to the decision hyper-plane first.
Performing the label manipulation attack is straightforward and can easily be done by changing the labels of a subset of the training set. This simplicity prevents the

attacker from performing more complex and sophisticated attacks and causes attacks to be easily noticeable to the victim. Therefore, making the poisoning attack through data manipulation could be more effective.

Data manipulation can be performed by injecting poisoned samples into the training set or modifying some existing samples. The adversary can achieve data manipulation by using two major algorithms: optimization-based and training-based.

In optimization-based algorithms, the attacker aims to affect the training process by introducing malicious samples by optimizing an objective function. Different objective functions can be designed based on the target model learning algorithm. In [14], a gradient ascent technique is used to optimize the objective function for poisoning attacks against SVM. The goal is to generate a poisoning sample that can increase test error. The experiments proved that their method could find the local optimum point effectively. Similarly, in [52], they used a bi-level optimization to generate poisoning samples against a linear regression model. They attack white-box and black-box settings using four different regularization terms for the linear regression model. Due to the complexity of techniques used in the training phase of deep learning models, using the previously mentioned optimization techniques is difficult. The authors in [84] used back-gradient optimization to overcome this complexity and perform a poisoning attack against deep-learning models.

In the training-based attack, the attacker either uses a shadow model to imitate the target model and perform the poisoning attack against it or incorporates generative models to create poisoned samples. Suya *et al.* proposed a targeted poisoning attack where they generated an intermediate model using a heuristic [108]. They created a highly effective poisoning dataset by iteratively adding a poisoning sample to the training dataset and maximizing the loss difference between the intermediate model and the target model. Li *et al.* proposed homograph and dynamic sentence attacks in NLP [65]. The homograph attack inserts homograph replacement triggers, but

it's easily detected. To overcome this, they devised a more subtle poisoning attack using context-aware suffix sentences generated by language models. They trained an auxiliary model on a relevant corpus to hide the trigger.

2.3.2 Evasion Attack (Adversarial Attack)

The primary purpose of adversarial attacks is to create inputs that can fool different machine learning techniques and force them to make wrong decisions. These crafted inputs are called adversarial examples. These examples are carefully crafted by adding small, often imperceptible perturbations to legitimate inputs to fool deep learning models into making wrong decisions. Simultaneously, a human observer can correctly classify these examples [38, 89].

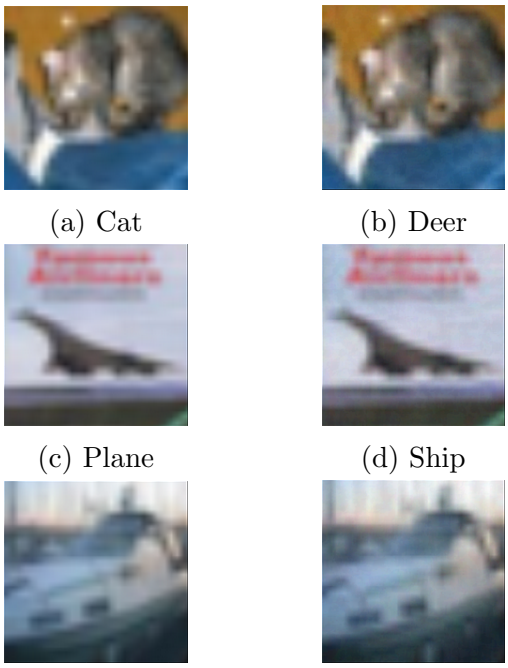
Figure 2.6 shows the original inputs and the adversarial example of those inputs. As it is seen, each image pair look the same to a human observer. Still, the images on the right, the adversarial examples, will force a particular image classification deep neural network to make a mistake and classify them falsely.

2.3.2.1 Adversarial Attack Threat Model

The adversarial attack threat model may be considered based on the attacker's knowledge, capabilities, goals, and used distance metrics.

- **Attacker's Knowledge**. Concerning the attacker's knowledge, the attacker may know all the information about the learned model, such as the learning algorithm, model architecture, parameters, hyper-parameters, and training data. This setting is called *White-box*. In contrast, in *Black-box* attacks, the attacker has limited information about the model and only knows the output.

- **Attacker's Goals**. Attackers may have different goals in an adversarial setting. In *Targeted* attacks, the goal is to make the model misclassify the input



(a) Cat (b) Deer

(c) Plane (d) Ship

(e) Ship (f) Car

Figure 2.6: Original and Adversarial Sample using FGSM

into a specific class. In contrast, in *Untargeted* attacks, the attacker only aims to fool the model and does not care about the output class. The other type of attack is *Confidence Reduction* or *Reliability* attacks. In this case, the attacker only seeks to reduce the confidence score of the machine learning model without necessarily changing the output class.

- **Distance Metrics**. As we said in the adversarial examples, these artificially crafted samples should be similar to the original input. Attack algorithms use different distance metrics to measure the similarity between an adversarial example and its original input. The three most common distance metrics are:
  - l_0 distance, which counts the number of perturbed features.
  - l_2 distance computes the standard Euclidean distance between the original and adversarial samples.
  - l_∞ distance measures the maximum difference between the original and

adversarial samples' features.

2.3.2.2 Adversarial Attack Algorithms

As we mentioned before, Szegedy *et al.* [109] for the first time demonstrated that there are small perturbations that can be added to an image and force a deep learning classifier into misclassification. Let f be the DNN classifier, and loss f be its associated loss function. For an image x and the target label l, to find the minimal perturbation r, they proposed the following optimization problem:

$$min \ \|r\|_2 \ s.t. \ f(x+r) = l; \ x + r \in [0, 1]. \tag{2.7}$$

Since this is a challenging problem, they used a box-constrained L-BFGS to find an approximate solution. They found the perturbation needed to add to the original image to create an adversarial example by solving this problem.

To make it easier to craft an adversarial example, Goodfellow *et al.* [38] proposed a fast and simple method for generating adversarial examples. They called their method the Fast Gradient Sign Method (FGSM). They used the model gradient's direction to calculate the perturbation they wanted to add to the original example. They used the following equation:

$$\eta = \epsilon sign(\bigtriangledown_x J(\theta, x, l)), \tag{2.8}$$

where η is the perturbation, ϵ is the magnitude of the perturbation, and l is the target label. This perturbation can be computed efficiently using backpropagation. This method calculates the perturbation in only one step toward the direction of the gradient of the model. Since this method only uses one step, it may not find the minimum required perturbation to fool the model.

Kurakin *et al.* [60] proposed a new method based on the FGSM, and instead of

applying it in one step, they applied the method multiple times with a smaller step size. Also, after each step, they clipped the results to prevent any large changes on each pixel:

$$x_{n+1} = Clip_{x,\epsilon}\left\{x_n + \alpha sign(\bigtriangledown_x J(\theta, x, l)\right\}. \tag{2.9}$$

Their experiments showed that the iterative gradient sign could produce better results than the fast gradient sign.

Papernot *et al.* [90] designed a new method based on adversarial saliency maps to craft adversarial examples. Their main idea was to craft adversarial examples based on the direct mapping between inputs and outputs of DNNs. They wanted to model the relation between perturbations added to the input and DNN output variations. To understand this relation, they used the forward derivative of the DNN. They defined the forward derivative as the Jacobian matrix of the function F learned by the DNN. If the DNN has M input and N output, the forward derivative of a sample X is:

$$J_F(X) = \frac{\partial F(X)}{\partial X} = \left[\frac{\partial F_j(X)}{\partial x_i}\right]_{i\in 1..M, j\in 1..N}. \tag{2.10}$$

Then they used these computed forward derivatives to create the adversarial saliency maps. Using these maps, they said we could find which input features we should add perturbations to have our desired changes in output most efficiently.

JSMA aims to add more minor perturbations to a small subset of features to cause the model to make a mistake compared to FGSM. However, JSMA is much slower than FGSM due to its high computational cost. Other iterative-based methods such as CW attack [18] and Deepfool [80] are also proposed. CW aims to find the perturbation with minimum l_2 distance while maximizing the loss function. They formulated the optimization problem as follows:

$$minimize \, \|\delta\|_2 + c.f(x+\delta)), \quad such \; that \; x+\delta \in [0,1]^n. \tag{2.11}$$

They defined f as:

$$f(x') = \max\left(\max_{0\leq i\leq n}\{Z(x')_i : i \neq t\} - Z(x')_t, -k\right), \tag{2.12}$$

where Z is the logits, t is the target class, and k is the parameter controlling the model's confidence.

Another interesting algorithm is the one-pixel attack. The idea of a one-pixel attack is, instead of adding a small value of perturbations to several pixels to create an adversarial example, just change the value of one (three or five) pixels as much as needed [107]. Su *et al.* proposed a method based on Differential Evolution (DE), a population-based optimization algorithm to choose the best pixel and perturbation value for generating an adversarial example. They said using DE for generating adversarial examples has several advantages, such as a higher probability of finding the global optima, requiring less information from the target system, and simplicity [107].

2.4 Poisoning Attack and Adversarial Attack in Cybersecurity

The early research focused on adversarial attacks, mainly on image domain problems. Still, with the increasing usage of DNN in security problems, the researchers realize that adversarial examples may widely exist in this domain. Grosse *et al.* [39] have studied adversarial examples in malware detection. They used the adversarial attack based on the Jacobian matrix. They have limited the number of changed features to 20 to ensure the modifications do not change the application drastically. They have achieved misclassification rates between 63% to 69%. In [123], the authors did a

straightforward experiment on the NSL-KDD dataset. They used the FGSM method to generate adversarial examples and showed the possibility of adversarial attacks in NIDSs. In [97], Rigaki performed an adversarial attack against the deep learning model used in NIDS, using FGSM and JSMA methods, and showed how significantly they can reduce the model's accuracy. Wang did an adversarial attack against the NSL-KDD dataset using four famous methods, including FGSM, JSMA, Deepfool, and C&W [121]. In his thorough study, he also analyzed the effect of different features in the dataset in the adversarial example generation process. Peng *et al.* [91] trained four different machine learning-based intrusion detection systems with DNN, SVM, Random Forest, and Logistic Regression and studied these models' robustness in adversarial settings. Ibitoye investigated the adversarial attacks against deep learning-based intrusion detection in IoT networks [51]. In [40], they showed how to evaluate an anomaly-based NIDS trained on network traffic in the face of adversarial inputs. They explained their attack method by categorizing network features and evaluated three recently proposed NIDS. Hashemi *et al.* presented a new technique for anomaly-based NIDS based on denoising autoencoders to increase the system's robustness against adversarial samples [41]. They trained their model on some parts of the input, and in this way, they increased the reconstruction error of the abnormal traffic. Alhajjar *et al.* in [3]. used two evolutionary algorithms, particle swarm optimization (PSO) and genetic algorithm (GA), and generative adversarial networks (GAN) to generate adversarial examples for NSL-KDD and UNSW datasets. Their results showed that the average evasion rate for NSL-KDD was more than 57%, and for UNSW was more than 49%. In [22], the authors performed an adversarial attack using FGSM, JSMA, C&W, and ENM against the Kitsune network intrusion detection system. Their experiments showed that Kitsune is more vulnerable to integrity attacks than availability attacks.

Adversarial attacks are performed in white-box or black-box settings. In the white-

box setting, the adversary has the perfect knowledge of the target model, including the model's used technique, architecture, and hyper-parameters. In contrast, in black-box settings, the attacker can only provide input and receive the model's output and does not have any information about the model's characteristics. First, we reviewed the related works in white-box settings, and in the following, we continued with the black-box setting. Yang *et al.* [129] made a black-box attack on the NSL-KDD dataset. They trained the DNN model on the dataset and used three attacks based on a substitute model, ZOO [19], and GAN [37]. In [59], they proposed a novel black-box attack that generates adversarial examples using spherical local subspaces. They evaluated their attack against seven state-of-the-art anomaly detectors. The authors in [5] evaluated the adversarial attacks against botnet detectors. They randomly changed four selected features of each network flow to achieve their adversarial goal. They evaluated botnet samples from four famous datasets and considered feature removal techniques as a defense against adversarial attacks. Huang *et al.* in [50] proposed two black-boxed methods based on a genetic algorithm and a packet saliency for generating adversarial examples against LSTM-based DDoS detection models. In [95], a black-boxed adversarial attack against NIDS in IoT systems was also introduced. They create a duplicate local model and use this model for their attack. They rank input features based on their saliency and use iterative FGSM for their attack. They changed the selected feature in each step until they fooled the model.

Most research on adversarial attacks in the network domain focuses on evasion attacks. Few works focus on adversarial attacks during the training phase for machine learning-based network intrusion systems. In [4], Apruzzese *et al.* made a label-flipping attack against a botnet dataset. In their attack, they selected existing malicious samples and slightly changed the value of three features, including *duration*, *exchanged_bytes*, and *total_packets*. They then flipped the label to benign. They per-

formed the attack against Random Forest, Multi-layer Perceptron, and KNN. Their experiments show that the model performance drops significantly after the poisoning attack. Papadopoulos *et al.* made a label-flipping attack against SVM based model for the Bot-IoT dataset [87]. They sorted the samples based on their distance from the SVM hyperplane and flipped the one with a smaller margin to the hyperplane. Their experiments showed that random and targeted attacks severely affect the classification metrics. One of the main areas for improvement of the previous works is focusing on traditional machine-learning techniques for performing adversarial deep-learning attacks. Furthermore, in most cases, researchers used simple and outdated datasets for experimental analysis and evaluation of their proposed models.

2.5 Concluding Remarks

In this chapter, first, we talked about the network intrusion detection system, different techniques used to improve them, and their evolution in the past few years. Then, we provided a brief background about deep learning and various deep neural networks used in NIDS. In the next section, we presented different attacks that can violate the privacy or security of the deep learning models. We explained poisoning and evasion attacks in more detail since they are the main focus of this research. Finally, we reviewed some existing work on poisoning and adversarial attacks in the network security domain and highlighted their challenges and shortcomings.

As mentioned before, the interest in attacks against deep learning models, specially evasion attacks started in the image processing and computer vision domain. Most of the proposed attack techniques were designed to target the models which work with image datasets. The main shortcoming of existing works in the network domain is transferring techniques that works for image data to network data without considering the differences between these two domains.

Some other challenges are the lack of techniques specific to network datasets that consider their essential characteristics, using outdated and simple datasets in their evaluation, and the lack of providing a sophisticated evaluation model with suitable criteria.

In the next chapter, we will present our proposed framework, including one poisoning attack and two evasion attacks. We also provide a novel evaluation model for adversarial attacks. Our goal is to overcome the challenges as mentioned above by proposing a new framework.

Chapter 3

Proposed Framework

In this chapter, we go over the proposed framework. In the following subsections, we will explain each component of the proposed framework in detail. Finally, we conclude this section by proposing an evaluation model based on several criteria for the proposed attacks.

3.1 Overview of the Proposed Framework

As mentioned before, this thesis focuses on poisoning and evasion attacks. Figure 3.1 shows the overall overview of the proposed framework, which consists of two main components: poisoning attack and adversarial attack. The adversarial attack uses the trained model and original samples from the dataset to generate adversarial examples for fooling the deep learning model. However, the poisoning attack happens before the training phase by changing the training set.

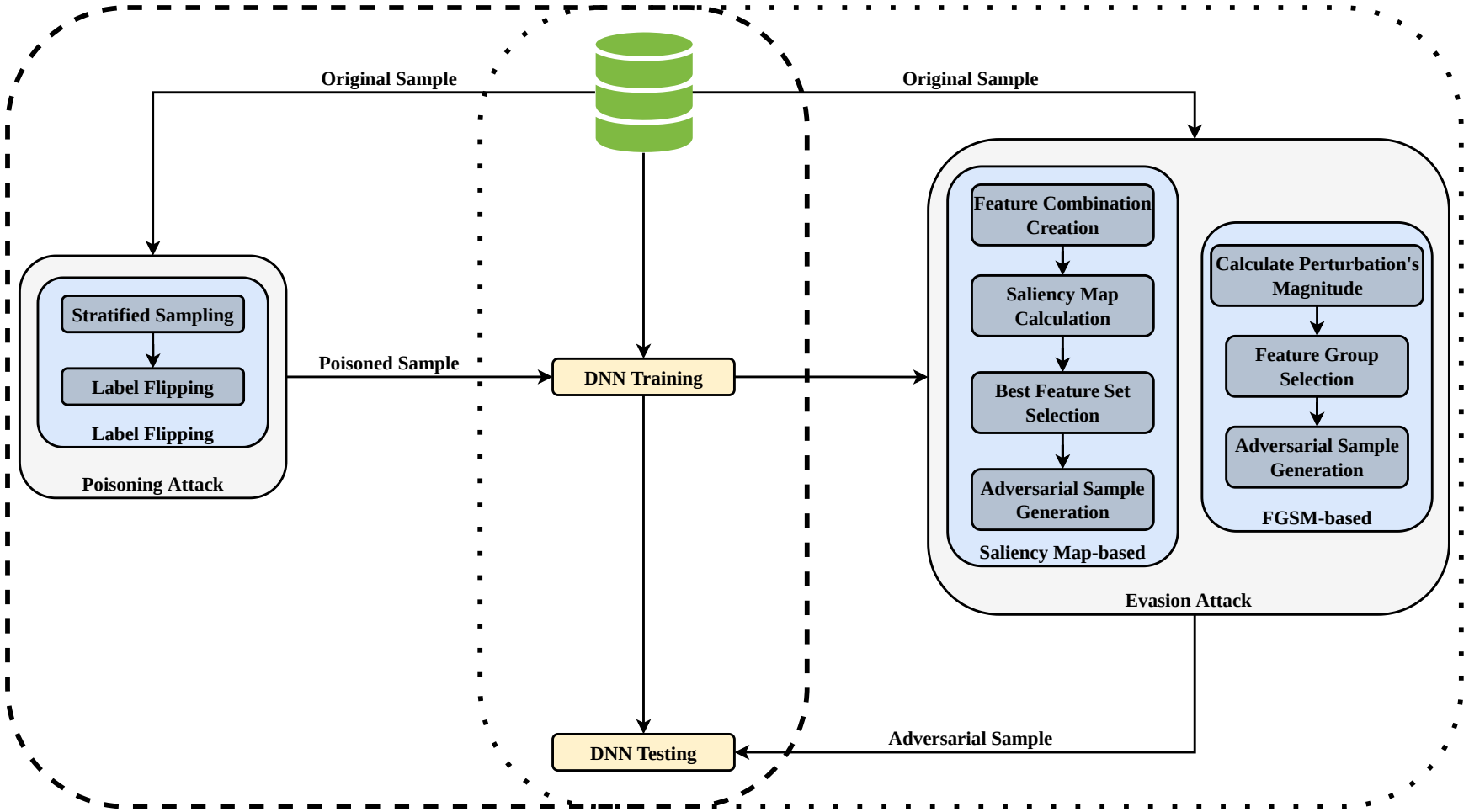


Figure 3.1: Overall overview of the proposed framework.

3.2 Proposed Label Flipping Attack

This section explains the proposed label-flipping attack for the poisoning attack component of the proposed framework [78]. In the label-flipping attack, the attacker's goal is to change the labels of the samples in the training set to decrease the performance of the deep learning model. Figure 3.2 shows the details of the proposed label-flipping attack, which includes two main steps. First, select the subset of training samples and then flip the labels of selected samples.

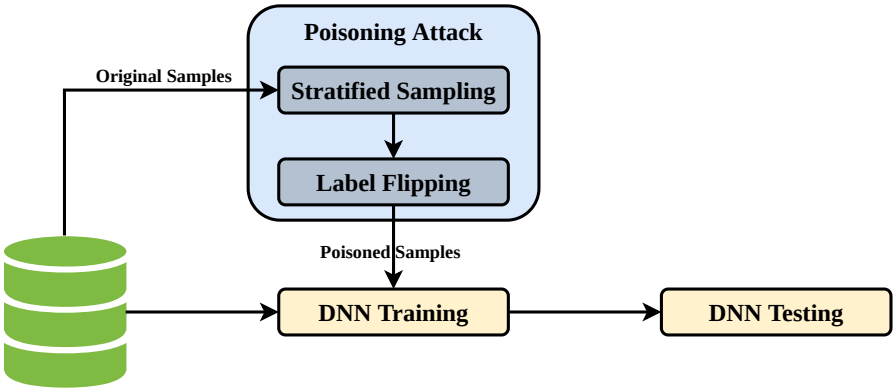


Figure 3.2: Proposed label flipping attack.

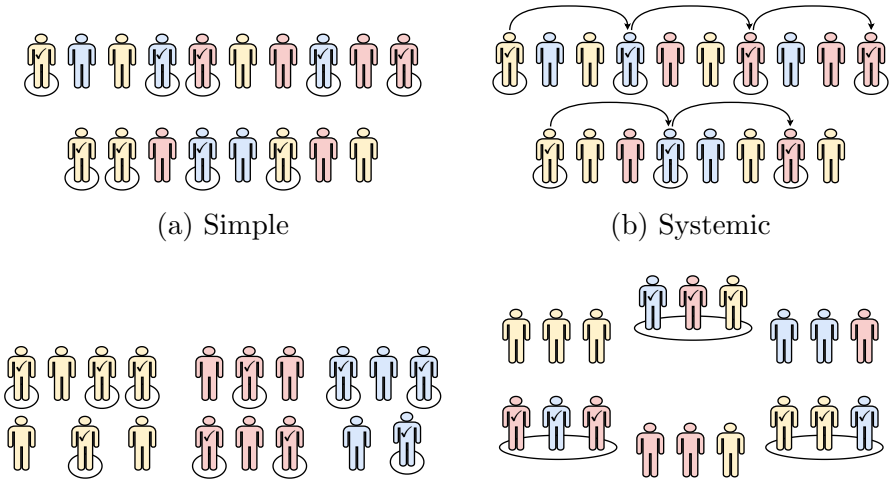
In probability sampling, each record in the dataset has the same possibility of being selected. There are four main types of probability sampling [70].

- **Simple Random Sampling** (Figure 3.3a)

  In this method, each record has an equal chance of being selected. The whole population is the sampling frame; each sample is selected with the same probability.

- **Systemic Sampling** (Figure 3.3b)

  In systemic sampling, samples are selected at regular intervals starting from a random start point. Since we specified the sample from a list with the same intervals, we should be careful that there is no hidden pattern in the provided list which may cause the sample to be skewed.



(a) Simple (b) Systemic

(c) Stratified (d) Cluster

Figure 3.3: Different types of probability sampling

- **Stratified Sampling** (Figure 3.3c)

  This method divides the population into subgroups (strata) based on their relevant characteristics. The number of subgroups are selected based on the number of distinct values of the target characteristics. The number of samples that should be selected from each subgroup is calculated based on the population proportion they include. Then, the samples are chosen using one simple or systemic sampling method.

- **Cluster Sampling** (Figure 3.3d)

  In cluster sampling, the population is divided into subgroups, but each subset should have the same characteristics as the whole sample. Then, an entire cluster is selected as the sample, or one of the other sampling methods is used to choose samples from the selected cluster.

Considering the necessity of obtaining samples from all the classes in the label-flipping attack, it appears that the application of stratified sampling is most suitable. We divide the training set samples into subgroups based on their class label and use simple random sampling to select samples from each subgroup. Then, we change the selected sample label to any label other than its actual label.

3.3 Proposed Adversarial Attacks

For the adversarial attack component, we will propose two techniques. First, a simple attack using FGSM and categorizing features based on their nature [79]. Then, our main proposed attack aims to generate adversarial samples changing the fewest possible features [77].

3.3.1 FGSM-based Attack

In this technique, we will perform the adversarial attack in a white-box setting and craft adversarial examples using different feature sets while using the FGSM [38] method for generating adversarial examples. Figure 3.4 includes the overview of the proposed FGSM-based attack.

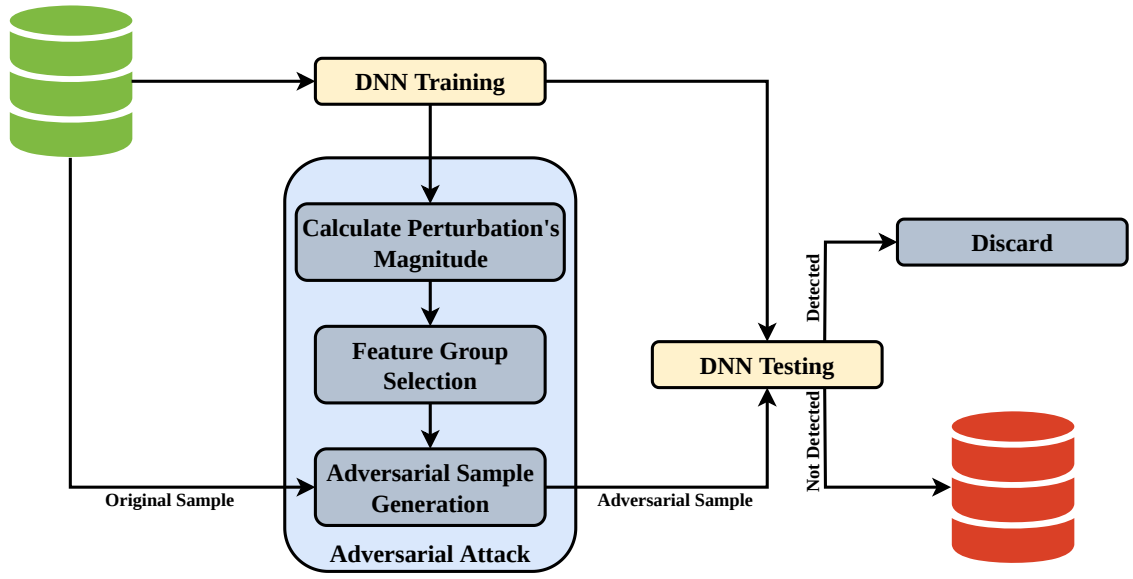


Figure 3.4: Proposed FGSM-based attack.

The datasets we will use in our experiments, like any network intrusion detection dataset, have several features that can be categorized based on their nature. We categorized these features into six groups: Forward Packet, Backward Packet, Flow-based, Time-based, Packet Header-based, and Packet Payload-based. The details of these feature sets are presented in Table 3.1.

Table 3.1: Feature sets.

Name of the feature set	List of features
Forward Packet (24)	total Fwd Packets, total Length of Fwd Packet, Fwd Packet Length Min Fwd Packet Length Max, Fwd Packet Length Mean, Fwd Packet Length Std Fwd IAT Min, Fwd IAT Max, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Total Fwd PSH flag, Fwd URG flag, Fwd Header Length, Fwd Packets/s Avg Fwd Segment Size, Fwd Avg Bytes/Bulk, Fwd AVG Packet/Bulk Fwd AVG Bulk Rate, Subflow Fwd Packets, Subflow Fwd Bytes Init_Win_bytes_forward, Fwd Act Data Pkts, min_seg_size_forward
Backward Packet (22)	total Bwd Packets, total Length of Bwd Packet, Bwd Packet Length Min Bwd Packet Length Max, Bwd Packet Length Mean, Bwd Packet Length Std Bwd IAT Min, Bwd IAT Max, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Total Bwd PSH flag, Bwd URG flag, Bwd Header Length, Bwd Packets/s Avg Bwd Segment Size, Bwd Avg Bytes/Bulk, Bwd AVG Packet/Bulk Bwd AVG Bulk Rate, Subflow Bwd Packets, Subflow Bwd Bytes Init_Win_bytes_backward

Flow-based (15)	Flow duration, Flow Byte/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std Flow IAT Max, Flow IAT Min, Active Min, Active Mean, Active Max Active Std, Idle Min, Idle Mean, Idle Max, Idle Std
Time-based (27)	Flow duration, Flow Byte/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max Flow IAT Min, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min Bwd IAT Min, Bwd IAT Max, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Total Fwd Packets/s, BWD Packets/s, Active Min, Active Mean, Active Max Active Std, Idle Min, Idle Mean, Idle Max, Idle Std
Packet Header-based (14)	Fwd PSH flag, Bwd PSH flag, Fwd URG flag, Bwd URG flag Fwd Header Length, Bwd Header Length, FIN Flag Count SYN Flag Count, RST Flag Count, PSH Flag Count ACK Flag Count, URG Flag Count, CWR Flag Count, ECE Flag Count
Packet Payload-based (16)	total Length of Fwd Packet, total Length of Bwd Packet, Fwd Packet Length Min Fwd Packet Length Max, Fwd Packet Length Mean, Fwd Packet Length Std Bwd Packet Length Min, Bwd Packet Length Max, Bwd Packet Length Mean Bwd Packet Length Std, Min Packet Length, Max Packet Length Packet Length Mean, Packet Length Std, Packet Length Variance, Average Packet Size

In the FGSM method, after computing the magnitude of the perturbation using Equation 2.8, the attacker will add the perturbation to all the input features to generate the adversarial example. But, since we only change a subset of input features to craft adversarial examples, we use the following equation:

$$X' = X + mask\_vector * \eta \tag{3.1}$$

Where X' is the adversarial example, X is the original example, η is the magnitude of the perturbation (ϵ) multiplied by the sign of the model gradient, and the *mask_vector* is a binary vector with the same size as input vector which for the features that we want to change, has the value 1 and for the other features 0.

Algorithm 1: FGSM-based attack algorithm.

```
Data: x original input, y original label, x' adversarial sample, F trained
      model, η perturbation magnitude
for each (x, y) ∈ Dataset do
    if F(x) = y then
        η = ε sign(∇_x J(θ, x))
        x' ← x + mask_vector * η
        if F(x') ≠ y then
            return x'
        end
    end
end
```

Algorithm 1 shows how we generate adversarial examples using different features. For each flow in the dataset, we use the FGSM method to compute the magnitude of the perturbation. Then, we multiply the mask vector of the set we use and add the result to the original input. If the classifier cannot predict the generated sample correctly, the algorithm will return it as a new adversarial example.

3.3.2 Saliency Map-based Attack

In this technique, we try to generate adversarial samples and fool the target model while changing the fewest possible features. Figure 3.5 shows the overall overview of the proposed method. In Algorithm 2, the pseudocode of the proposed method is presented.

As shown in Figure 3.5, the proposed method consists of different steps, which will be explained sequentially.

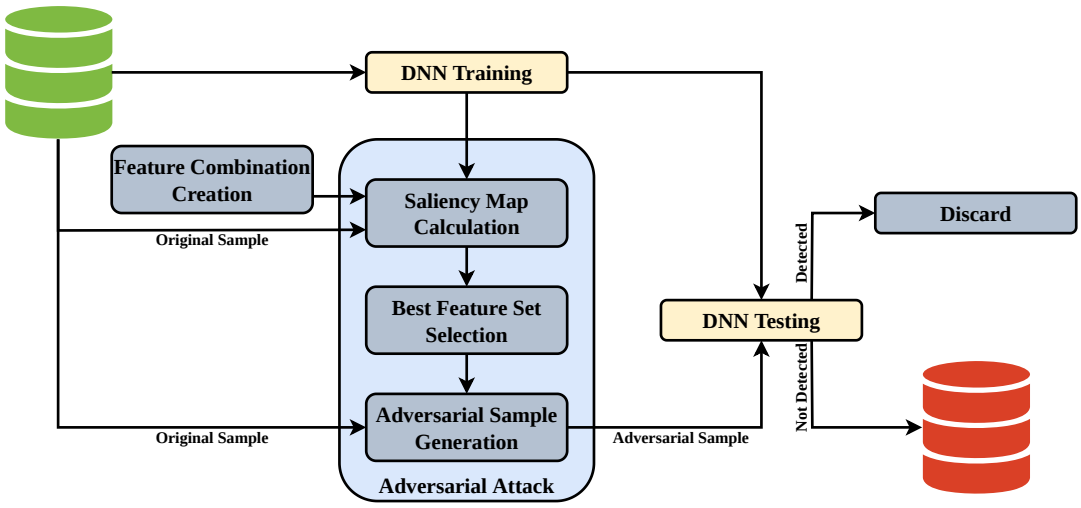


Figure 3.5: Proposed saliency map-based attack.

3.3.2.1 Feature Combinations Calculation

We want to find the best combination of features to perform the adversarial attack regardless of their nature in the proposed method. Different feature combinations are created using the following equation.

$$C_n^m \Rightarrow f_1, ..., f_n, \tag{3.2}$$

where m is all the features count, n is the number of features in the selected combination, and $f_1, ..., f_n$ are the selected features. Also, the number of different feature combinations with n number of features is:

Algorithm 2: Saliency map-based attack algorithm.

Data: $\mathbf{X}$ original input, $\mathbf{X}'$ adversarial sample, $\mathbf{F}$ trained model, η Feature combinations, ϵ perturbation magnitude

Result: Adversarial_samples

for *each X in dataset* **do**
  Compute forward derivative $J_F(X)$
  $max \leftarrow 0$
  for *each* η_i *in* η **do**
    $\alpha = \sum_{i=f_1,\ldots,f_n} J_{it}(X)$
    $\beta = \sum_{f_1,\ldots,f_n} \sum_{j \neq t} J_{ij}(X)$
    if $\alpha < 0$ ***and*** $\beta > 0$ ***and*** $-\alpha \times \beta > max$ **then**
      $selected\_features \leftarrow \eta_i$
      $max \leftarrow -\alpha \times \beta$
    end
  end
  $X' = X + selected\_features * \epsilon$
  if $F(X') \neq F(X)$ **then**
    $Adversarial\_samples \leftarrow X'$
  end
end
return *Adversarial_samples*

$$C_n^m = \frac{m!}{m!(n-m)!}. \tag{3.3}$$

3.3.2.2 Saliency Map Calculation

After creating the feature combinations, we would want to rank these combinations and select the best one for performing an adversarial attack. In [90], Papernot *et al.* extended the saliency map introduced in [105] to create an adversarial saliency map and rank the input features. This ranking shows which input features should be perturbed to have the most effect on model output. The adversarial saliency map is based on the forward derivative of the trained model.

The forward derivative is a tool to help the attacker perform the adversarial attack and cause the model to mistake and misclassify an input. The equation for the forward derivative is:

$$J_F(X) = \frac{\partial F(X)}{\partial X} = \left[\frac{\partial F_j(X)}{\partial x_i}\right]_{i \in 1...m, j \in 1...o}. \tag{3.4}$$

The forward derivative here is the Jacobian of the trained model's function regarding the input features. $F(X)$ is the function that maps the input features to an output probability and has been learned during the training phase. x_i is the feature i of input x and $F_j(X)$ is the output of the neuron j for input x. m is the number of input features, and o is the number of output neurons or classes.

The saliency map aims to find a feature to change and cause the model to misclassify an input. The model labels an input based on the $argmax\ F_j(X)$. To cause the model to misclassify an input to a target class t, the probability of the t should be increased, and the probability of all other classes should be decreased or stay unchanged until $t = argmax\ F_j(X)$.

Based on the above explanation, the saliency map is generated using the following equations:

$$S(X,t)[i] = \left\{ \begin{array}{c} 0\ if\ J_{it}(X) < 0\ or\ \sum_{j \neq t} J_{ij}(X) > 0 \\ J_{it}(X) \left| \sum_{j \neq t} J_{ij}(X) \right|\ otherwise \end{array} \right\}. \tag{3.5}$$

In the above equation, i is the input feature, X is the input sample, and t is the target class. In the first condition, if the forward derivative of the target class t concerning feature i is less than zero, which means the probability of the target class is decreasing, or the sum of the forward derivative of all other classes is positive, which means the probability of at least one of the other classes is increasing, the saliency map value is zero. The second line uses the forwarding derivative product for the target class and the sum of all other classes for the saliency map value.

Everything we explained about the saliency map is toward performing the targeted adversarial attack. But, here, we are primarily interested in fooling NIDS and do not care about the target class. To achieve this, we make some changes in the saliency

map equation. First, we consider the valid class as the target class, and instead of increasing this class probability, we try to decrease it. To do so, we change the conditions as follows:

$$S(X,t)[i] = \begin{cases} 0 \ if \ J_{it}(X) > 0 \ or \ \sum_{j \neq t} J_{ij}(X) < 0 \\ |J_{it}(X)| \left(\sum_{j \neq t} J_{ij}(X) \right) \ otherwise \end{cases}. \tag{3.6}$$

The only difference between Equation 3.6 and Equation 3.5, is the direction of the inequalities in the first line condition. Since the goal is to decrease the target class's probability, we set the saliency map to zero when the forward derivative of the target class is positive or the sum of the forward derivative of all other classes is negative. After calculating the saliency map, the features with the higher values are the ones that decrease the target class probability or increase or keep the probability unchanged for all the other classes.

3.3.2.3 Best Features Combination Selection

Since we aim to find more than one feature to change during our adversarial attack, the following equation calculates the saliency map values.

$$argmax_{f_1,\ldots,f_n} \left| \sum_{i=f_1}^{f_n} J_{it}(X) \right| \times \left(\sum_{i=f_1}^{f_n} \sum_{j \neq t} J_{ij}(X) \right). \tag{3.7}$$

Then, the combination of features that maximizes this equation will be selected to use in the adversarial samples generation step.

3.3.2.4 Adversarial Samples Generation

After finding the best feature combination for our untargeted attack, we define a mask based on those features and generate the adversarial example.

$$X' = X + mask * \epsilon. \tag{3.8}$$

where X' is the adversarial sample, X is the original sample, $mask$ is the selected features for adding the perturbation, and ϵ is the amount of perturbation.
After generating the adversarial sample, we test the model with it. If the model makes a mistake and misclassifies the input, it will be accepted as an adversarial sample; otherwise, it will be discarded. We continue with the following original input.

3.4 Proposed Evaluation Model

In this section, we will explain our evaluation model and different criteria. Figure 3.6 shows the evaluation model components. After providing the experimental results, these criteria will be used to analyze the proposed framework further.

- **Parameter Evaluation**

  The two critical parameters in the proposed method are the magnitude of ϵ and the number of selected features. We compare the effect of these two factors on the suggested method results and use it as one of the evaluation criteria.

- **Distance Analysis**

  As we mentioned before, one of the essential factors of an adversarial attack is the distance between the original and adversarial samples. The three most commonly used distance metrics are l_0, l_2, and l_∞.

- **The DNN Model's Performance**

  The main goal of an adversarial attack is to decrease the performance of the target model. We replace the original samples with the generated adversarial samples in the dataset and examine the performance of the trained model on the new dataset.

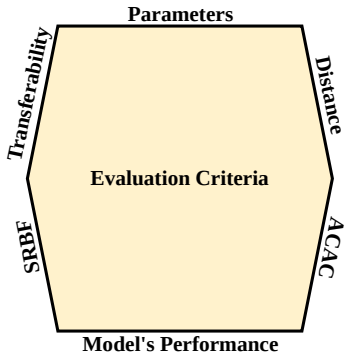


Figure 3.6: Proposed evaluation model.

- **Success Rates of the Best Feature Sets (SRBF)**

  The features that are the best for generating adversarial samples for each record are selected in the proposed method. Still, they may be able to generate adversarial examples for other records while they are not the best selection. We select the most successful features with each ϵ value and use them to generate adversarial samples for the whole dataset and consider these results as one of the evaluation criteria.

- **Average Confidence of Adversarial Class (ACAC)**

  A deep learning model generates a probability for each output class and selects the class with the highest probability as its prediction. If the prediction probability for adversarial samples is high, the model classifies them with high confidence, and the attack is more powerful. We calculate the average confidence score for generated adversarial examples and use it as a criterion to compare our different experiments.

- **Transferability**

  One of the most interesting properties of adversarial examples is transferability. Adversarial examples that affect one model often affect another, even if the two models have different architectures or were trained on different training

sets [88]. One of the primary uses of transferability is to craft adversarial examples on a substitute model and use these examples to make a black-box attack on another model. There are two types of adversarial examples of transferability:

- **Intra-technique Transferability** is defined as transferring examples between machine learning models trained using the same machine learning technique but with different parameters or datasets.
- **Cross-technique Transferability** is defined as transferring examples between two machine-learning models trained using different machine-learning techniques.

3.5 Concluding Remarks

In this chapter, we covered contributions from one to five. We presented our proposed framework, which includes two main components: poisoning attack and evasion attack. For the poisoning attack, we proposed a label-flipping using stratified sampling and random label-flipping. We explained our two methods based on the FGSM and Saliency map for the evasion attack. In the FGSM-based attack we categorized network features in to six groups based on their nature and used one or combination of these groups for generating adversarial examples, while in the saliency map-based attack our goal was to find the best combination of features, regardless of their nature, which have the most effect on the target model decision. Finally, we introduce an evaluation model for the evasion attack based on five criteria: parameter evaluation, distance analysis, target model performance under attack, the success rate of the best feature sets, the average confidence score of the adversarial class, and transferability.

In the next chapter, we explain the implementation process, including the dataset

description, tools, and framework used for the implementation.

Chapter 4

Implementation

In this chapter, first, we introduce the selected datasets in detail, including their attack specifications and generation process. Then, we explain the tools and platforms used to implement the proposed framework.

4.1 Datasets Description

One of the critical factors in researching NIDS is selecting a comprehensive dataset that includes an extensive set of labeled intrusions and abnormal behavior that covers up-to-date network attacks. To do our experiments, we decided on three datasets. CIC-IDS2017 [101] and CIC-IDS2018, which are from our institute. To have more evaluation on an external dataset, we chose UNSW-NB15 [81, 82]. In the following subsections, we will review these datasets in more detail.

4.1.1 CIC-IDS2017 and CIC-IDS2018

These two datasets contain several network attacks. In the CIC-IDS2017 [101], they argued that covering eleven criteria mentioned in the evaluation framework in [36] is necessary for an IDS dataset, and their dataset can cover all of them. CIC-IDS2018 is an extension to CIC-IDS2017 and contains the same kind of attacks. They extracted

Table 4.1: CIC-IDS2017 details.

Category	Number of Records
Benign	2271320
DDoS	128025
PortScan	158804
Botnet	1956
Infiltration	36
Web Attack-Brute Force	1507
Web Attack-SQL Injection	21
Web Attack-XSS	652
FTP-Patator	7935
SSH-Patator	5897
DoS GoldenEye	10293
DoS Hulk	230124
DoS Slowhttp	5499
Dos Slowloris	5796
Heartbleed	11

Table 4.2: CIC-IDS2018 details.

Category	Number of Records
Benign	13390249
Bot	286191
Infiltration	160639
Brute Force-Web	611
Brute Force-XSS	230
SQL Injection	87
FTP-BruteForce	193354
SSH-BruteForce	187589
DoS GoldenEye	41508
DoS Hulk	461912
DoS Slowhttp	139890
Dos Slowloris	10990
DDoS LOIC-HTTP	576191
DDoS LOIC-UDP	1730
DDoS HOIC	686012

over 80 network traffic features from their datasets using CICFlowMeter [61] and labeled each flow as benign or attack name. Tables 4.1 and 4.2 present the details of these two datasets. Some features in these datasets, including Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, and Timestamp, are unsuitable for a DNN model. We removed them during the prepossessing step.

As mentioned before the intention of these datasets is to be useful for network intrusion detection systems. Therefore they should cover a diverse and up-to-date set of attack scenarios. In the following, we will introduce each of the attack families that

are present in these datasets.

- **Brute force:** It is a method used by hackers to gain unauthorized access to a system or encrypted data by systematically trying all possible combinations of passwords or encryption keys until the correct one is found. It's an exhaustive trial-and-error approach that uses computational power and time to discover the correct password or key.

  Brute force attacks can be resource-intensive and time-consuming, especially if the target has robust security measures, such as complex passwords, account lockouts, or encryption algorithms with long keys. However, they can still be successful if the attacker has sufficient computational power and time available or if the target's security measures are weak.

  Both CIC-IDS2017 and CIC-IDS2018 used Patator, which is written in Python and supports many scenarios, including FTP and SSH.

- **Heartbleed:** This is a security vulnerability discovered in 2014 in a widely used encryption library called OpenSSL, which is used to secure communications on the internet. The vulnerability allowed attackers to access sensitive information from the memory of the targeted system.

  The attack exploited a flaw in the OpenSSL's implementation of the Transport Layer Security (TLS) heartbeat extension. Attackers could send a maliciously crafted heartbeat request to a vulnerable server, tricking it into returning more data from its memory than it should. This extra data could include sensitive information such as usernames, passwords, encryption keys, and other confidential data. In CIC-IDS2017, they used Heartleech to exploit Heartbleed vulnerability.

- **Botnet:** It is a cyber attack where a large number of infected computers,

called bots, are controlled remotely by an attacker. The attacker can use the botnet for various purposes, such as launching DDoS attacks, sending spam or phishing emails, stealing information, or engaging in click fraud. Botnet attacks are challenging to defend against because they involve many distributed and often geographically dispersed bots.

Creating a botnet usually involves infecting computers with malware through various means, such as email attachments, malicious downloads, or exploiting vulnerabilities in software or operating systems. Once infected, the compromised machines become part of the botnet and can be remotely controlled by the attacker.

They used Ares in CIC-IDS2017 and both Zeus and Ares in CIC-IDS2018. Ares's capabilities include remote shell, taking screenshots, and keylogging, while Zeus is mainly used to steal banking information by keystroke logging and form grabbing.

- **Denial-of-Service (DoS):** This attack is a malicious attempt to disrupt the normal functioning of a computer network, service, or website by overwhelming it with a flood of illegitimate requests or by exploiting vulnerabilities in the targeted system. The objective of a DoS attack is to exhaust the resources of the targeted system, rendering it unable to fulfill legitimate requests or causing it to crash. The basic principle behind a DoS attack is to flood the target with excessive traffic or requests, exceeding its capacity to handle them.

  It can result in extended downtime, loss of revenue for businesses, disruption of critical services, and damage to a company's reputation. In some cases, DoS attacks may serve as a diversionary tactic, drawing attention away from malicious activities such as data theft or network infiltration.

  Distributed Denial-of-Service (DDoS) attack is a DoS attack involving multiple

compromised computers or devices, forming a botnet. The attacker controls this network of bots, instructing them to attack the target simultaneously.

There are several tools available for performing DoS and DDoS attacks. They used GoldenEye, Hulk, Slowhttp, and Slowloris in both datasets to conduct DoS attacks. They used two well-known High Orbit Ion Canon (HOIC) and Low Orbit Ion Canon (LOIC) tools for the DDoS attack.

- **Web Attacks:** They are malicious activities aimed at exploiting vulnerabilities in web applications or websites to gain unauthorized access, disrupt their functioning, or steal sensitive information. These attacks target the underlying technologies and weaknesses in web applications, web servers, or the communication protocols used on the web.

  In both datasets, they used three web attacks:

  - **Cross-Site Scripting (XSS)** occurs when an attacker injects malicious code (usually JavaScript) into a web application, which is then executed by unsuspecting users visiting the website. This can be used to steal user data, manipulate web content, or redirect users to malicious sites.
  - **SQL Injection** involves inserting malicious SQL statements into an application's database query to manipulate the database or gain unauthorized access. Attackers can extract sensitive information, modify or delete data, or escalate their privileges through this vulnerability.
  - **Brute force over HTTP** is a method employed by attackers to gain unauthorized access to web applications, user accounts, or sensitive information by systematically trying various combinations of usernames and passwords. It involves using automated tools or scripts to repeatedly submit login attempts to a web application's login page.

- **Infiltration:** This attack, also known as an intrusion or penetration attack, is a type of cybersecurity attack in which an unauthorized individual gains access to a computer system, network, or application to compromise its security, steal sensitive information, or perform malicious activities. They used Metasploit to implement this scenario. After infecting the victim machine, they used Nmap on the entire victim network.

- **PortScan:** This attack is used by attackers to identify open ports on a target system or network. Ports are endpoints on a computer that enable specific services or applications to communicate over a network. By scanning ports, attackers can gather information about the network's vulnerabilities and potential entry points for further exploitation. In CIC-IDS2017, they performed the PortScan attack using NMap main switches.

4.1.2 UNSW-NB15

As we mentioned before, we decided to select an external dataset to have a deeper analysis of our proposed framework. After doing extensive research, we chose the UNSW-NB15 dataset. They used the IXIA PerfectStorm tool to generate this dataset to create modern normal and abnormal network traffic. Their dataset includes nine attack categories and benign traffic. They captured 100GBs of network traffic in two days, and to extract features from the captured network traffic, they used Argus and Bro-IDS tools. They extracted 47 features in categories, including Basic, Content, Time, and additional generated features. In the following, we briefly explain each attack category in the dataset.

- **Fuzzers:** A fuzzer attack, or fuzzing, is a technique used to discover vulnerabilities in software. It involves sending unexpected or random input data to an application to see how it responds. Overwhelming the application with varied

inputs can identify weaknesses or flaws. Fuzzing is an automated process using specialized tools called fuzzers. When a vulnerability is found, attackers can further analyze it and potentially exploit it.

- **Analysis:** This attack is a method where attackers gather and study information to exploit vulnerabilities in a system. This attack involves techniques such as traffic analysis, cryptographic analysis, code analysis, data analysis, and protocol analysis. Attackers use these techniques to gain insights, extract sensitive data, or identify weaknesses that can be exploited for malicious purposes.

- **Backdoor:** A backdoor attack is a cybersecurity threat where unauthorized access is gained to a computer system or network by exploiting hidden vulnerabilities or intentionally creating openings. It involves the insertion of malicious code or modifications to existing code within a system, allowing attackers to bypass standard security measures and gain control over the targeted system to perform various malicious activities, such as stealing sensitive data, installing additional malware, or launching further attacks on the system or network.

- **Exploit:** An exploit attack refers to exploiting computer systems or software vulnerabilities to gain unauthorized access or perform malicious activities. Exploits take advantage of weaknesses or flaws in a system's design or implementation, allowing attackers to execute specific commands or actions not intended by the system's developers. These vulnerabilities can exist in various components, such as operating systems, applications, or network protocols.

- **Generic:** It is a kind of attack against the cryptography systems, which can run against all block-ciphers independently of their structure.

- **Reconnaissance:** This attack, also known as information gathering or footprinting, is a cyber attack that focuses on gathering valuable intelligence and

information about a target system or network. The main objective of a reconnaissance attack is to gain a deeper understanding of the target's infrastructure, vulnerabilities, and potential entry points without directly causing any damage.

- **Shellcode:** The term shellcode refers to a small piece of code that is injected into a vulnerable program, typically to gain unauthorized access and control over the system. The attacker first identifies a vulnerability in the target software, such as a buffer overflow or a code injection flaw. They then craft a payload, usually written in assembly language or machine code and designed to perform specific actions once executed. This payload is the shellcode.

- **Worms:** This malicious cyber attack spreads through computer networks, targeting vulnerable systems and exploiting security vulnerabilities. Unlike viruses or Trojans, worms do not require user interaction to propagate. They can independently replicate and spread across a network, infecting multiple computers and devices.

Since they have provided all the pcap files and we require datasets with the same set of features, we used CICFlowMeter to extract the new set of features from the provided captured network traffic. After extracting the flows using CICFlowMeter, we need to label them using the ground truth from the original dataset files.

We matched the extracted flows with the records in the ground truth file based on the source IP, destination IP, source port, destination port, and protocol. If any flows match with a record from the ground truth file, we set the label using the ground truth attack category. If the flow is matched with more than one record from the ground truth file, we compare the timestamps and choose the record's label that matches the flow timestamp. In the worst case, the flow will be dropped even if we can not decide on the label by comparing the timestamp. Any remaining flows will

Table 4.3: UNSW-NB15 details.

Category	Original Dataset	CICFlowMeter	CIC-UNSW
Benign	221876	3450658	358332
Analysis	2677	385	385
Backdoor	2329	452	452
DoS	16353	4467	4467
Exploits	44525	30951	30951
Fuzzers	24246	29613	29613
Generic	215481	4632	4632
Reconnaissance	13987	16735	16735
Shellcode	1511	2102	2102
Worms	174	246	246

be labeled benign after labeling all the malicious flows.

Table 4.3, includes the details of the original UNSW-NB15 dataset and the extracted flows using CICFlowMeter. In most of the network traffic datasets to be closer to the real world, they keep the ratio between the benign and malicious samples 80 percent to 20 percent. To gain this ratio, we keep all the malicious flows extracted by the CICFlowMeter and randomly sample the required number of flows from the benign flows. The last column of Table 4.3, shows the final details of the newly generated dataset. Since we used the raw packet files from UNSW-NB15 and the CICFlowMeter to generate this dataset, we will call it CIC-UNSW.

4.2 Implementation

First, we need to train DNN models for classifying network attacks in the three selected datasets to do our experiments. Then, we use the proposed framework to perform the adversarial attacks against the trained classifiers. We performed the experiments on a machine with the NVIDIA TITAN V GPU and used Python language for the programming. Additionally, PyTorch and scikit-learn libraries were used to develop the deep learning model, machine learning models, and adversarial attacks.

Table 4.4: Results of the Classifiers.

Machine Learning Techniques	**CIC-IDS2017**			**CIC-IDS2018**			**CIC-UNSW**		
	F1-score	PC	RC	F1-score	PC	RC	F1-score	PC	RC
DT	99.84	99.76	99.92	97.70	99.73	96.14	98.81	98.93	98.7
Naive Bayes	28.47	32.43	73.75	48.29	49.48	72.79	22.43	43.14	44.57
LR	36.71	39.76	34.96	57.97	64.13	56.84	32.99	46.26	33.61
RF	96.58	99.79	94.24	94.23	99.81	90.97	91.87	96.06	88.79
DNN	98.18	98.27	98.22	98.71	98.99	99.04	92.06	93.63	92.14

There are a number of different methods that can be used to create a NIDS. Since in the literature, it is common [110, 29, 56] to use DNN as a tool to build a NIDS, we train a DNN model for multi-class classification on each dataset and compare its performance with other machine learning techniques. The focus here is on adversarial attacks against deep learning models. Therefore, the results from other machine learning methods (Decision Tree, Naive Bayes, Logistic Regression, and Random Forest) presented in Table 4.4 are only for comparison purposes. Moreover, we use these trained machine learning models to evaluate the transferability of adversarial samples. All the hyper-parameters for the machine learning models are the default values in the scikit-learn library. For example for the Decision Tree, the criterion is gini and min_sample_split is two. The DNN model is a simple multi-layer perceptron. We use F1-score, precision (PC), and recall (RC) as the evaluation metrics. As presented in Table 4.4, the DNN model has comparable performance with other machine learning models and state-of-the-art works like Lopez-Martin et al. [68] where they used deep neural network and RBF neural network for classification task on CIC-IDS2017 and CIC-DDoS2019.

For implementing our proposed framework we used Python and PyTorch. We used PyTorch for training the target models, implementing the FGSM-based and Saliency map-based attack, and evaluating the attack techniques. Also, Scikit-learn and Pandas were used for the dataset pre-processing and implementation of the label-flipping attack.

4.3 Concluding Remarks

In this chapter, we explained the implementation process of the proposed framework including the tools and frameworks used for the implementation and datasets used for the evaluation. We also explained the feature extraction and labeling process of the CIC-UNSW, which covers part of our sixth contribution.

In the next chapter, we will present our experimental results and provide a detailed analysis and discussion.

Chapter 5

Evaluation Results and Discussion

As explained in the previous chapter, the proposed framework includes three attacks. This chapter will provide the detailed results of these attacks for each dataset. We also present a deeper analysis and discussion of the experimental results for each attack. In addition to the results and analysis of the three attacks, in Section 5.4, we provide the results of an experiment where the target model is under both label-flipping and saliency map-based attacks.

5.1 Label Flipping Attack

We're going to present the results of the proposed label-flipping attack. To better understand the attack's effect, we start by flipping 10 percent of the labels and go up to 70 percent. In each step, we keep the previously flipped labels and select another 10 percent of actual labels for the label flipping. We do the attack for different percentages of the flipped labels ten times and report our results' average and standard deviation. Also, from those ten various experiments, the one with the less effect on the model's performance is selected and used in the next step.

In most network datasets, including the selected ones, the ratio of malicious to benign traffic is 20 percent to 80 percent. To evaluate the effect of different ratios on the

deep learning model performance in the presence of label flipping attack, we adjust the balance between the malicious and benign traffic from 20-80 to 30-70, 40-60, 50-50, 60-40, 70-30, and 80-20 in the preformed experiences.

5.1.1 CIC-IDS2017

Table 5.1 presents the results for the CIC-IDS2017 dataset. The first main row is the classifier's performance without label flipping, and each sub-row is for the different ratio of the malicious to benign traffic in the dataset. The other main rows are label-flipping attack results with varying percentages of flipped labels from 10 to 70. As shown in Table 5.1, increasing the percentage of flipped labels decreases the performance of the deep learning model. The average accuracy is 98.06% without flipped labels, dropping to 29.25% when 70 percent of the labels are flipped. Figure 5.1 shows the model's performance for the dataset with the 40-60 ratio and different percentages of the filliped label. All four metrics drop significantly when the percentage increases from 10 to 70.

With the same percentage of the flipped labels, when we change the ratio of malicious to benign samples, the accuracy stays almost the same, but the recall and F1-score increase. In the beginning, when the distribution is 20-80, most of the selected labels for flipping attacks are from the benign class, and the number of actual benign samples is high enough for the model to learn and detect them. But, with changing the training dataset toward increasing the ratio of malicious samples, the model's ability to detect the benign samples in the presence of label flipping attack drops while its performance for other classes increases, and since in the multi-class classification, the average is used to calculate the recall, this metric will increase. Figure 5.2 shows the results for the attack with 40% flipped labels with different malicious to benign ratios. As it is clear, with changing the distribution, accuracy stays almost without change, but recall and F1-score increase.

Table 5.1: CIC-IDS2017 Label flipping results.

Poisoning Percentage	Ratio	Accuracy	Precision	Recall	F1-score
0	20-80	97.66	95.47	87.82	89.84
	30-70	98.05	95.83	87.52	89.99
	40-60	97.94	93.97	91.06	91.72
	50-50	98.14	95.22	94.59	94.82
	60-40	97.93	96.38	91.99	93.68
	70-30	98.17	97.03	90.28	92.82
	80-20	98.55	95.62	94.64	94.84
10	20-80	88.08±0.21	77.62±1.57	38.44±0.37	43.28±0.38
	30-70	87.94±0.20	79.28±2.53	43.46±0.12	48.60±0.15
	40-60	87.84±0.16	78.90±2.58	46.80±0.16	52.25±0.26
	50-50	87.87±0.08	79.37±0.52	49.29±0.12	54.91±0.19
	60-40	87.99±0.02	78.76±0.48	51.69±0.12	57.50±0.19
	70-30	88.26±0.03	81.68±4.19	53.52±0.06	59.41±0.07
	80-20	88.55±0.06	79.50±2.27	55.72±0.81	61.53±0.72
20	20-80	78.03±0.06	66.42±3.32	30.86±0.27	33.62±0.17
	30-70	77.96±0.17	69.91±0.71	35.49±0.20	38.07±0.20
	40-60	77.86±0.06	70.35±2.06	38.48±0.08	41.23±0.12
	50-50	77.98±0.06	70.02±0.80	40.68±0.12	43.62±0.21
	60-40	78.15±0.02	70.20±0.64	42.55±0.12	45.73±0.20
	70-30	78.43±0.03	70.43±0.52	44.17±0.06	47.63±0.10
	80-20	78.64±0.04	70.65±0.20	45.38±0.04	49.07±0.04
30	20-80	68.23±0.7	58.05±2.74	26.33±0.14	27.75±0.13
	30-70	68.09±0.04	60.57±2.93	30.64±0.08	31.34±0.09
	40-60	68.07±0.06	61.39±1.51	33.41±0.04	33.99±0.12
	50-50	68.22±0.02	61.59±0.66	35.53±0.9	36.15±0.17
	60-40	68.36±0.02	61.08±0.44	37.25±0.04	38.06±0.07
	70-30	68.59±0.02	61.80±0.44	38.54±0.06	39.50±0.11
	80-20	68.80±0.02	61.68±0.16	39.51±0.02	40.76±0.03
40	20-80	58.48±0.07	47.42±1.67	22.77±0.08	23.05±0.03
	30-70	58.29±0.08	52.06±2.63	26.67±0.14	26.10±0.07
	40-60	58.34±0.05	50.18±2.47	29.43±0.05	28.38±0.09
	50-50	58.40±0.03	52.86±1.49	31.37±0.04	30.07±0.07
	60-40	58.57±0.04	51.92±1.44	32.97±0.08	31.63±0.15
	70-30	58.77±0.02	53.22±0.48	34.17±0.04	32.98±0.06
	80-20	58.94±0.04	52.78±0.30	34.89±0.05	33.98±0.08
50	20-80	48.69±0.03	39.01±0.1	19.69±0.11	18.84±0.06
	30-70	48.54±0.07	41.13±2.71	23.18±0.14	21.51±0.05
	40-60	48.58±0.04	40.08±1.58	25.84±0.03	23.43±0.05
	50-50	48.65±0.02	43.70±2.17	27.66±0.02	24.83±0.03
	60-40	48.76±0.02	42.24±1.62	29.12±0.05	26.09±0.11
	70-30	48.96±0.02	53.22±0.48	34.17±0.04	32.98±0.06
	80-20	49.07±0.04	43.98±0.18	30.68±0.04	27.96±0.05
60	20-80	38.96±0.02	31.19±0.23	17.00±0.05	14.96±0.05
	30-70	38.84±0.04	32.04±1.45	19.93±0.13	17.19±0.07
	40-60	38.90±0.04	31.64±0.12	22.30±0.02	18.82±0.05
	50-50	38.95±0.02	33.49±2.02	23.98±0.01	20.00±0.03
	60-40	39.03±0.01	32.83±1.51	25.27±0.01	20.96±0.02
	70-30	39.13±0.03	33.54±1.54	26.13±0.05	21.79±0.10
	80-20	39.20±0.03	34.97±0.16	26.48±0.02	22.37±0.04
70	20-80	29.22±0.06	22.91±1.22	14.41±0.1	11.13±0.09
	30-70	29.15±0.03	24.40±2.94	16.65±0.05	12.94±0.11
	40-60	29.17±0.02	23.80±0.11	18.84±0.06	14.19±0.07
	50-50	29.25±0.01	25.03±1.84	20.00±0.01	15.21±0.02
	60-40	29.27±0.02	25.55±2.86	21.03±0.01	15.93±0.03
	70-30	29.35±0.01	25.19±1.37	21.69±0.01	16.52±0.02
	80-20	29.40±0.02	27.02±2.93	21.92±0.02	16.95±0.06

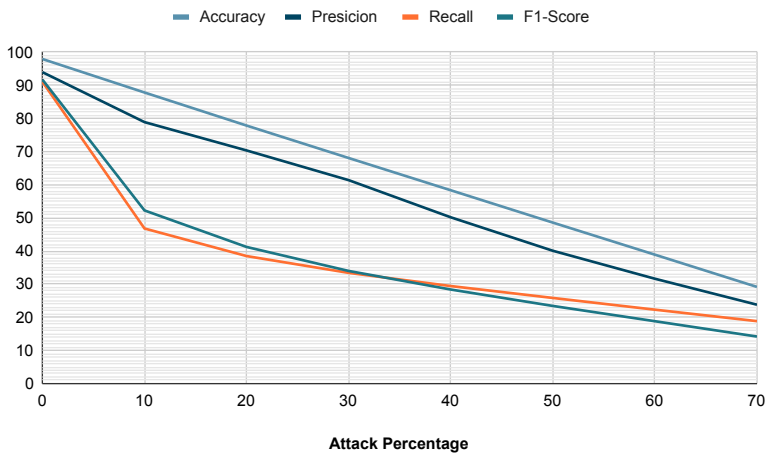


Figure 5.1: CIC-IDS2017 Results for Dataset with 40-60 Malicious to Benign Ratio.

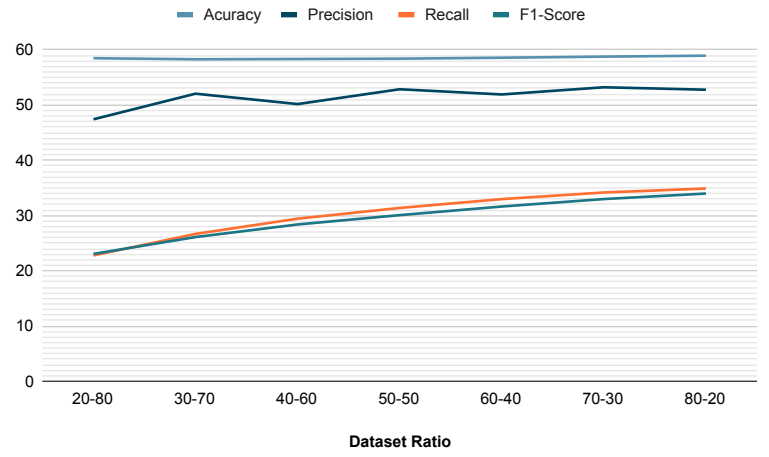


Figure 5.2: CIC-IDS2017 Results with 40% Flipped Labels.

5.1.2 CIC-IDS2018

The same experiments have been done for the CIC-IDS2018 dataset, and the results are reported in Table 5.2. The percentage of flipped labels are increased from 10 to 70, and for each value, the experiments are done with different ratio of the malicious to benign samples. Similar to the results of the CIC-IDS2017 dataset, the model's performance drops when the percentage of the flipped labels is increased. The average accuracy drops from 97.36% with no flipped labels to 29.2% when 70% of the labels are flipped. In Figure 5.3, decreasing performance metrics are clearly shown for the

Table 5.2: CIC-IDS2018 Label flipping results.

Poisoning Percentage	Ratio	Accuracy	Precision	Recall	F1-score
0	20-80	95.87	91.53	81.64	83.58
	30-70	98.27	91.80	88.69	89.48
	40-60	97.96	91.13	92.03	90.97
	50-50	97.66	91.71	92.17	91.51
	60-40	97.71	92.46	94.02	92.82
	70-30	97.05	92.19	92.41	91.81
	80-20	97.01	92.54	94.07	92.94
10	20-80	88.71±0.27	82.66±0.74	46.49±0.62	53.16±0.44
	30-70	88.32±0.15	82.75±0.52	55.61±0.34	60.11±0.31
	40-60	87.90±0.16	82.62±0.51	59.22±0.33	62.81±0.38
	50-50	87.48±0.21	82.79±0.58	61.96±0.16	64.95±0.29
	60-40	87.91±0.04	83.17±0.25	65.58±0.09	68.60±0.12
	70-30	87.26±0.08	83.09±0.41	65.94±0.27	68.57±0.38
	80-20	87.29±0.05	83.65±0.36	68.27±0.10	70.69±0.13
20	20-80	78.87±0.19	73.21±0.42	35.69±0.49	41.71±0.36
	30-70	78.40±0.11	73.41±0.39	44.97±0.12	49.20±0.15
	40-60	78.15±0.03	73.31±0.31	49.42±0.08	52.50±0.11
	50-50	77.86±0.08	73.56±0.37	52.51±0.18	54.90±0.18
	60-40	78.05±0.4	73.91±0.17	55.87±0.08	57.90±0.13
	70-30	77.47±0.07	73.66±0.37	57.09±0.10	58.46±0.15
	80-20	77.56±0.05	74.34±0.38	59.30±0.10	60.55±0.14
30	20-80	68.99±0.16	63.49±0.90	28.79±0.27	33.39±0.34
	30-70	68.63±0.06	64.59±0.32	37.44±0.02	40.83±0.06
	40-60	68.36±0.11	64.40±0.39	41.83±0.13	44.15±0.18
	50-50	68.03±0.14	64.38±0.19	45.00±0.16	46.49±0.18
	60-40	68.27±0.02	64.58±0.21	48.41±0.04	49.38±0.07
	70-30	67.76±0.09	64.48±0.15	49.93±0.12	50.14±0.13
	80-20	67.79±0.08	64.77±0.16	51.96±0.10	51.99±0.16
40	20-80	58.98±0.44	54.21±1.12	23.40±0.71	26.45±0.88
	30-70	58.81±0.02	55.24±0.26	31.35±0.04	33.64±0.06
	40-60	58.53±0.09	55.21±0.29	35.38±0.11	36.75±0.11
	50-50	58.31±0.03	55.18±0.18	38.59±0.05	39.10±0.09
	60-40	58.49±0.07	55.64±0.33	41.79±0.08	41.70±0.07
	70-30	58.04±0.10	55.28±0.05	43.44±0.15	42.61±0.16
	80-20	58.11±0.03	55.47±0.03	45.30±0.04	44.27±0.08
50	20-80	48.88±0.35	44.62±0.89	19.27±0.49	20.51±0.69
	30-70	48.99±0.6	45.95±0.22	26.20±0.06	27.20±0.10
	40-60	48.72±0.11	45.86±0.33	29.73±0.12	29.99±0.12
	50-50	48.58±0.05	46.09±0.18	32.71±0.05	32.16±0.08
	60-40	48.70±0.06	46.49±0.28	35.56±0.07	34.48±0.05
	70-30	48.33±0.09	46.23±0.11	37.14±0.11	35.35±0.11
	80-20	48.45±0.01	46.36±0.04	38.83±0.01	36.87±0.01
60	20-80	39.23±0.35	36.02±0.85	16.40±0.47	16.05±0.61
	30-70	39.05±0.33	36.67±0.49	21.52±0.36	21.09±0.48
	40-60	39.00±0.07	36.85±0.20	24.65±0.07	23.74±0.08
	50-50	38.87±0.07	36.85±0.18	27.16±0.07	25.61±0.06
	60-40	38.99±0.01	37.22±0.12	29.55±0.01	27.55±0.02
	70-30	38.72±0.06	37.21±0.23	30.98±0.07	28.41±0.05
	80-20	38.81±0.02	36.95±0.85	32.30±0.02	29.64±0.05
70	20-80	29.40±0.12	26.46±1.08	13.82±0.15	11.63±0.25
	30-70	29.18±0.28	27.13±0.66	17.37±0.29	15.47±0.36
	40-60	29.15±0.29	27.70±0.35	19.68±0.28	17.60±0.29
	50-50	29.15±0.08	28.00±0.17	21.66±0.07	19.16±0.12
	60-40	29.30±0.01	27.95±0.72	23.49±0.01	20.75±0.02
	70-30	29.04±0.04	28.00±0.19	24.49±0.04	21.47±0.05
	80-20	29.18±0.00	28.11±0.02	25.48±0.00	22.48±0.01

dataset with the 40-60 ratio.

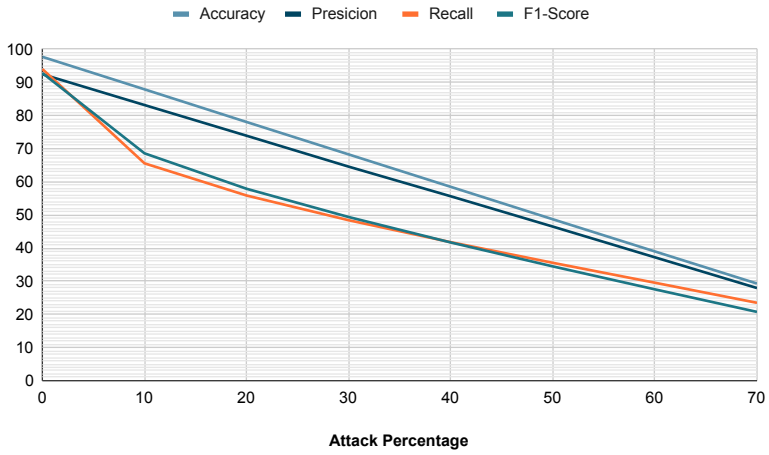


Figure 5.3: CIC-IDS2018 Results for Dataset with 40-60 Malicious to Benign Ratio.

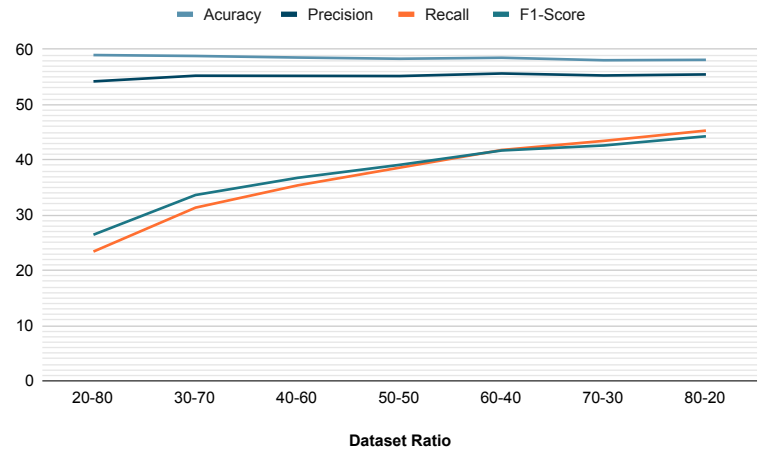


Figure 5.4: CIC-IDS2018 Results with 40% Flipped Labels.

The results for each percentage of the flipped label are the same as CIC-IDS2017. The accuracy stays almost unchanged, and the F1-score and recall increase significantly. In Figure 5.4, you see how accuracy, precision, recall, and F1-score change when the ratio of the dataset changes from 20-80 to 80-20.

Table 5.3: CIC-UNSW Label flipping results.

Poisoning Percentage	Ratio	Accuracy	Precision	Recall	F1-score
0	20-80	92.14	61.64	47.08	48.22
	30-70	89.37	68.1	44.57	46.49
	40-60	86.81	66.48	42.74	44.40
	50-50	83.80	70.12	48.08	50.45
	60-40	81.06	68.07	43.58	45.77
	70-30	78.11	69.37	42.02	43.39
	80-20	75.36	79.41	41.69	43.41
10	20-80	82.71±0.03	58.85±1.31	33.41±0.18	33.86±0.25
	30-70	80.15±0.05	60.79±3.89	35.22±0.18	30.51±0.26
	40-60	77.52±0.06	55.74±5.39	36.05±0.16	36.14±0.19
	50-50	75.02±0.10	55.38±1.02	37.09±0.24	37.09±0.26
	60-40	72.55±0.14	55.52±1.03	37.70±0.25	37.73±0.25
	70-30	69.90±0.09	55.24±1.50	37.95±0.33	37.89±0.35
	80-20	67.49±0.19	55.92±0.55	38.10±0.26	38.31±0.37
20	20-80	73.56±0.02	49.59±4.24	28.63±0.08	28.58±0.16
	30-70	71.31±0.03	54.93±4.11	31.22±0.05	30.71±0.11
	40-60	69.00±0.04	50.96±4.33	32.58±0.10	31.64±0.13
	50-50	66.70±0.10	51.68±3.89	33.70±0.14	32.50±0.17
	60-40	64.45±0.07	49.35±1.77	34.34±0.10	33.11±0.14
	70-30	62.21±0.14	49.15±0.59	34.94±0.17	33.62±0.20
	80-20	60.06±0.19	50.48±3.43	35.09±0.17	33.90±0.19
30	20-80	64.42±0.3	40.81±3.42	24.96±0.05	24.27±0.07
	30-70	62.52±0.05	45.38±3.59	27.88±0.08	26.53±0.12
	40-60	60.44±0.05	41.34±1.87	29.50±0.07	27.57±0.09
	50-50	58.52±0.07	44.08±1.73	30.84±0.09	28.59±0.11
	60-40	56.58±0.09	42.09±2.02	31.60±0.07	29.17±0.09
	70-30	54.72±0.10	44.34±0.08	32.26±0.08	29.69±0.10
	80-20	52.96±0.13	43.32±3.00	32.44±0.08	30.13±0.09
40	20-80	55.34±0.03	35.18±3.1	22.07±0.05	24.27±0.07
	30-70	53.74±0.05	36.21±1.87	24.93±0.06	22.81±0.08
	40-60	51.96±0.06	34.72±3.23	26.59±0.09	23.72±0.17
	50-50	50.45±0.07	35.35±1.17	28.18±0.08	24.97±0.08
	60-40	48.79±0.07	35.78±1.57	29.01±0.05	25.53±0.08
	70-30	47.18±0.12	35.20±4.56	29.45±0.12	25.72±0.17
	80-20	45.74±0.12	35.03±1.48	29.63±0.09	26.22±0.09
50	20-80	46.23±0.02	29.35±1.20	19.56±0.03	17.27±0.03
	30-70	44.94±0.04	31.79±2.28	22.15±0.04	19.20±0.05
	40-60	43.50±0.08	31.67±5.83	23.80±0.08	20.13±0.16
	50-50	42.23±0.12	31.60±2.07	25.23±0.10	21.16±0.09
	60-40	40.94±0.07	31.71±1.66	26.17±0.05	21.82±0.06
	70-30	39.50±0.11	27.04±0.17	26.49±0.11	21.91±0.14
	80-20	38.62±0.08	37.13±3.03	26.69±0.06	22.45±0.10
60	20-80	37.17±0.05	23.63±1.66	17.35±0.06	14.03±0.09
	30-70	36.08±0.04	24.15±2.52	19.41±0.04	15.62±0.08
	40-60	35.05±0.06	25.15±4.79	20.95±0.07	16.53±0.09
	50-50	34.14±0.09	24.96±3.64	22.26±0.08	17.42±0.06
	60-40	33.15±0.07	23.22±1.93	23.03±0.05	17.90±0.08
	70-30	32.04±0.11	27.04±0.17	26.49±0.11	21.91±0.14
	80-20	31.30±0.07	31.34±3.98	23.32±0.07	18.49±0.12
70	20-80	28.04±0.02	18.23±1.72	15.23±0.03	10.77±0.06
	30-70	27.32±0.03	21.90±2.26	16.77±0.04	12.13±0.09
	40-60	26.65±0.06	19.14±3.22	17.96±0.06	12.85±0.07
	50-50	25.95±0.08	24.25±4.57	18.88±0.10	13.49±0.21
	60-40	25.33±0.05	18.07±1.70	19.51±0.05	13.99±0.12
	70-30	24.41±0.15	18.50±2.80	19.50±0.12	13.98±0.13
	80-20	24.11±0.14	23.01±5.71	19.60±0.08	14.55±0.10

5.1.3 CIC-UNSW

Same as two other datasets the results are presented in Table 5.3. The first interesting thing about CIC-UNSW is that even when there is no label-flipping attack present, changing the ratio of the dataset has a noticeable effect on the accuracy which decreased from 92.14% to 75.36%. The reason for this phenomenon is that the number of samples in CIC-UNSW is less than the other datasets and changing the ratio reduces this number way more and causes a decline in the deep learning model performance.

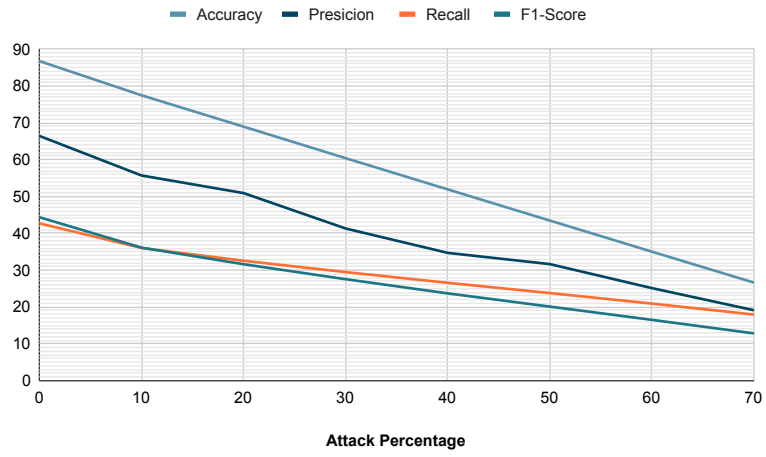


Figure 5.5: CIC-UNSW Results for Dataset with 40-60 Malicious to Benign Ratio.

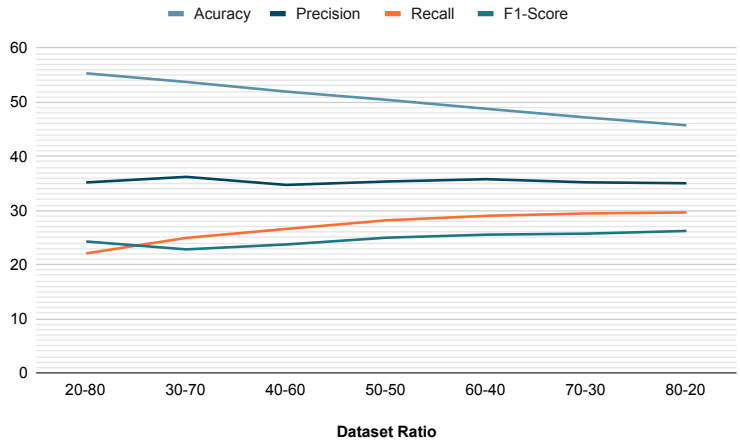


Figure 5.6: CIC-UNSW Results with 40% Flipped Labels.

In the presence of label flipping and gradually increasing the amount of flipped labels as expected all the performance metrics decline for the dataset with the same ratio. Figure 5.5 shows the result for the CIC-UNSW with the ratio 40-60 and clearly illustrates the decrease in all the metrics. For a constant percentage of the flipped labels, recall keeps increasing when changing the ratio while the changes in the precision and F1-score do not follow a clear trend and the changes in the accuracy value are the same as when there is no attack. Figure 5.6 shows the result for the CIC-UNSW dataset with 40% flipped labels.

5.2 FGSM-based Attack

In this section, we use the proposed FGSM-based attack to generate adversarial examples for the selected datasets. We use the FGSM-based attack with two different values for ϵ to perform the adversarial attack. In order to choose the suitable ϵ values for our detailed experiments, we first perform the attack using 6 different values including 0.1, 0.01, 0.001, 0.0001, 0.00001, and 0.000001 for ϵ. Based on the results, 0.001 and 0.01 were chosen as the preferred values for ϵ.

5.2.1 CIC-IDS2017

After training the target model on CIC-IDS2017, we started generating adversarial examples. We only use those original samples that the model detected correctly. The number of these samples is 2,777,668. As the model couldn't detect the Web Attack-SQL Injection, we don't use them for adversarial sample generation.

Table 5.4 contains the result of adversarial sample generation on the CIC-IDS2017 dataset with 0.001 and 0.01 as values for ϵ. The table shows the number of adversarial examples generated using different feature sets. The first column is the result when we use all features in the dataset.

With 0.001, the attack can't generate any adversarial examples for Infiltration, Web Attack-XSS, and Heartbleed, so we remove them from the results table. As we expected, the best result in both cases was when all the features were used. With $\epsilon = 0.001$, the attack can generate adversarial examples for 9.05% of the original samples and with $\epsilon = 0.01$ for 38.22% of the actual samples.

In both cases, the second-best set of features is the combination of *Forward, Backward* and *Flow-based* features with 8.89% for $\epsilon = 0.001$ and 23.28% for $\epsilon = 0.01$. The third and fourth-best feature sets are the same for both ϵ values. The combination of *Forward* and *Backward* features is the third, and the combination of *Forward* and *Flow-based* features is the fourth one.

The worst result for both cases is when we use *Packet header-based* features. The reason could be that the number of features in this set is the lowest, and also, almost all of the features in this set are based on the packet flags, which may not have much effect on detecting the attack types.

If we only compare the results for the main feature sets, the best results for both ϵ values is when the *Forward* features were used. This result supports our previous findings that the best feature sets combination are the ones with the *Forward* features. There is a difference in the second-best features set between two ϵ values. For value 0.001, the second-best set is *Time-based* features, but for value 0.01 is *Packet Payload-based* set. This result shows that increasing the magnitude of the perturbation increases the effect of *Packet Payload-based* features more than *Time-based* features. The third best feature set is *Backward* features.

Figure 5.7 compares the results for each attack type in the CIC-IDS2017 dataset. Each sub-figure presents the result of adversarial sample generation with ϵ values 0.001 and 0.01 for each attack, and the last sub-figure (5.7o) is the result for the whole dataset.

Table 5.4: CIC-IDS2017 results for $\epsilon = 0.01$ and $\epsilon = 0.001$.

Attack Type	ϵ	All	FWD (F)	BWD (B)	Flow (FL)	F+B	F+FL	B+FL	F+B+FL	Time (T)	Packet Header (PH)	Packet Payload (PP)	PH+PP	T+PH+PP	Samples
Benign	0.01	585398	102109	31552	31903	143058	109848	12940	172927	29308	7097	63103	65740	76836	2239270
		26.14	**4.55**	**1.40**	**1.42**	**6.38**	**4.90**	**0.57**	**7.72**	**1.30**	**0.31**	**2.81**	**2.93**	**3.43**	
	0.001	61354	56864	19117	23789	55926	58217	29973	57936	42826	1099	13967	14174	51669	
		2.73	**2.53**	**0.85**	**1.06**	**2.49**	**2.59**	**1.33**	**2.58**	**1.91**	**0.04**	**0.62**	**0.63**	**2.3**	
DDoS	0.01	126278	103981	61040	46574	120764	112552	88763	123044	67991	789	89729	91053	117519	127351
		99.15	**81.64**	**47.93**	**36.57**	**94.82**	**88.37**	**69.69**	**96.61**	**53.38**	**0.61**	**70.45**	**71.49**	**92.27**	
	0.001	46408	12641	1960	2497	39023	21059	15629	45182	12173	61	4847	5115	38635	
		36.44	**9.92**	**1.53**	**1.96**	**30.64**	**16.53**	**12.27**	**35.47**	**9.55**	**0.04**	**3.8**	**4.01**	**30.33**	
PortScan	0.01	151032	90279	135567	63349	151032	144544	141332	151032	130072	2439	72248	73116	147790	151480
		99.70	**59.59**	**89.49**	**41.82**	**99.70**	**95.42**	**93.30**	**99.70**	**85.86**	**1.61**	**47.69**	**48.26**	**97.56**	
	0.001	66222	15377	29964	566	66538	21105	32368	66606	2757	1	2823	3823	21468	
		43.71	**10.15**	**19.78**	**0.37**	**43.92**	**1393**	**21.36**	**43.97**	**1.82**	**0**	**1.86**	**2.52**	**14.17**	
Botnet	0.01	699	696	686	690	699	699	698	699	699	1	685	686	699	699
		100	**99.57**	**98.14**	**98.71**	**100**	**100**	**99.85**	**100**	**100**	**0.14**	**97.99**	**98.14**	**100**	
	0.001	676	12	3	2	671	584	559	675	13	0	2	3	672	
		96.7	**1.71**	**0.42**	**0.28**	**95.99**	**83.54**	**79.97**	**96.56**	**1.85**	**0**	**0.28**	**0.42**	**96.13**	
Infiltration	0.01	5	2	5	0	5	4	5	5	4	0	4	4	5	5
		100	**40**	**100**	**0**	**100**	**80**	**100**	**100**	**80**	**0**	**80**	**80**	**100**	
Web Attack-Brute Force	0.01	107	36	36	107	38	107	107	107	107	9	36	36	107	107
		100	**33.64**	**33.64**	**100**	**35.51**	**100**	**100**	**100**	**100**	**8.41**	**33.64**	**33.64**	**100**	
	0.001	35	9	0	0	35	35	5	35	0	0	0	0	9	
		32.71	**8.41**	**0**	**0**	**32.71**	**32.71**	**4.67**	**32.71**	**0**	**0**	**0**	**0**	**8.41**	
Web Attack-XSS	0.01	16	0	10	3	15	3	14	16	3	0	2	7	16	16
		100	**0**	**62.5**	**18.75**	**93.75**	**18.75**	**87.50**	**100**	**18.75**	**0**	**12.50**	**43.75**	**100**	
FTP-Patator	0.01	7771	7722	7732	3847	7771	7767	7771	7771	4211	14	7771	7771	7771	7771
		100	**99.36**	**99.49**	**49.50**	**100**	**99.94**	**100**	**100**	**54.18**	**0.18**	**100**	**100**	**100**	
	0.001	3810	3809	1396	3810	3810	3810	3810	3810	3810	4	3809	3809	3810	
		49.02	**49.01**	**17.96**	**49.02**	**49.02**	**49.02**	**49.02**	**49.02**	**49.02**	**0.05**	**49.01**	**49.01**	**49.02**	
SSH-Patator	0.01	2936	2830	1939	2472	2936	2935	2936	2936	2935	0	67	219	2936	2936
		100	**96.38**	**66.04**	**84.19**	**100**	**99.96**	**100**	**100**	**99.96**	**0**	**2.28**	**7.45**	**100**	
	0.001	4	0	0	0	0	1	2	3	2	0	0	0	3	

		0.13	**0**	**0**	**0**	**0**	**0.03**	**0.06**	**0.1**	**0.06**	**0**	**0**	**0**	**0.1**	
DoS GoldenEye	0.01	7281	4781	4948	598	6643	5129	5443	6676	2809	54	5365	5421	6336	9955
		73.13	**48.02**	**49.70**	**6.00**	**66.73**	**51.52**	**54.67**	**67.06**	**28.21**	**0.54**	**53.89**	**54.45**	**63.64**	
	0.001	710	104	197	22	446	129	305	573	96	7	161	167	467	
		7.13	**1.04**	**1.97**	**0.22**	**4.48**	**1.29**	**3.06**	**5.75**	**0.96**	**0.07**	**1.61**	**1.67**	**4.69**	
DoS Hulk	0.01	174567	165145	81397	58921	173204	167679	81257	173565	59457	2469	105626	105687	149214	227131
		76.85	**72.70**	**35.83**	**25.94**	**76.25**	**73.82**	**35.77**	**76.41**	**26.17**	**1.08**	**46.50**	**46.53**	**65.69**	
	0.001	71477	71185	18027	23375	71423	71226	27507	71645	27256	291	23730	23755	27311	
		31.46	**31.34**	**7.93**	**10.29**	**31.44**	**31.35**	**12.11**	**31.54**	**12.00**	**0.12**	**10.44**	**10.45**	**12.02**	
DoS Slowhttp	0.01	1356	453	462	99	3712	463	483	3734	304	5	528	529	929	5328
		25.45	**8.50**	**8.67**	**1.85**	**69.66**	**8.68**	**9.06**	**70.08**	**5.70**	**0.09**	**9.90**	**9.92**	**17.43**	
	0.001	69	49	46	7	56	50	47	56	41	1	53	53	59	
		1.29	**0.91**	**0.86**	**0.13**	**1.05**	**0.93**	**0.88**	**1.05**	**0.76**	**0.01**	**0.99**	**0.99**	**1.10**	
Dos Slowloris	0.01	4303	4076	2397	2138	4276	4262	2478	4297	2381	10	3693	4211	4245	5609
		76.71	**72.66**	**42.73**	**38.11**	**76.23**	**75.98**	**44.17**	**76.60**	**42.44**	**0.17**	**65.84**	**75.07**	**75.68**	
	0.001	688	522	147	9	659	530	170	682	181	0	253	267	499	
		12.26	**9.3**	**2.62**	**0.16**	**11.74**	**9.44**	**3.03**	**12.15**	**3.22**	**0**	**4.51**	**4.76**	**8.89**	
Heartbleed	0.01	1	0	0	0	1	0	1	1	1	0	0	0	1	10
		10	**0**	**0**	**0**	**10**	**0**	**10**	**10**	**10**	**0**	**0**	**0**	**10**	
Sum	0.01	1061750	482110	327771	210701	614154	555992	344228	646810	300282	12887	348857	354480	514404	2777668
		38.22	**17.35**	**11.80**	**7.58**	**22.11**	**20.01**	**12.39**	**23.28**	**10.81**	**0.46**	**12.55**	**12.76**	**18.51**	
	0.001	251453	160572	70857	54077	238587	176746	110375	247023	89193	1464	50645	51166	144602	
		9.05	**5.78**	**2.55**	**1.94**	**8.58**	**6.36**	**3.97**	**8.89**	**3.21**	**0.05**	**1.82**	**1.84**	**5.2**	

The sub-figure 5.7a shows the results for Benign samples. As it shows, in all cases increasing the value of the ϵ will increase the percentage of generated samples, except for the combination of *Backward*, *Flow-based* features, and *Time-based* features.

For DDoS (sub-figure 5.7b) attack we are able to generate adversarial examples for 99.15% of original samples when we use *All* of the features with $\epsilon = 0.01$. Unlike Benign samples, the results for all feature sets got better when the ϵ value is increased.

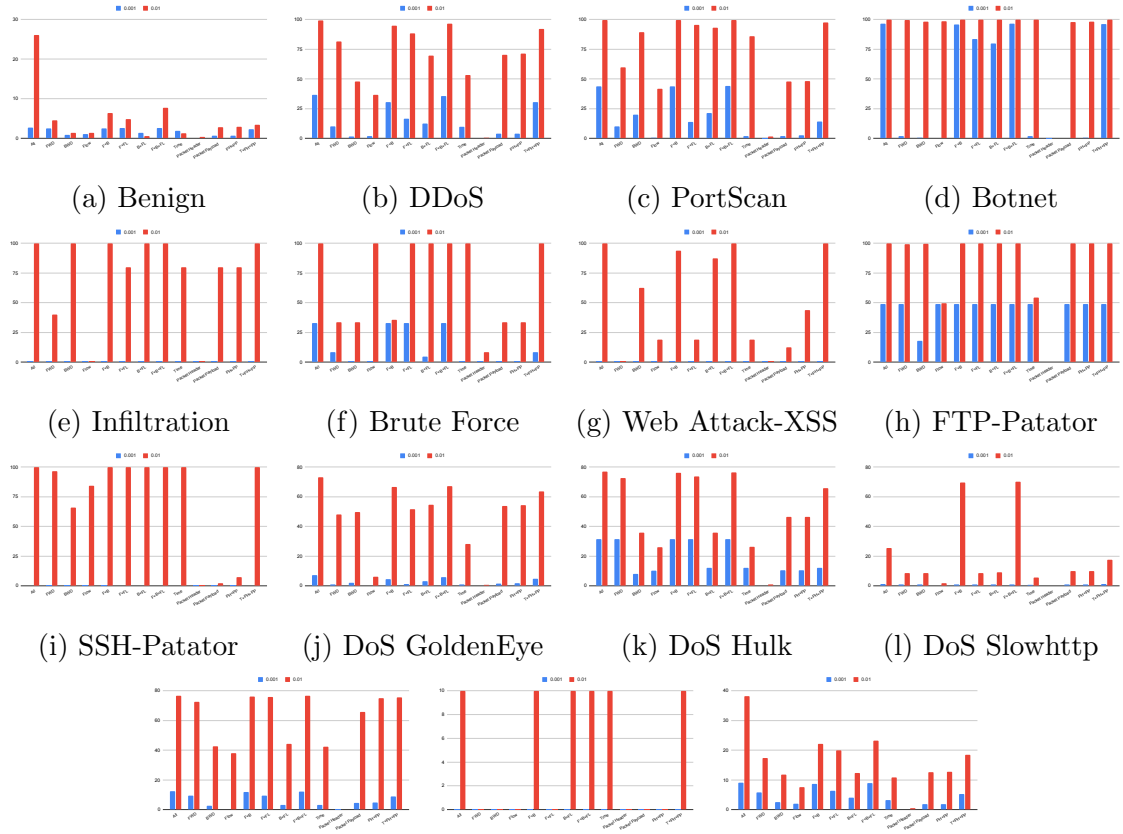
The third sub-figure (5.7c) shows the comparison for the PortScan attack. The highest percentage of generated examples is 99.7% with $\epsilon = 0.01$ for three different feature sets. This result shows we can completely fool our model without even using all of the features during adversarial sample generation.

The comparison for Botnet is shown in sub-figure 5.7d. These results show even with $\epsilon = 0.001$ in four cases; we were able to generate adversarial examples for more than 95% of the original samples, which means Botnet attack is vulnerable to adversarial attack.

Infiltration (sub-figure 5.7e), Web Attack-XSS (sub-figure 5.7g) and Heartbleed (sub-figure 5.7n) sub-figures only contain the results for $\epsilon = 0.01$ because the attack can't generate any adversarial examples with $\epsilon = 0.001$.

In 7 cases we were able to generate adversarial examples for all of the original examples with $\epsilon = 0.01$ for Web Attack-Brute Force (sub-figure 5.7f). Two interesting results are for *Flow-based* and *Time-based* features. The number of generated samples was 0 with $\epsilon = 0.001$ for these two sets, but with $\epsilon = 0.01$ the success rate was 100%.

For FTP-Patator (sub-figure 5.7h) with $\epsilon = 0.001$ the results for all of the feature sets are the same (94%) except for *Backward* and *Packet Header-based* features. Also with $\epsilon = 0.01$ the success was more than 99% for almost all of the feature sets. When we perform the adversarial attack against SSH-Patator (sub-figure 5.7i) samples with $\epsilon = 0.01$ the success rate is almost zero for all feature sets. But after increasing the



(a) Benign (b) DDoS (c) PortScan (d) Botnet

(e) Infiltration (f) Brute Force (g) Web Attack-XSS (h) FTP-Patator

(i) SSH-Patator (j) DoS GoldenEye (k) DoS Hulk (l) DoS Slowhttp

(m) DoS Slowloris (n) Heartbleed (o) Sum

Figure 5.7: CIC-IDS2017 results comparison.

value of the ϵ we got perfect results except when packet-related features were used. The next four sub-figures (5.7j, 5.7k, 5.7l, 5.7m) are for different types of DoS attacks. It seems the best result with $\epsilon = 0.001$ is for DoS Hulk and with $\epsilon = 0.01$ is for DoS GoldenEye. With both ϵ DoS Slowhttp has the worst result, with success less than 2% for all sets with $\epsilon = 0.001$.

The last sub-figure (5.7o) is the comparison for all the generated samples using different feature sets. As we mentioned earlier, the best and worst feature sets for adversarial sample generation are *All* and *Packet Header-based* features.

5.2.2 CIC-IDS2018

We do the same for the CIC-IDS2018 and use the 15,982,737 detected samples to perform the adversarial attack. Table 5.5, shows the results for CIC-IDS2018 with ϵ values 0.01 and 0.001.

For the Sql Injection and DoS Slowhttp with $\epsilon = 0.001$, no adversarial samples were generated, but for all the other attack categories we were able to generate some adversarial samples for each ϵ value. Same as CIC-IDS2018, the highest number of adversarial samples were generated using all the features: 11.38% for $\epsilon = 0.01$ and 2.74% for $\epsilon = 0.001$.

For both ϵ values the second-best feature combination is using *Forward, Backward* and *Flow-based* together same as CIC-IDS2017. The third best feature group is *Forward* and *Backward* together with 8.77% for 0.01 and 2.42% for 0.001. The worst result for both cases is when *Flow-based* features were used.

The best feature set with only one category is *Forward* features for both ϵ values. This is in line with the above-mentioned results where *Forward* features are present in all top three feature combinations.

In Figure 5.8, we presented the graph comparison of each network attack in the CIC-IDS2018 for the two ϵ values. For Benign samples in sub-figure 5.8a, the attack success rate is less than 1% for all the cases except for $\epsilon = 0.01$ and *All* the features. Also, with $\epsilon = 0.01$ the success rate is always better except with *Backward* and *Time-based* features.

For Bot (sub-figure 5.8b) attack the success rate is almost 100% in some cases for 0.01 as the ϵ value and in all the cases the success rate is better with a higher value of the ϵ, in some case up to twice as the success with the smaller value. The success rate for $\epsilon = 0.001$ is close to 50% for most of the feature groups. These results show that it is easy to generate adversarial samples for Bot attacks even with a small value for the ϵ.

Table 5.5: CIC-IDS2018 results for $\epsilon = 0.01$ and $\epsilon = 0.001$.

Attack Type	ϵ	All	FWD (F)	BWD (B)	Flow (FL)	F+B	F+FL	B+FL	F+B+FL	Time (T)	Packet Header (PH)	Packet Payload (PP)	PH+PP	T+PH+PP	Samples
Benign	0.01	428487	88604	21829	17351	68983	57913	20398	63642	17256	11813	15744	30223	46892	13379185
		3.2	**0.66**	**0.16**	**0.12**	**0.51**	**0.43**	**0.15**	**0.47**	**0.12**	**0.08**	**0.11**	**0.22**	**0.35**	
	0.001	26771	29416	3601	18410	27725	39228	17584	28604	31850	7826	3560	8563	26100	
		0.2	**0.21**	**0.02**	**0.13**	**0.2**	**0.29**	**0.13**	**0.21**	**0.23**	**0.05**	**0.02**	**0.06**	**0.19**	
Bot	0.01	285723	150031	285417	143101	285723	150578	285722	285723	146066	143321	145792	150470	285723	285868
		99.94	**52.48**	**99.84**	**50.05**	**99.94**	**52.67**	**99.94**	**99.94**	**51.09**	**50.13**	**50.99**	**52.63**	**99.94**	
	0.001	143476	142782	140064	31139	142867	142845	142838	143235	142806	142498	104174	142502	142850	
		50.18	**49.84**	**48.99**	**10.89**	**49.97**	**49.96**	**49.96**	**50.1**	**49.95**	**49.84**	**36.44**	**49.84**	**49.97**	
Infiltration	0.01	20254	19036	16898	3079	19886	19881	19685	20254	19080	17740	16654	19054	19140	20272
		99.91	**93.9**	**83.35**	**15.18**	**98.09**	**98.07**	**97.1**	**99.91**	**94.12**	**87.5**	**82.12**	**93.99**	**94.41**	
	0.001	17683	1848	431	464	3273	4200	1139	17582	2741	458	254	778	4281	
		87.22	**9.11**	**2.12**	**2.28**	**16.14**	**20.71**	**5.61**	**86.73**	**13.52**	**2.25**	**1.25**	**3.83**	**21.11**	
Brute Force-Web	0.01	193	68	66	75	193	133	133	193	133	36	36	37	133	193
		100	**35.23**	**34.19**	**38.86**	**100**	**68.91**	**68.91**	**100**	**68.91**	**18.65**	**18.65**	**19.17**	**68.91**	
	0.001	36	36	35	0	36	36	35	36	35	0	0	35	36	
		18.68	**18.65**	**18.13**	**0**	**18.65**	**18.65**	**18.13**	**18.65**	**18.13**	**0**	**0**	**18.13**	**18.65**	
Brute Force-XSS	0.01	50	49	49	49	49	50	49	50	49	49	49	49	50	121
		41.32	**40.49**	**40.49**	**40.49**	**40.49**	**41.32**	**40.49**	**41.32**	**40.49**	**40.49**	**40.49**	**40.49**	**41.32**	
	0.001	18	15	15	15	15	15	15	16	15	7	15	15	15	
		14.87	**12.39**	**12.39**	**12.39**	**12.39**	**12.39**	**12.39**	**13.22**	**12.39**	**5.78**	**12.39**	**12.39**	**12.39**	
Sql Injection	0.01	1	1	1	1	1	1	1	1	1	0	0	0	1	1
		100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**0**	**0**	**0**	**100**	
FTP BruteForce	0.01	41710	17	1052	48	2480	484	1639	6440	466	1	0	1	668	193354
		21.57	**0.008**	**0.54**	**0.024**	**1.28**	**0.25**	**0.84**	**3.33**	**0.24**	**0.0005**	**0**	**0.0005**	**0.34**	
	0.001	1	1	1	1	1	1	1	1	1	**0**	**0**	**0**	1	
		0.0005	**0.0005**	**0.0005**	**0.0005**	**0.0005**	**0.0005**	**0.0005**	**0.0005**	**0.0005**	**0**	**0**	**0**	**0.0005**	
SSH BruteForce	0.01	182	111	15	157	170	180	171	186	193	1	1	13	181	187520
		0.097	**0.059**	**0.008**	**0.083**	**0.09**	**0.096**	**0.091**	**0.099**	**0.1**	**0.0005**	**0.0005**	**0.006**	**0.096**	
	0.001	35	14	1	1	18	28	8	35	19	1	0	1	28	
		0.018	**0.007**	**0.0005**	**0.0005**	**0.009**	**0.014**	**0.004**	**0.018**	**0.01**	**0.0005**	**0**	**0.0005**	**0.014**	
DoS GoldenEye	0.01	40522	35265	34758	9365	39868	37560	25399	40498	13296	12513	13366	17266	39688	41441

		97.78	**85.09**	**83.87**	**22.59**	**96.2**	**90.63**	**61.28**	**97.72**	**32.08**	**30.19**	**32.25**	**41.66**	**95.76**	
	0.001	11501	6526	2	4	11486	7236	7105	11490	5407	6	11	6649	9862	
		27.75	**15.74**	**0.004**	**0.009**	**27.71**	**17.46**	**17.14**	**27.72**	**13.04**	**0.014**	**0.026**	**16.04**	**23.79**	
DoS Hulk	0.01	299497	291534	221401	21807	352018	292100	250548	299524	62434	103085	21761	132017	280034	461907
		64.83	**63.11**	**47.93**	**4.72**	**76.2**	**63.23**	**54.24**	**64.84**	**13.51**	**22.31**	**4.71**	**28.58**	**60.62**	
	0.001	21760	12	2	0	12160	234	99	21440	3	0	6	12	21460	
		4.71	**0.002**	**0.0004**	**0**	**2.63**	**0.05**	**0.02**	**4.64**	**0.0006**	**0**	**0.001**	**0.002**	**4.71**	
DoS Slowhttp	0.01	139890	365	48655	6	57507	445	57507	139227	469	365	0	366	78086	139890
		100	**0.26**	**34.78**	**0.004**	**41.1**	**0.31**	**41.1**	**99.52**	**0.33**	**0.26**	**0**	**0.26**	**55.81**	
DoS Slowloris	0.01	3791	3529	2364	2315	3538	3531	2371	3540	2340	2300	3348	3439	3519	10842
		34.96	**32.54**	**21.8**	**21.35**	**32.63**	**32.56**	**21.86**	**32.65**	**21.58**	**21.21**	**30.87**	**31.71**	**32.45**	
	0.001	2309	24	29	5	2308	69	65	2309	52	7	29	26	2307	
		21.29	**0.22**	**0.26**	**0.046**	**21.28**	**0.63**	**0.59**	**21.29**	**0.47**	**0.064**	**0.26**	**0.23**	**21.27**	
DDoS LOIC HTTP	0.01	286581	281699	280202	109266	299814	298945	282098	294163	155927	209373	199776	280468	286338	574523
		49.88	**49.03**	**48.77**	**19.01**	**52.18**	**51.95**	**49.1**	**51.2**	**27.14**	**36.44**	**34.77**	**48.81**	**49.83**	
	0.001	193321	137072	19515	848	165721	145403	40926	177745	22311	71062	4221	98979	129144	
		33.64	**23.85**	**3.39**	**0.14**	**28.84**	**25.3**	**7.12**	**30.93**	**3.88**	**12.36**	**0.73**	**17.22**	**22.47**	
DDoS LOIC UDP	0.01	149	70	40	52	73	84	46	90	59	18	38	54	66	1654
		9	**4.23**	**2.41**	**3.14**	**4.41**	**5.07**	**2.78**	**5.44**	**3.56**	**1.08**	**2.29**	**3.26**	**3.99**	
	0.001	19	10	4	6	12	14	10	16	11	0	4	8	16	
		1.14	**0.6**	**0.24**	**0.36**	**0.72**	**0.84**	**0.6**	**0.96**	**0.66**	**0**	**0.24**	**0.48**	**0.96**	
DDoS HOIC	0.01	272285	272250	22303	22285	272285	272285	22329	272285	22341	22303	22297	22309	272285	685966
		39.69	**39.68**	**3.25**	**3.24**	**39.69**	**39.69**	**3.25**	**39.69**	**3.25**	**3.25**	**3.25**	**3.25**	**39.69**	
	0.001	22288	22285	66	76	22285	22285	22288	22288	8305	1	54	59	22285	
		3.24	**3.24**	**0.009**	**0.011**	**3.24**	**3.24**	**3.24**	**3.24**	**1.21**	**0.0001**	**0.007**	**0.005**	**3.24**	
Sum	0.01	1819315	1142629	935050	328957	1402588	1133720	968096	1425816	440110	522918	438862	655766	1312804	15982737
		11.38	**7.14**	**5.85**	**2.05**	**8.77**	**7.09**	**6.05**	**8.92**	**2.75**	**3.27**	**2.74**	**4.1**	**8.21**	
	0.001	439218	340041	163766	50969	387909	361594	210224	424797	213556	221866	112328	257627	358685	
		2.74	**2.12**	**1.02**	**0.31**	**2.42**	**2.26**	**1.31**	**2.65**	**1.33**	**1.38**	**0.7**	**1.61**	**2.24**	

The next sub-figure is the comparison of the Infiltration attack. With $\epsilon = 0.01$, the success rate is really high and is more than 80% for all the feature groups except for *Flow-based* features. Decreasing the ϵ reduces the rate significantly for almost all the cases, for some to less than 3%, except for *All* and the combination of *Forward, Backward*, and *Flow-based* features.

The sub-figures 5.8d, 5.8e, and 5.8f are for the different Web attacks. For Brute Force-Web the success 100% for $\epsilon = 0.01$ with *All* features, *Forward* and *Backward* combination, and *Forward, Backward*, and *Flow-based* combination. With $\epsilon = 0.001$ either the success rate is almost 18% or no adversarial sample is generated. For the Brute Force-XSS attack with both ϵ values, the success is almost the same for all the feature groups. The target model can only detect one of the Sql Injection samples and the attack can not generate an adversarial sample for it with $\epsilon = 0.001$.

For FTP BruteForce the DNN correctly classifies all the samples but can only generate one adversarial sample with $\epsilon = 0.001$. Also, with $\epsilon = 0.01$ the success rate is really low and close to 0 for all the feature groups except for *All* where it is 21.57%. Sub-figure 5.8h shows the result for the SSH BruteForce attack and the success rate is even worse than FTP BruteForce. The model's detection rate is almost 100%, but the attack success rate is always almost 0.

The next four sub-figures are the comparison of different DoS attacks. On average the success rate with $\epsilon = 0.01$ is acceptable for different DoS attacks, except for Slowhttp where there is much difference between different feature groups from 100% to almost 0 for many cases. With $\epsilon = 0.001$ the success rate for DoS GoldenEye and Dos Slowloris is around 20% for some feature groups but for two other DoS attacks, it is always 0 or close to 0.

For DDoS attacks (sub-figures 5.8m, 5.8n, and 5.8o) while the model's detection is really high but the attack success rate for LOIC UDP is really lower compare to two other DDoS attacks, especially for $\epsilon = 0.01$. Between LOIC HTTP and HOIC,

(a) Benign (b) Bot (c) Infiltration (d) Brute Force-Web

(e) Brute Force-XSS (f) Sql Injection (g) FTP BruteForce (h) SSH BruteForce

(i) DoS GoldenEye (j) DoS Hulk (k) DoS Slowhttp (l) DoS Slowloris

(m) DDoS LOIC Http (n) DDoS LOIC UDP (o) DDoS HOIC (p) Sum

Figure 5.8: CIC-IDS2018 results comparison.

the success rate for both ϵ values is higher on average. Finally, The sub-figure 5.8p shows the result comparison for the total generated samples with two ϵ values and different feature groups.

5.2.3 CIC-UNSW

The number of detected samples for CIC-UNSW is 412,491 and the results of the attack are shown in Table 5.6. The target could not detect any of the Worm attack samples, therefore, there are no adversarial samples generated for this attack category. Same to the two other datasets the attack is most successful when all the features were used with 11.89% and 4.94% for the two ϵ values. The worst feature group for $\epsilon = 0.01$, same as the CIC-2018 is the *Flow-based* features, and for the

$\epsilon = 0.001$, same as the CIC-IDS2017 is the *Packet Header-based* features.

The second and third-best feature groups for 0.01 are the combination of *Forward, Backward*, and *Flow-based* features, and for 0.001 are the combination of *Time-based, Packet Header-based*, and *Packet Payload-based* features. But for $\epsilon = 0.001$ they are vice versa.

If we only consider the single feature groups for both values the top feature group is the *Time-based* features, which is completely different from the two other datasets. Since the attacks in the CIC-UNSW are not the same as the other datasets, this change in the top feature groups is expected.

To compare the success rate of the adversarial attack in different network attack categories in CIC-UNSW, Figure 5.9 shows the graph comparison of the results for each network attack.

Although the detection rate of the target model for benign traffic is really high, the adversarial attack success rate for both ϵ values is really low and below 2% for all the feature groups which is the same as the CIC-IDS2018 results. The comparison for the ϵ values is shown in sub-figure 5.9a.

The next two sub-figures (5.9b and 5.9c) are for Backdoor and DoS attacks. The number of detected samples is less than 200 for these two categories but the attack success rate is better compared to Benign traffic especially for Backdoor with 85% in most of the feature groups and $\epsilon = 0.01$.

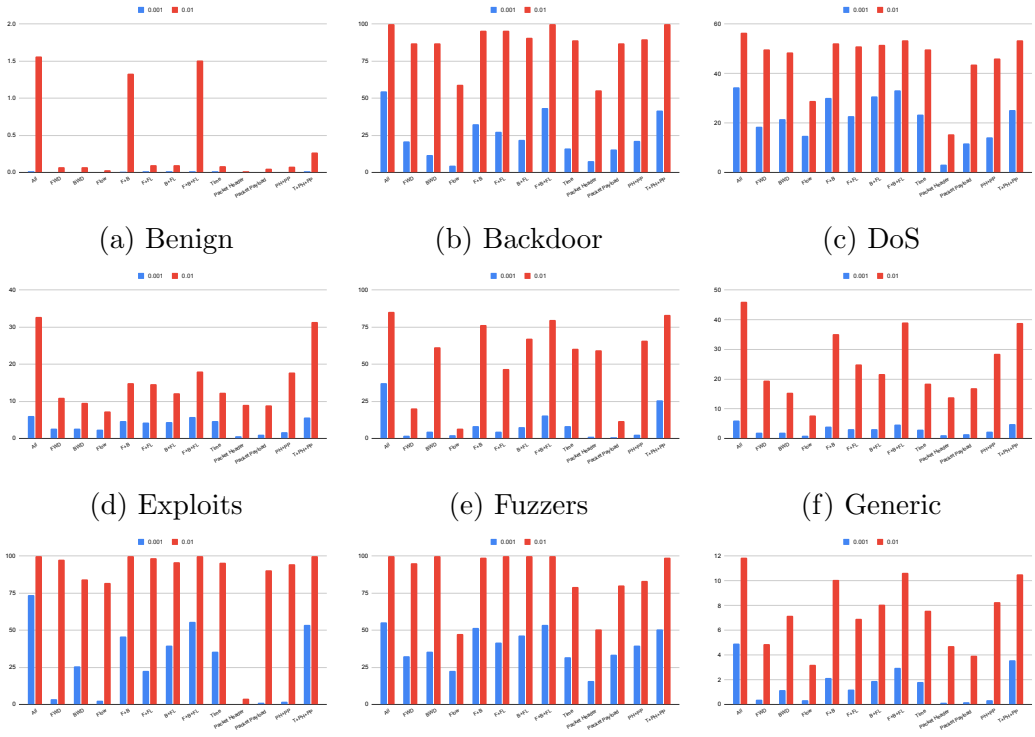
The result of the Exploits and Fuzzers attacks are shown in sub-figures 5.9d and 5.9e. The number of detected samples for both attacks is more than 20,000 but the success rate for Fuzzers is almost twice as high as that for Exploits. For both attacks, the worst feature groups are *Flow-based* and *Packet Payload-based* features.

Table 5.6: CIC-UNSW results for $\epsilon = 0.01$ and $\epsilon = 0.001$.

Attack Type	ϵ	All	FWD (F)	BWD (B)	Flow (FL)	F+B	F+FL	B+FL	F+B+FL	Time (T)	Packet Header (PH)	Packet Payload (PP)	PH+PP	T+PH+PP	Samples
Benign	0.01	5467	259	258	106	4650	351	363	5297	310	51	288	264	956	348976
		1.56	**0.074**	**0.073**	**0.03**	**1.33**	**0.1**	**0.1**	**1.51**	**0.088**	**0.014**	**0.05**	**0.075**	**0.27**	
	0.001	73	10	13	23	33	55	57	69	57	4	8	11	69	
		0.02	**0.002**	**0.003**	**0.006**	**0.009**	**0.015**	**0.016**	**0.019**	**0.016**	**0.001**	**0.002**	**0.003**	**0.019**	
Backdoor	0.01	154	134	134	91	147	147	140	154	137	85	134	138	154	154
		100	**87.01**	**87.01**	**59.09**	**95.45**	**95.45**	**90.9**	**100**	**88.96**	**55.19**	**87.01**	**89.61**	**100**	
	0.001	84	32	18	7	50	42	34	67	25	12	24	33	64	
		54.54	**20.77**	**11.68**	**4.54**	**32.46**	**27.27**	**22.07**	**43.5**	**16.23**	**7.79**	**15.58**	**21.42**	**41.55**	
DoS	0.01	92	81	79	47	85	83	84	87	81	25	71	75	87	163
		56.44	**49.69**	**48.46**	**28.83**	**52.14**	**50.92**	**51.53**	**53.37**	**49.69**	**15.33**	**43.55**	**46.01**	**53.37**	
	0.001	56	30	35	24	49	37	50	54	38	5	19	23	41	
		34.35	**18.4**	**21.47**	**14.72**	**30.06**	**22.69**	**30.67**	**33.12**	**23.31**	**3.06**	**11.65**	**14.11**	**25.15**	
Exploits	0.01	7075	2370	2082	1582	3217	3147	2624	3878	2671	1955	1916	3828	6772	21553
		32.82	**10.99**	**9.65**	**7.34**	**14.92**	**14.6**	**12.17**	**17.99**	**12.39**	**9.07**	**8.88**	**17.76**	**31.42**	
	0.001	1308	565	565	521	1009	916	940	1233	1012	131	224	371	1224	
		6.06	**2.62**	**2.62**	**2.41**	**4.68**	**4.25**	**4.36**	**5.72**	**4.69**	**0.6**	**1.03**	**1.72**	**5.67**	
Fuzzers	0.01	23735	5659	17089	1860	21272	12981	18731	22202	16812	16514	3306	18342	23189	27802
		85.37	**20.35**	**61.46**	**6.69**	**76.51**	**46.69**	**67.37**	**79.85**	**60.47**	**59.39**	**11.89**	**65.97**	**83.4**	
	0.001	10348	558	1268	602	2347	1310	2152	4291	2304	369	267	665	7189	
		37.22	**2**	**4.56**	**2.16**	**8.44**	**4.71**	**7.74**	**15.43**	**8.28**	**1.32**	**0.96**	**2.39**	**25.85**	
Generic	0.01	1126	475	377	191	860	610	529	955	450	341	412	695	949	2443
		46.09	**19.44**	**15.43**	**7.81**	**35.2**	**24.96**	**21.65**	**39.09**	**18.42**	**13.95**	**16.86**	**28.44**	**38.84**	
	0.001	149	48	49	21	97	77	79	116	74	**26**	**35**	**57**	120	
		6.09	**1.96**	**2**	**0.85**	**3.97**	**3.15**	**3.23**	**4.74**	**3.02**	**1.06**	**1.43**	**2.33**	**4.91**	
Reconnaissance	0.01	11299	11016	9505	9237	11299	11129	10816	11294	10801	429	10209	10684	11299	11299
		100	**97.49**	**84.12**	**81.75**	**100**	**98.49**	**95.72**	**99.95**	**95.59**	**3.79**	**90.35**	**94.66**	**100**	
	0.001	8316	386	2891	301	5181	2545	4491	6306	4036	22	134	221	6076	
		73.59	**3.41**	**25.58**	**2.66**	**45.85**	**22.52**	**39.74**	**55.81**	**35.72**	**0.19**	**1.18**	**1.95**	**53.77**	
Shellcode	0.01	101	96	101	48	100	101	101	101	80	51	81	84	100	101
		100	**95.04**	**100**	**47.52**	**99**	**100**	**100**	**100**	**79.2**	**50.49**	**80.19**	**83.16**	**99**	
	0.001	56	33	36	23	52	42	47	54	32	16	34	40	51	

		55.44	**32.67**	**35.64**	**22.77**	**51.48**	**41.58**	**46.53**	**53.46**	**31.68**	**15.84**	**33.66**	**39.6**	**50.49**	
Sum	0.01	49049	20090	29625	13162	41630	28549	33388	43968	31342	19451	16337	34110	43506	412491
		11.89	**4.87**	**7.18**	**3.19**	**10.09**	**6.92**	**8.09**	**10.65**	**7.59**	**4.71**	**3.96**	**8.26**	**10.54**	
	0.001	20390	1662	4875	1522	8818	5024	7850	12190	7578	585	745	1421	14834	
		4.94	**0.4**	**1.18**	**0.36**	**2.13**	**1.21**	**1.9**	**2.95**	**1.83**	**0.14**	**0.18**	**0.34**	**3.59**	

For the Generic attacks, the success rate with $\epsilon = 0.001$ is always less than 5% but for $\epsilon = 0.01$ is fairly better and more than 15% for most of the feature groups. For Reconnaissance and Shellcode, the results are compared in sub-figures 5.9g and 5.9h. For both attack categories with $\epsilon = 0.01$, the success rate is really high and more than 80% with most of the feature categories and even 100% for a number of cases. With $\epsilon = 0.001$ the attack for the Shellcode category has better success on average and always more the 30% except with *Flow-based* and *Packet Header-based* features. The last sub-figure shows the comparison of the success rate for different feature groups between two ϵ values. As mentioned before the best result is when using all the features and the interesting thing is the combination of *Time-based, Packet Header-based*, and *Packet Payload-based* features being among the top three categories.



(a) Benign (b) Backdoor (c) DoS

(d) Exploits (e) Fuzzers (f) Generic

(g) Reconnaissance (h) Shellcode (i) Sum

Figure 5.9: CIC-UNSW results comparison.

5.2.4 Discussion

In the previous sections, we provided a comprehensive description and analysis of the results of all three datasets separately. We talked about each attack group's results in the three datasets one by one and compared the effect of different feature sets and ϵ values on the adversarial examples generation results.

As mentioned before in order to select the suitable ϵ values for our experimental analysis, we did some experiments with more values for ϵ. Tables 5.7, 5.8, and 5.9 contains the results of these experiments for the three datasets. We started the experiments with $\epsilon = 0.000001$ and multiplied it by 10 each time for the next ϵ value until 0.1.

By increasing the value of ϵ by a factor of 10 each time, it is evident that the number of generated examples increases for all of the different feature groups. But there is no relation between how much we increase the values of the ϵ and how many more adversarial samples we can generate.

Furthermore, the increase between different feature groups is not equal for the same ϵ value. For example, after increasing the value of ϵ from 0.01 to 0.1 for the CIC-IDS2017 dataset, the percentage of generated adversarial samples with *Forward* features went up from 17.35% to 50.27% which is almost multiplied by 2.9 but samples with *Backward* features increased from 11.80% to 17.35% which is an increase by a factor of 1.5.

After choosing the two final ϵ values, we also did another experiment. We add a small amount to these ϵ values to evaluate the effect of these small changes. This time we use 0.0015 and 0.015 as the ϵ values. Results for these two values are also in Tables 5.7, 5.8, and 5.9. As you can see in all cases, the number of generated adversarial examples increased, sometimes by a factor of more than 2. These results show that even a small increase in the ϵ value increases the attack success rate regardless of the selected feature group for the adversarial sample generation.

Table 5.7: CIC-IDS2017 results for different ϵ values.

ϵ	**All**	**FWD (F)**	**BWD (B)**	**Flow (FL)**	**F+B**	**F+FL**	**B+FL**	**F+B+FL**	**Time (T)**	**Packet Header (PH)**	**Packet Payload (PP)**	**PH+PP**	**T+PH+PP**
0.1	1666564	1396501	482203	501943	1638785	1481315	571554	1641154	530359	194026	759937	924751	672617
	59.99	**50.27**	**17.35**	**18.07**	**58.99**	**53.32**	**20.57**	**59.08**	**19.09**	**6.98**	**27.35**	**33.29**	**24.21**
0.015	1482685	650795	326461	217063	853691	710699	377653	919083	322101	19500	412036	467719	545263
	53.37	**23.42**	**11.75**	**7.81**	**30.73**	**25.58**	**13.59**	**33.08**	**11.59**	**0.70**	**14.83**	**16.83**	**19.63**
0.01	1061750	482110	327771	210701	614154	555992	344228	646810	300282	12887	348857	354480	514404
	38.22	**17.35**	**11.80**	**7.58**	**22.11**	**20.01**	**12.39**	**23.28**	**10.81**	**0.46**	**12.55**	**12.76**	**18.51**
0.0015	266418	196423	92176	62471	260403	218363	170243	255986	112350	2363	77408	80244	189854
	9.59	**7.07**	**3.31**	**2.24**	**9.37**	**7.86**	**6.12**	**9.21**	**4.04**	**0.08**	**2.78**	**2.88**	**6.83**
0.001	251453	160572	70857	54077	238587	176746	110375	247023	89193	1464	50645	51166	144602
	9.05	**5.78**	**2.55**	**1.94**	**8.58**	**6.36**	**3.97**	**8.89**	**3.21**	**0.05**	**1.82**	**1.84**	**5.2**
0.0001	37195	15055	5461	5674	21066	23628	10197	34413	15626	81	5332	5367	29851
	1.33	**0.54**	**0.19**	**0.20**	**0.75**	**0.85**	**0.36**	**1.23**	**0.56**	**0.002**	**0.19**	**0.19**	**1.07**
0.00001	2878	1960	932	956	2217	2551	1694	2762	1484	3	1114	1117	2269
	0.10	**0.07**	**0.03**	**0.03**	**0.07**	**0.09**	**0.06**	**0.09**	**0.05**	**0.0001**	**0.04**	**0.04**	**0.08**
0.000001	447	202	34	46	213	418	77	434	204	1	24	24	398
	0.01	**0.007**	**0.001**	**0.001**	**0.007**	**0.01**	**0.002**	**0.01**	**0.007**	**0.00003**	**0.0008**	**0.0008**	**0.01**

Table 5.8: CIC-IDS2018 results for different ϵ values.

ϵ	**All**	**FWD (F)**	**BWD (B)**	**Flow (FL)**	**F+B**	**F+FL**	**B+FL**	**F+B+FL**	**Time (T)**	**Packet Header (PH)**	**Packet Payload (PP)**	**PH+PP**	**T+PH+PP**
0.1	5263407	5306473	2004686	2877603	4519288	6227844	3111591	4354592	1799097	1733240	3271036	3692281	4021307
	32.93	**33.2**	**12.54**	**18**	**28.27**	**38.96**	**19.46**	**27.24**	**11.25**	**10.84**	**20.46**	**23.1**	**25.16**
0.015	2547905	1237964	1004758	367370	1700320	1538907	1464639	1841323	916523	689221	559865	1224294	1430428
	15.94	**7.74**	**6.28**	**2.29**	**10.63**	**9.62**	**9.16**	**11.52**	**5.73**	**4.31**	**3.5**	**7.66**	**8.94**
0.01	1819315	1142629	935050	328957	1402588	1133720	968096	1425816	440110	522918	438862	655766	1312804
	11.38	**7.14**	**5.85**	**2.05**	**8.77**	**7.09**	**6.05**	**8.92**	**2.75**	**3.27**	**2.74**	**4.1**	**8.21**
0.0015	522066	373195	199134	151342	478654	424970	274865	499879	241297	244885	154270	306145	390181
	3.26	**2.33**	**1.24**	**0.94**	**2.99**	**2.65**	**1.71**	**3.12**	**1.5**	**1.53**	**0.96**	**1.91**	**2.44**
0.001	439218	340041	163766	50969	387909	361594	210224	424797	213556	221866	112328	257627	358685
	2.74	**2.12**	**1.02**	**0.31**	**2.42**	**2.26**	**1.31**	**2.65**	**1.33**	**1.38**	**0.7**	**1.61**	**2.24**
0.0001	138268	59105	1072	1869	116293	85483	3095	129002	3585	5252	779	6584	18884
	0.86	**0.36**	**0.006**	**0.01**	**0.72**	**0.53**	**0.01**	**0.8**	**0.02**	**0.03**	**0.004**	**0.04**	**0.11**
0.00001	2008	1660	107	198	1795	1880	288	1971	360	604	88	674	1018
	0.01	**0.01**	**0.0007**	**0.001**	**0.01**	**0.01**	**0.001**	**0.01**	**0.002**	**0.003**	**0.0006**	**0.004**	**0.006**
0.000001	231	194	12	18	205	217	34	228	39	68	12	76	126
	0.001	**0.001**	**0.0001**	**0.0001**	**0.001**	**0.001**	**0.0002**	**0.001**	**0.0002**	**0.0004**	**0.0001**	**0.0005**	**0.0008**

Table 5.9: CIC-UNSW results for different ϵ values.

ϵ	**All**	**FWD (F)**	**BWD (B)**	**Flow (FL)**	**F+B**	**F+FL**	**B+FL**	**F+B+FL**	**Time (T)**	**Packet Header (PH)**	**Packet Payload (PP)**	**PH+PP**	**T+PH+PP**
0.1	333352	132826	104813	57299	277460	186443	130757	322208	118205	47158	68923	80318	228790
	80.81	**32.2**	**25.4**	**13.89**	**67.26**	**45.19**	**31.69**	**78.11**	**28.65**	**11.43**	**16.7**	**19.47**	**55.46**
0.015	72236	25183	35036	14759	46570	37234	42271	53462	35056	24371	19361	40413	49729
	17.51	**6.1**	**8.49**	**3.57**	**11.28**	**9.02**	**10.24**	**12.96**	**8.49**	**5.9**	**4.69**	**9.79**	**12.05**
0.01	49049	20090	29625	13162	41630	28549	33388	43968	31342	19451	16337	34110	43506
	11.89	**4.87**	**7.18**	**3.19**	**10.09**	**6.92**	**8.09**	**10.65**	**7.59**	**4.71**	**3.96**	**8.26**	**10.54**
0.0015	24505	2851	8032	3208	13916	10727	11050	21627	11532	953	1470	3340	22949
	5.49	**0.69**	**1.94**	**0.77**	**3.37**	**2.6**	**2.67**	**5.24**	**2.97**	**0.23**	**0.35**	**0.8**	**5.56**
0.001	20390	1662	4875	1522	8818	5024	7850	12190	7578	585	745	1421	14834
	4.94	**0.4**	**1.18**	**0.36**	**2.13**	**1.21**	**1.9**	**2.95**	**1.83**	**0.14**	**0.18**	**0.34**	**3.59**
0.0001	783	297	328	278	523	472	512	694	577	71	81	127	702
	0.18	**0.072**	**0.079**	**0.06**	**0.12**	**0.11**	**0.12**	**0.16**	**0.13**	**0.017**	**0.019**	**0.03**	**0.17**
0.00001	126	43	45	42	81	71	77	113	100	11	13	22	112
	0.03	**0.01**	**0.01**	**0.01**	**0.019**	**0.017**	**0.018**	**0.027**	**0.024**	**0.002**	**0.003**	**0.005**	**0.027**
0.000001	19	5	7	5	13	12	13	18	15	2	1	3	17
	0.004	**0.001**	**0.001**	**0.001**	**0.003**	**0.002**	**0.003**	**0.004**	**0.003**	**0.0005**	**0.0002**	**0.0007**	**0.004**

By comparing all of the findings from three datasets, we are not able to make a general conclusion on the most influential feature sets for an adversarial attack. For example, we expect to have the best results when using *All* of the features, but for DoS Slowhttp in CIC-IDS2017, and DoS Hulk and DDoS LOIC HTTP in CIC-IDS2018 they do not get the best result with *All* of the features.

The next key finding is that the rankings of the top feature groups for the three datasets are almost the same. Since the CIC-IDS2017 and CIC-IDS2018 have the same set of attacks the similarity between these two is more than CIC-UNSW. The first five best feature sets are the same for the three datasets with a slight difference in their ranking for different ϵ values. Also, the worst feature groups for different ϵ values are either *Flow-based* or *Packet Header-based* features. This means it would be better to focus on these feature sets for evaluating and enhancing adversarial attack performance in network intrusion detection and network traffic classification domains.

On average, it seems that the CIC-IDS2018 and CIC-UNSW datasets are more robust to the proposed adversarial attack than CIC-IDS2017. With $\epsilon = 0.001$, the average percentage of generated adversarial samples are 1.69% and 1.62%, for CIC-IDS2018 and CIC-UNSW are lower compared to 4.55% for CIC-IDS2017. Same with $\epsilon = 0.01$, where the average success rate is 15.98% for CIC-IDS2017 and almost twice the rate for the other datasets.

5.3 Saliency Map-based Attack

After training the DNN model, we use the proposed saliency map-based attack to generate adversarial examples for the selected datasets. In the experiments, the attack is performed with 1, 2, and 3 as the number of features and 0.1, 0.01, and 0.001 as different ϵ values. To compare the performance of this attack we use the

Table 5.10: CIC-IDS2017 results comparison.

ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
0.1	1666564	375970	489274	516815
	59.99	**13.53**	**17.61**	**18.60**
0.01	1061750	136595	173568	240890
	38.22	**4.91**	**6.24**	**8.67**
0.001	251453	52924	66430	72202
	9.05	**1.9**	**2.39**	**2.59**

result from the FGSM-based attack presented in the previous section.

5.3.1 CIC-IDS2017

First, we start our experiment with the CIC-IDS2017 datasets. The number of correctly classified samples is 2,777,668, and we only use them for performing the adversarial attack.

Table 5.10 shows the results of the attack. The first column is the number of generated samples when we use all the features with the FGSM method to make an adversarial attack. As it is evident, the best performance is when three features with $\epsilon = 0.1$ were used for generating the samples.

In Table 5.11, the result of the FGSM-based attack is shown for the CIC-IDS2017. These two tables show that the new method is better than the previous method while changing fewer features. For example, when one feature is used with all three values for ϵ the attack is more successful than using *Packet header-based* features which contains 14 features. Also, when we use three features the results are better or really close compared to results for *Backward*, *Flow-based*, and *Time-based* feature groups with 22, 15, or 27 features.

In Tables 5.12 the more detailed results for CIC-IDS2017 are shown. In this table, we show the top five best features with the highest number of generated samples for each ϵ value.

Table 5.11: CIC-IDS2017 results for features selected based on their nature.

ϵ	**F(24)**	**B(22)**	**FL(15)**	**T(27)**	**PH(14)**	**PP(16)**
0.1	1396501	482203	501943	530359	194026	759937
	50.27	**17.35**	**18.07**	**19.09**	**6.98**	**27.35**
0.01	482110	327771	210701	300282	12887	348857
	17.35	**11.80**	**7.58**	**10.81**	**0.46**	**12.55**
0.001	160572	70857	54077	89193	1464	50645
	5.78	**2.55**	**1.94**	**3.21**	**0.05**	**1.82**

Table 5.12: CIC-IDS2017 top 5 best feature groups for different ϵ values.

ϵ	**1 Feature**		**2 Features**		**3 Features**	
	Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
0.1	144408	Flow IAT Min	178888	Flow IAT Min Fwd IAT Total	105054	Flow IAT Min Fwd IAT Total Fwd Packets/s
	88979	Fwd Act Data Pkts	83551	Subflow Fwd Packets Fwd Act Data Pkts	83514	total Fwd Packets Subflow Fwd Packets Fwd Act Data Pkts

	35250	Fwd IAT Total	30178	Fwd Packets/s Fwd Act Data Pkts	39899	Fwd Packet Length Std Flow IAT Min Fwd IAT Total
	29513	Subflow Bwd Packets	27470	Flow IAT Std Fwd Act Data Pkts	28313	Fwd Packets/s Fwd Act Data Pkts Active Std
	20638	Flow IAT Std	20229	Flow duration Fwd Act Data Pkts	27846	total Bwd Packets Flow IAT Min Subflow Bwd Packets
0.01	59490	Flow IAT Min	81639	Flow IAT Min Fwd IAT Total	54402	Flow IAT Min Fwd IAT Total Fwd Packets/s
	29084	Subflow Bwd Packets	20208	Flow duration Fwd Act Data Pkts	27729	total Bwd Packets Flow IAT Min Subflow Bwd Packets
	25591	Fwd IAT Total	16861	total Bwd Packets Subflow Bwd Packets	24912	Fwd Packet Length Std Flow IAT Min Fwd IAT Total

	7551	Flow IAT Std	12298	Flow IAT Min Subflow Bwd Packets	24830	Flow IAT Mean Flow IAT Min Fwd IAT Total
	7273	total Fwd Packets	7605	Flow IAT Std Fwd IAT Mean	23119	Flow IAT Std Fwd IAT Mean Fwd Act Data Pkts
0.001	37623	Flow IAT Min	44118	Flow IAT Min Fwd IAT Total	24856	Flow IAT Mean Fwd IAT Min Fwd IAT Total
	6816	total Fwd Packets	7439	total Fwd Packets Subflow Fwd Packets	12675	Flow IAT Min Fwd IAT Total Init_Win_bytes_forward
	3027	Fwd Act Data Pkts	4826	total Bwd Packets Flow IAT Min	7951	Fwd Packet Length Std Flow IAT Min Fwd IAT Total

	2050	Fwd IAT Total	3959	Flow IAT Std Fwd IAT Mean	6012	Flow IAT Min Fwd IAT Total Fwd Packets/s
	1549	Flow IAT Std	3031	total Fwd Packets Fwd Act Data Pkts	5810	total Fwd Packets Subflow Fwd Packets Fwd Act Data Pkts

Table 5.13: CIC-IDS2017 top 3 best feature sets for each ϵ value.

0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
178888	Flow IAT Min Fwd IAT Total	81639	Flow IAT Min Fwd IAT Total	44118	Flow IAT Min Fwd IAT Total
144408	Flow IAT Min	59490	Flow IAT Min	37623	Flow IAT Min
105054	Flow IAT Min Fwd IAT Total Fwd Packets/s	54402	Flow IAT Min Fwd IAT Total Fwd Packets/s	24856	Fwd IAT Min Fwd IAT Total Flow IAT Mean

For $\epsilon = 0.1$ the best single feature is *Flow IAT Min* with 144,408 generated samples. For two features, the combination of *Flow IAT Min* and *Fwd IAT Total* with 178,888 generated samples, and for the three-feature attack, *Flow IAT Min*, *Fwd IAT Total* and *Fwd Packets/s* together with 105,054 generated samples are the best.

With $\epsilon = 0.01$, while the number of generated samples decreased due to the smaller value of the ϵ, still the best single feature is *Flow IAT Min*, the best two features combination are *Flow IAT Min* and *Fwd IAT Total*, and for three features combination the best are *Flow IAT Min*, *Fwd IAT Total*, and *Fwd Packets/s*.

The best-selected features for $\epsilon = 0.001$ are almost the same as the best features of the two other ϵ values. The best single feature is *Flow IAT Min*. The combination of *Flow IAT Min* and *Fwd IAT Total* is the best two-feature group and the best with three features is the combination of *Flow IAT Mean*, *Flow IAT Min*, and *Fwd IAT Total*.

We select the top three best feature groups for each ϵ value and present them in Table 5.13. As shown for all three ϵ values, the highest number of generated samples is when *Flow IAT Min* and *Fwd IAT Total* are used together. The second best-selected feature is also the same for all the ϵ values, with *Flow IAT Min*. There is a slight difference in the third-best features group. For all three ϵs the third-best feature group contains three features: the same for 0.1 and 0.01, but one of the features is different for 0.001. For 0.1 and 0.01 the selected features are *Flow IAT Min*, *Fwd IAT Total* and *Fwd Packets/s* but for 0.001 instead of *Fwd Packets/s*, *Fwd IAT Mean* is selected.

5.3.2 CIC-IDS2018

For the CIC-IDS2018, our model can classify 15,982,736 of the samples and use these records to generate adversarial samples.

In Table 5.14, the results of the attack based on our proposed method are shown for

Table 5.14: CIC-IDS2018 results comparison

ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
0.1	5263407	1422321	1759628	2455384
	32.93	**8.89**	**11.00**	**15.36**
0.01	1819315	223747	356181	384268
	11.38	**1.39**	**2.22**	**2.4**
0.001	439218	158953	177312	182429
	2.74	**0.99**	**1.1**	**1.14**

the CIC-IDS2018. For purposes of comparison, we also include the results when all the features are used. As expected, the highest numbers of generated samples are when the bigger value for ϵ and more features are used. The number of generated samples with $\epsilon = 0.1$ and three features is 2,455,384.

To compare the proposed method results with the number of generated samples when features are grouped based on their nature, the results for that attack are shown in Table 5.15. Comparing these two tables presents exciting findings. When 3 features were used with $\epsilon = 0.1$ the attack is successful for 15.36% of the samples and it outperforms attacks using *Backward*, *Time-based*, and *Packet Header-based* features with 22, 27, and 14 number of features. With $\epsilon = 0.01$ and two features, we get results better than *Flow-based* features and really close to *Time-based* and *Packet Payload-based* features. And at last, with $\epsilon = 0.001$ and selecting one feature we still are able to outperform *Flow-based* and *Packet Payload-based* features and get almost as good performance as *Backward* features.

In the next table (Table 5.16) the top five best-selected features with different values of ϵ are shown for the CIC-IDS2018 dataset.

The results for $\epsilon = 0.1$ shows that the top single feature is *Fwd Header Length* with 483,241 successful attempts. The combination of *Fwd Header Length* and *total Fwd Packets* is the best two-feature group with 433,771 adversarial samples. And the top three-feature combination is *Fwd Header Length*, *total Fwd Packets*, and *Subflow Fwd Packets* together with 586,626 generated samples.

Table 5.15: CIC-IDS2018 results for features selected based on their nature

ϵ	**F(24)**	**B(22)**	**FL(15)**	**T(27)**	**PH(14)**	**PP(16)**
0.1	5306473	2004686	2877603	1799097	1733240	3271036
	33.2	**12.5**	**18.00**	**11.25**	**10.84**	**20.46**
0.01	1142629	935050	328957	440110	522918	438862
	7.14	**5.85**	**2.05**	**2.75**	**3.27**	**2.74**
0.001	340041	163766	50696	213556	221866	112328
	2.12	**1.02**	**0.31**	**1.33**	**1.38**	**0.7**

Table 5.16: CIC-IDS2018 top 5 best feature groups for different ϵ values.

ϵ	**1 Feature**		**2 Features**		**3 Features**	
	Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
0.1	483241	Fwd Header Length	433771	Fwd Header Length Total Fwd Packets	586626	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	464612	act_data_pkt_fwd	148266	Fwd Packet Length Std act_data_pkt_fwd	204103	Flow Packets/s Fwd Packets/s act_data_pkt_fwd

	157896	Subflow Bwd Packets	140708	Fwd Header Length act_data_pkt_fwd	134622	Fwd Header Length Subflow Bwd Packets act_data_pkt_fwd
	87685	Bwd Header Length	131098	Fwd Packets/s act_data_pkt_fwd	132157	Fwd Packets Length Std Packets Length Std Packets Length Variance
	48999	Fwd Packet Length Std	113596	Packet Length Variance act_data_pkt_fwd	129758	Bwd Packets Length Min Flow IAT Std Fwd IAT Std
0.01	166746	Fwd Header Length	153576	Fwd Header Length Total Fwd Packets	185881	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	42379	Fwd Packets/s	92181	Bwd Header Length Subflow Bwd Packets	103111	Bwd Header Length Total Bwd Packets Subflow Bwd Packets
	4001	Subflow Fwd Packets	38501	Fwd Packets/s Flow Packets/s	29711	Bwd Packet Length Min Flow Packets/s Fwd Packets/s

	2201	Bwd IAT Min	22359	Fwd Header Length Subflow Fwd Packets	8070	Flow Bytes/s Fwd Packets/s Active Min
	2087	Total Fwd Packets	10862	Total Backward Packets Bwd Header Length	5891	Total Fwd Packets Flow Bytes/s Fwd Header Length
0.001	149334	Fwd Header Length	142512	Fwd Header Length Total Fwd Packets	162831	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	6471	Fwd Packets/s	22255	Fwd Header Length Subflow Fwd Packets	5517	Flow Packets/s Fwd Packets/s act_data_pkt_fwd
	1460	Subflow Fwd Packets	7345	Fwd Packets/s Flow Packets/s	4958	Total Fwd Packets Fwd Header Length Fwd Packets/s

	685	Flow Packets/s	2024	Subflow Fwd Packets Total Fwd Packets	2618	Bwd IAT Min Fwd Packets/s Active Min
	494	Total Fwd Packets	1865	Fwd Packets/s Active Min	1943	Flow Packets/s Bwd IAT Min Fwd Packets/s

Table 5.17: CIC-IDS2018 top 3 best feature sets for each ϵ value.

0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
586626	Fwd Header Length Total Fwd Packets Subflow Fwd Packets	185881	Fwd Header Length Total Fwd Packets Subflow Fwd Packets	162831	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
483241	Fwd Header Length	166746	Fwd Header Length	149334	Fwd Header Length
464612	act_data_pkt_fwd	153576	Fwd Header Length Total Fwd Packets	142512	Fwd Header Length Total Fwd Packets

Decreasing the value of ϵ does not change the top selected features. Still, the best single feature is *Fwd Header Length*, two feature combinations are *Fwd Header Length* and *total Fwd Packets*, and the combination of *Fwd Header Length*, *total Fwd Packets* and *Subflow Fwd Packets* is the best three-feature group for both ϵ 0.01 and 0.001. Table 5.17 shows the top three feature groups for each value of ϵ. The highest number of generated samples for all three values of ϵ is when three features including *Fwd Header Length*, *total Fwd Packets* and *Subflow Fwd Packets* are used for adversarial attack. When the *Fwd Header Length* is used we got the second-best results for all the ϵ values. The third-best selected feature for $\epsilon = 0.1$ is *act_data_pkt_fwd* and is different from features for ϵ values of 0.01 and 0.001 which are *Fwd Header Length* and *total Fwd Packets*.

5.3.3 CIC-UNSW

For CIC-UNSW, the saliency map-based attack results are presented in Table 5.18 and FGSM-based results are shown in Table 5.19. As expected and the same as the two other datasets increasing the value of the ϵ and the number of changed features, increases the attack's success rate. The success rate for $\epsilon = 0.1$ and three features is 13.08% while it is 0.12% with $\epsilon = 0.001$ and one feature.

Table 5.18: CIC-UNSW results comparison.

ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
0.1	333352	41541	48513	53959
	80.81	**10.07**	**11.76**	**13.08**
0.01	49049	19152	22285	26427
	11.89	**4.64**	**6.13**	**6.41**
0.001	20390	483	1145	1408
	4.94	**0.12**	**0.28**	**0.34**

Table 5.19: CIC-UNSW results for features selected based on their nature.

ϵ	**F(24)**	**B(22)**	**FL(15)**	**T(27)**	**PH(14)**	**PP(16)**
0.1	132826	104813	57299	118205	47158	68923
	32.2	**25.4**	**13.89**	**28.65**	**11.43**	**16.7**
0.01	20090	29625	13162	31342	19451	16337
	4.87	**7.18**	**3.19**	**7.59**	**4.71**	**3.96**
0.001	1662	4875	1522	7578	585	745
	0.4	**1.18**	**0.36**	**1.83**	**0.14**	**0.18**

Table 5.20: CIC-UNSW top 5 best feature groups for different ϵ values.

ϵ	**1 Feature**		**2 Features**		**3 Features**	
	Samples	**Selected Featrues**	**Samples**	**Selected Featrues**	**Samples**	**Selected Featrues**
0.1	19758	Bwd Packets/s	24525	Bwd Packets/s PSH Flag Count	9650	Fwd IAT Min Bwd Packets/s PSH Flag Count
	8830	PSH Flag Count	3903	Bwd IAT Std Bwd IAT Min	4380	Bwd IAT Std Bwd IAT Min Down/Up Ratio

	3964	Bwd IAT Min	2391	Flow IAT Mean Bwd IAT Mean	3829	Bwd IAT Min Bwd Packets/s PSH Flag Count
	2005	Down/Up Ratio	2057	Fwd Packet Length Max PSH Flag Count	3363	Flow IAT Mean Flow IAT Std Bwd IAT Mean
	1346	Bwd IAT Std	1888	Bwd IAT Min PSH Flag Count	3197	Bwd Packets/s PSH Flag Count Down/Up Ratio
0.01	10610	Bwd Packets/s	17702	Bwd Packets/s PSH Flag Count	9641	Fwd IAT Min Bwd Packets/s PSH Flag Count
	3709	Bwd IAT Min	3923	Bwd IAT Std Bwd IAT Min	4378	Bwd IAT Std Bwd IAT Min Down/Up Ratio
	2315	PSH Flag Count	644	Bwd IAT Min Down/Up Ratio	2890	Bwd IAT Min Bwd Packets/s PSH Flag Count

	898	Bwd IAT Std	502	Flow IAT Mean Flow IAT Std	1739	Bwd Packets/s PSH Flag Count Down/Up Ratio
	466	Fwd IAT Mean	416	Fwd IAT Mean Fwd IAT Std	1231	Bwd IAT Total Bwd Packets/s PSH Flag Count
0.001	263	Bwd Packets/s	769	Bwd Packets/s PSH Flag Count	615	Fwd IAT Min Bwd Packets/s PSH Flag Count
	59	PSH Flag Count	185	Bwd Packet Length Std Bwd IAT Std	185	Bwd Packet Length Std Bwd IAT Std Bwd IAT Min
	50	Bwd IAT Std	35	Flow IAT Mean Flow IAT Std	104	Bwd Packet Length Max Bwd Packets/s PSH Flag Count

	37	Bwd Packet Length Std	16	Flow IAT Std Bwd IAT Total	78	Bwd IAT Min Bwd Packets/s PSH Flag Count
	18	Down/Up Ratio	10	Flow IAT Mean Down/Up Ratio	57	Bwd Packet Length Std Fwd IAT Mean Bwd IAT Std

Table 5.21: CIC-UNSW top 3 best feature sets for each ϵ value.

0.1		**0.01**		**0.001**	
Samples	**Selected Featrues**	**Samples**	**Selected Featrues**	**Samples**	**Selected Featrues**
24525	Bwd Packets/s PSH Flag Count	17702	Bwd Packets/s PSH Flag Count	769	Bwd Packets/s PSH Flag Count
19758	Bwd Packets/s	10610	Bwd Packets/s	615	Fwd IAT Min Bwd Packets/s PSH Flag Count
9650	Fwd IAT Min Bwd Packets/s PSH Flag Count	9641	Fwd IAT Min Bwd Packets/s PSH Flag Count	263	Bwd Packets/s

Comparing the results for the FGSM-based attack with the new attack success rate reveals interesting outcomes. With the smallest value of the ϵ and only one feature, we get the success really close to when *Packet Header-based* features were used. By increasing the changed feature to three, we are able to outperform *Packet Header-based* and *Packet Payload-based* and get really close to *Forward* and *Flow-based* features' success rate. With three features and $\epsilon = 0.01$, the success rate is 6.41% and the new method can defeat FGSM-based in all the cases except for *Backward* and *Time-based* features. Finally, with the biggest value of the ϵ and three features, it is still possible to outperform *Packet Header-based* features result.

The details of the selected features are presented in Table 5.20. The top single feature for $\epsilon = 0.1$ is *Bwd Packets/s* with 19,758 generated samples. The *Bwd Packets/s* and *PSH Flag Count* together are the best two-feature group with 24,525 successful adversarial samples and adding *Fwd IAT Min* makes the best three-feature group with 9650 samples.

With two other values of the ϵ, although they are smaller, the best one-feature, two-feature, and three-feature groups are still the same as before. The best single feature is *Bwd Packets/s*, the two-feature combination is *Bwd Packets/s* and *PSH Flag Count*, and the combination of *Bwd Packets/s*, *PSH Flag Count*, and *Fwd IAT Min* is the best three-feature group.

The top three feature groups for each ϵ are presented in Table 5.21. Same as the CIC-IDS2017 for all three ϵ values, the most successful feature group includes two features: *Bwd Packets/s* and *PSH Flag Count*. The second and third best feature groups are the same for $\epsilon = 0.1$ and $\epsilon = 0.01$ but their places have changed for $\epsilon = 0.001$.

5.3.4 Discussion

In three previous sections, we presented the saliency map-based attack results for the three datasets and went through their details. In this section, we are going to do a deeper analysis of the presented results based on our proposed evaluation model.

5.3.4.1 Parameter Evaluation

Previously, the detailed results of the attack with three different values of ϵ and numbers of features are presented. The main findings that were expected are that increasing the number of features and value of ϵ improves the effectiveness of the attack.

The other significant result is that while the highest number of features used in the attacks is three, in many cases, the number of generated samples is either close to or higher than the other types of attacks that use a more significant number of features. Changing the value of ϵ does not affect the most successful feature groups in CIC-IDS2018, only changes one feature for CIC-IDS2017, and changes the ranking of two feature groups for CIC-UNSW. As you can see in Tables 5.13, 5.17, and 5.21 the selected features in all three columns are almost the same. The best group of features has two members for CIC-IDS2017 and CIC-UNSW, but three members for CIC-IDS2018. This means that the most successful feature groups do not always have more features.

5.3.4.2 Distance Analysis

The l_0 distance will count the number of changed features between two samples and does not take into account the values of ϵ. On the other hand, the l_∞ distance only measures the maximum changes between different features of the two samples, which is always equal to ϵ. The l_2 distance yields the standard Euclidean distance using the following equation:

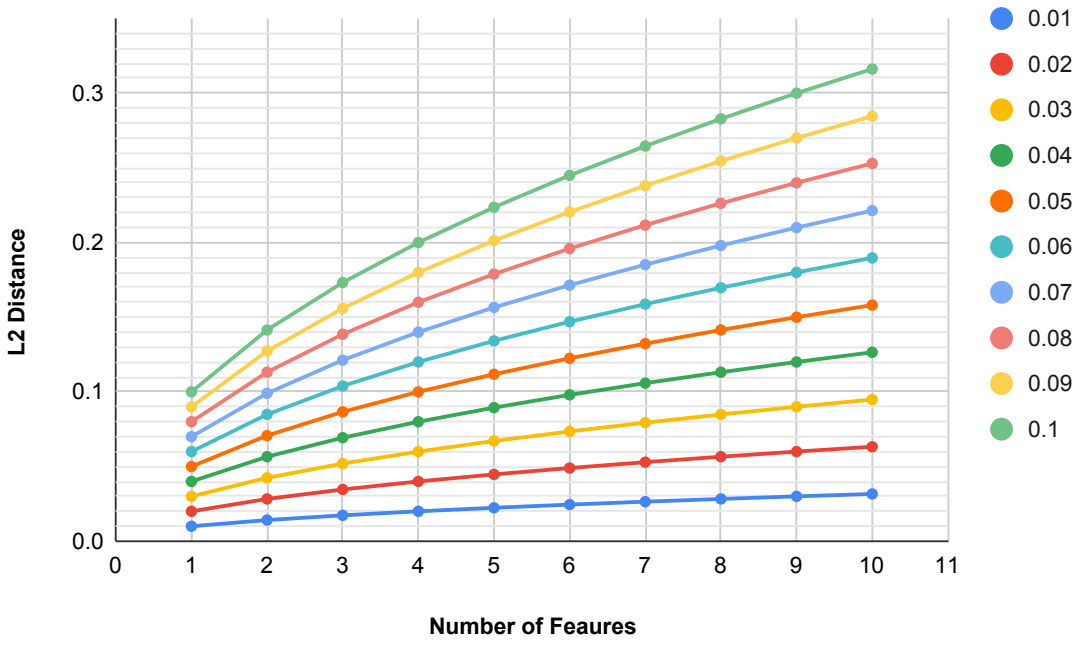


Figure 5.10: Distance analysis based on m and ϵ

$$l_2 = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + ... + (x_n - x_n')^2}. \tag{5.1}$$

The difference between features in the above equation is ϵ and the number of changed features in m; therefore, the l_2 distance is equal to $\epsilon\sqrt{m}$.

In Figure 5.10, we plot the l_2 distance for ϵ between 0.01 and 0.1 with the number of features from 1 to 10. Figure 5.10 shows that by increasing the value of the ϵ the effect of the number of features increases. For example, with $\epsilon = 0.01$ and $m = 10$ the distance is around 0.03 and is the same as the distance with $\epsilon = 0.02$ and $m = 2$, but the distance for $\epsilon = 0.09$ and $m = 10$ is almost 0.29 and is close to the distance with $\epsilon = 0.1$ and $m = 8$.

5.3.4.3 The DNN Model's Performance

In Table 5.22, the performance of the DNN model on datasets containing adversarial samples is presented based on F1-score (F1), Precision (PC), and Recall (RC).

Table 5.22: DNN model performance on datasets with adversarial samples.

Attack Parameters		CIC-IDS2017			CIC-IDS2018			CIC-UNSW		
m	ϵ	**F1**	**PC**	**RC**	**F1**	**PC**	**RC**	**F1**	**PC**	**RC**
No Attack		68.45	88.96	65.36	81.23	96.04	79.44	48.22	61.64	47.08
3	**0.1**	16.57	20.93	14.78	28.14	41.84	27.84	18.39	18.74	18.43
	0.01	51.86	78.18	45.45	73.95	93.75	68.84	33.36	40.41	32.39
	0.001	64.35	86.44	60.31	77.75	94.87	74.91	47.21	59.49	46.20
2	**0.1**	18.01	25.45	15.99	39.36	58.21	35.64	19.53	20.43	19.5
	0.01	55.14	80.09	49.48	74.94	94.58	69.98	34.36	40.96	33.36
	0.001	64.66	86.81	60.59	78.39	94.90	75.58	47.56	60.36	46.48
1	**0.1**	26.56	42.49	22.64	44.42	60.64	40.12	23.59	27.50	22.85
	0.01	58.59	80.09	54.30	76.31	84.64	72.67	38.22	46.45	37.06
	0.001	66.78	87.98	62.99	78.64	95.19	75.83	47.85	60.71	46.76

As it is expected, with higher values of m and ϵ, the decrease in the performance of the model is more severe. The F1-score for CIC-IDS2017 drops from 68.45% to 16.57%, for CIC-IDS2018, it drops from 81.23% to 28.14%, and for CIC-UNSW drops from 48.22% to 18.39% when the attack was performed with three features and $\epsilon = 0.1$. Also, the impact of ϵ is more than the number of features on the model's performance. With $\epsilon = 0.1$ and only one feature, the drop in F1-score, PC, and RC is more severe than the attack with three features and $\epsilon = 0.001$.

In general, the proposed saliency map-based attack successfully fools the DNN model trained on three datasets and diminishes the performance of these models. Additionally, compared to [5], we have better performance. In their experiments, after performing the adversarial attack, the Recall for CIC-IDS2017 is 65.6% and for CIC-IDS2018 is 56.4%, while in Table 5.22, the Recall for the attack with three features and ϵ value of 0.1 is 14.78% and 27.84% for these two datasets.

5.3.4.4 Success Rates of the Best Feature Sets

In the experiments, we used three values for ϵ and different features for the attacks. Then after presenting the results for each of these values, we selected the more successful features in generating adversarial samples.

Table 5.23 contains the number of generated samples for each dataset with three

Table 5.23: Generated adversarial samples for the whole datasets.

ϵ	**0.1**	**0.01**	**0.001**
CIC-IDS2017	355644	127414	82693
CIC-IDS2018	1764773	427063	283186
CIC-UNSW	56837	22964	1046

different values of ϵ. As we showed before in Table 5.13, the most successful group of features for the CIC-IDS2017 for all three ϵ values is the same, and it is *Flow IAT Min* and *Fwd IAT Total* together. When these features were used to generate samples for all the records in the dataset, they were almost twice as successful as before.

In the case of the CIC-IDS2018, the most successful group of features contains *Fwd Header Length*, *Total Fwd Packets*, and *Subflow Fwd Packets* for all three ϵs. Using these features to generate adversarial samples shows great performance in the CIC-IDS2018 dataset. For example, $\epsilon = 0.1$ yields three times more successful adversarial samples.

For CIC-UNSW, we used *Bwd Packets/s* and *PSH Flag Count* which are the best features combination for all three ϵ values to generate adversarial samples for the whole datasets. For all three ϵ values the number of successful adversarial samples is almost twice the number presented in Table 5.21.

5.3.4.5 Average Confidence of Adversarial Class (ACAC)

A deep learning model generates a probability for each output class and selects the class with the highest probability as its prediction. If the prediction probability for adversarial samples is high, this means the model is classifying them with high confidence, and the attack is more powerful. We calculate the average confidence score for generated adversarial samples and use it as a criterion to compare our different experiments.

Table 5.24 contains the average confidence of the adversarial class for each dataset.

Table 5.24: The average confidence of adversarial classes.

ϵ	**0.1**	**0.01**	**0.001**
CIC-IDS2017	97.28	85.23	87.19
CIC-IDS2018	98.64	95.01	97.71
CIC-UNSW	91.8	54.19	32.36

We calculate the average for the samples generated for the whole datasets in the previous section. For CIC-IDS2017, CIC-IDS2018, and CIC-UNSW datasets with $\epsilon = 0.1$ the average confidence is more than 91%, which is expected, because with the higher ϵ the changes in the original samples are more severe, and the model decision is more confident. For the CIC-IDS2017, when we decrease the ϵ, the average confidence drops more than 10%, but for the CIC-IDS2018, the change is less. This indicates that the model trained on CIC-IDS2017 is more robust than the model trained on CIC-IDS2018 against the proposed adversarial attack. For CIC-UNSW decreasing the magnitude of ϵ, decreases the average confidence score significantly. This significant drop means that although the attack is still able to generate some adversarial sample, fooling the model is getting harder and it is making the mistake of classification with less confidence.

5.3.4.6 Transferability

Table 5.25 presents the results of the cross-technique transferability. The adversarial samples generated in Section 5.3.4.4 were used in the transferability experiment. Other than the Deep learning model, we trained four other classifiers, including *Decision Tree Random Forest Naive Bayes*, and *Logistic Regression* on the three datasets. Each value in Table 5.25 is the number of adversarial samples that were not detected correctly by the target machine learning model.

For the CIC-IDS2017, since the Naive Bayes and Logistic Regression do not have excellent performance, the number of transferred samples is higher than for Decision Tree and Random Forest for almost all of the ϵ values. Between Decision Tree

Table 5.25: Cross-technique transferability.

Dataset	**Model**	**0.1**	**0.01**	**0.001**
CIC-IDS2017	DT	75643	62053	22735
	RF	83854	55469	22435
	NB	351786	48464	38658
	LR	126139	85095	61104
CIC-IDS2018	DT	599720	314292	263353
	RF	758238	334633	270823
	NB	1090903	240907	152384
	LR	662927	255927	221321
CIC-UNSW	DT	22479	7760	897
	RF	48693	17979	955
	NB	18906	21420	1032
	LR	18603	14457	942

and Random Forest, the one with the better performance has the lower number of transferred samples.

The number of transferred samples of the Decision Tree and Random Forest for the CIC-IDS2018 follows the same trend as the CIC-IDS2017: the one with the better performance has fewer transferred samples. But for Naive Bayes and Logistic Regression, while the performance is still worse than that of the two other models, the number of transferred samples is lower for ϵ values of 0.01 and 0.001.

For CIC-UNSW and $\epsilon = 0.1$ the highest number of transferred samples are for Random Forest, but for the two other ϵ values Naive Bayes has more transfer samples. For ϵ values of 0.01 and 0.001, the Decision Tree has the lowest number of transfer samples, and for $\epsilon = 0.1$ the worst transferability is for Logistic Regression.

Comparing the transferability results of the three datasets shows that there is not any clear relation between the value of the ϵ, target model performance, and transferability. Even CIC-IDS2017 and CIC-IDS2018 which have similar types of network attacks do not show any apparent similarity. But for CIC-UNSW with smaller values of ϵ, the target model with lower performance has higher transferability.

5.4 Combined Attack

As explained before we are going to perform the saliency map-based attack against a target model which has been under label flipping attack during its training phase. For the label-flipping attack, we did experiments on datasets with benign to malicious ratios and various percentages of flipped labels from 10 to 70 percent. For the combined attack we select the target models that trained on the original datasets with 10, 20, and 30 percent label-flipping attacks and perform the saliency map-based attack with the same number of features and ϵ values in the previous section. In the following subsections, we will present the combined attack results for each dataset separately.

5.4.1 CIC-IDS2017

In Table 5.26, the results of the combined attack for CIC-IDS2017 are presented. In general, when the target model is under label flipping attack the success rate of the adversarial attack clearly drops even with 10 percent flipped labels. Also, in almost all cases increasing the number of features while keeping the ϵ value unchanged or increasing the ϵ with the same number of features, increases the success rate.

With the same value of ϵ and the number of features, increasing the flipped labels percentage does not always have the same effect. With $\epsilon = 0.01$ and three features increasing the flip percentage increases the attack's success but with $\epsilon = 0.1$ and two features the success rate first increased and then decreased.

The top feature groups with different ϵ values and the amount of flipped labels are shown in Table 5.27. In the presence of a label-flipping attack, it is not possible to make a general assumption about the most successful feature groups even in the same label-flipping attack. Introducing more flipped labels during the training phase improves the effect of some features on adversarial sample generation. With 10%

Table 5.26: Combined attack results for CIC-IDS2017.

Flip %	ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
10	**0.1**	629871	308274	385399	370666
		22.69	**11.1**	**13.88**	**13.35**
	0.01	474727	114089	176824	181575
		17.1	**4.11**	**6.37**	**6.54**
	0.001	211920	114089	43789	53935
		7.63	**1.29**	**1.57**	**1.94**
20	**0.1**	504644	260303	398862	365101
		18.28	**9.43**	**14.44**	**13.22**
	0.01	479248	147531	184202	184580
		17.36	**5.34**	**6.67**	**6.68**
	0.001	219441	17872	37490	70504
		7.94	**0.64**	**1.35**	**2.55**
30	**0.1**	648208	275192	334839	394709
		23.51	**9.98**	**12.14**	**14.31**
	0.01	544936	174196	232044	224843
		19.76	**6.31**	**8.41**	**8.15**
	0.001	188737	7311	16772	55185
		7.21	**0.26**	**0.6**	**2**

flipped, *Fwd Act Data Pkts* is present in two top feature groups, with 20% flipped in five top feature groups, and finally in all the cases with 30% flip. The same thing happens with the combination of *Total Fwd Packet, Subflow Fwd Packets*, and *Fwd Act Data Pkts* where five of the top feature groups with 30% flip are this combination.

Table 5.27: CIC-IDS2017 top 3 best feature sets for the combined attack.

10					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
77828	Fwd Act Data Pkts	28831	Flow IAT Min	22106	Flow IAT Min
64282	Subflow Fwd Packets	27374	Total Bwd packets Flow IAT Min Subflow Bwd Packets	20595	Flow IAT Mean Flow IAT Min
62482	Total Fwd Packets Subflow Fwd Packets	27342	Fwd Act Data Pkts	8880	Flow IAT Mean Flow IAT Min Fwd IAT Min
20					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
85300	Fwd Act Data Pkts	79286	Fwd Act Data Pkts	20683	Flow Duration Flow IAT Max Fwd Act Data Pkts
61391	Total Bwd packets Subflow Bwd Packets Fwd Act Data Pkts	62173	Total Bwd packets Subflow Bwd Packets Fwd Act Data Pkts	15033	Packet Length Min Down/Up Ratio
45371	Total Bwd packets	42863	Total Bwd packets Subflow Bwd Packets	14925	Packet Length Min Down/Up Ratio FWD Init Win Bytes
30					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
148382	Fwd Act Data Pkts	85271	Fwd Act Data Pkts	22167	Total Bwd packets Subflow Bwd Packets Fwd Act Data Pkts
101123	Total Bwd packets Subflow Bwd Packets Fwd Act Data Pkts	78135	Total Fwd Packet Subflow Fwd Packets Fwd Act Data Pkts	12964	Total Bwd packets Bwd Packet Length Max Fwd Act Data Pkts
86026	Total Bwd packets Fwd Act Data Pkts	77797	Total Bwd packets Subflow Bwd Packets Fwd Act Data Pkts	8378	Total Bwd packets Fwd Act Data Pkts

5.4.2 CIC-IDS2018

The result of the combined attack for CIC-IDS2018 is completely different from CIC-IDS2017. Here, the adversarial attack performance is always better than when there is no label-flipping attack. As expected increasing the number of features or the ϵ value, increases the saliency map-based attack success rate. Increasing the number of flipped labels does not have an understandable relation with the number of generated adversarial samples. In some cases, the success rate keeps increasing and in some other cases, it decreases and then increases. Table 5.28 shows the detailed results of CIC-IDS2018.

The top three best feature groups of the combined attack for CIC-IDS2018 are shown in Table 5.29. Unlike CIC-IDS2017, there are many similarities between the top feature groups for CIC-IDS2018. For the attack on the target model with 10% flipped labels, the top three feature groups for different ϵ values are the same but

Table 5.28: Combined attack results for CIC-IDS2018.

Flip %	ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
10	**0.1**	2451237	1687428	2201309	2164226
		15.49	**10.66**	**13.91**	**13.68**
	0.01	2842554	998248	1855722	1794083
		17.97	**6.31**	**11.73**	**11.34**
	0.001	2153134	577312	813897	856811
		13.61	**3.65**	**5.14**	**5.41**
20	**0.1**	2941030	2219836	2510622	2985471
		18.64	**14.06**	**15.91**	**18.92**
	0.01	2974524	981066	1484063	1794128
		18.85	**6.21**	**9.4**	**11.37**
	0.001	1611037	282524	324749	272954
		10.21	**1.79**	**2.05**	**1.73**
30	**0.1**	3081374	1337511	2367677	2483340
		19.53	**8.47**	**15**	**15.74**
	0.01	3007534	889846	1327110	1931678
		19.06	**5.64**	**8.41**	**12.24**
	0.001	1633541	266534	177377	304872
		10.35	**1.68**	**1.12**	**1.93**

with different rankings. For the 20% flip, the feature groups are also the same except for $\epsilon = 0.001$ where one of the features of the third best group is different. With ϵ values of 0.1 and 0.01 for the 30% flipped attack the top features are exactly the same. But with $\epsilon = 0.001$ the feature groups are the same as the 10% flipped attack.

Table 5.29: CIC-IDS2018 top 3 best feature sets for the combined attack.

10					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
552409	Idle Min	537715	Idle Mean Idle Max Idle Min	536706	Idle Mean Idle Max Idle Min
527001	Idle Mean Idle Max Idle Min	535132	Idle Min	530229	Idle Mean Idle Min
523484	Idle Mean Idle Min	530820	Idle Mean Idle Min	523378	Idle Min
20					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
1215432	Fwd Packets/s	1053043	Flow Packets/s Fwd Packets/s	236450	Flow Packets/s Fwd Packets/s
1214704	Flow Packets/s Fwd Packets/s	771014	Fwd Packets/s	233273	Fwd Packets/s
685659	Flow Packets/s Fwd Packets/s Bwd Packets/s	521909	Flow Packets/s Fwd Packets/s Bwd Packets/s	80758	Flow Packets/s Fwd Packets/s Bwd Init Win Bytes
30					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
721033	Fwd Packets/s	699030	Fwd Packets/s	126410	Idle Mean Idle Max Idle Min
685774	Fwd Packet Length Std Flow Packets/s Fwd Packets/s	685528	Fwd Packet Length Std Flow Packets/s Fwd Packets/s	123397	Idle Min
685774	Fwd Packets/s Fwd Segment Size Avg	685524	Fwd Packets/s Fwd Segment Size Avg	122805	Idle Mean Idle Min

5.4.3 CIC-UNSW

We presented the results for CIC-UNSW in Table 5.30. Other than the results with the 10% flipped attack and $\epsilon = 0.1$, all the success rates are worse than the original saliency map-based attack. It seems in general the attack on the target model with more flipped labels, has less success and the number of generated samples is fewer. Also, the same as the two other datasets, increasing the number of features or the ϵ value, improves the attack's performance.

The top feature groups for the combined attack, are different than the original saliency map-based attack. The list of best features is presented in Table 5.31. Same as the CIC-IDS2017, there is not much similarity between the top feature groups for the CIC-UNSW with the different percentages of the flipped labels. Some features like *PSH Flag Count*, *Down/Up Ratio*, and *Bwd IAT Min* look more effective than others. But, a general decision can not be made based on the presented findings.

Table 5.30: Combined attack results for CIC-UNSW.

Flip %	ϵ	**All**	**1 Feature**	**2 Features**	**3 Features**
10	**0.1**	133776	34099	73896	99216
		32.53	**8.29**	**17.97**	**24.13**
	0.01	43585	5965	14502	17759
		10.6	**1.45**	**3.52**	**4.31**
	0.001	12271	192	355	590
		2.98	**0.04**	**0.08**	**0.14**
20	**0.1**	163824	37850	46250	51945
		39.88	**9.21**	**11.25**	**12.64**
	0.01	40757	3270	8974	17748
		9.92	**0.79**	**2.18**	**4.32**
	0.001	8648	294	551	779
		2.1	**0.07**	**0.13**	**0.18**
30	**0.1**	123383	35859	45725	53287
		30.06	**8.73**	**11.14**	**12.98**
	0.01	43435	5166	9888	11896
		10.58	**1.25**	**2.4**	**2.89**
	0.001	7984	202	348	488
		1.94	**0.04**	**0.08**	**0.11**

Table 5.31: CIC-UNSW top 3 best feature sets for the combined attack.

10					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
37248	Total Bwd packets Fwd IAT Min Bwd IAT Total	3395	PSH Flag Count Down/Up Ratio	119	Total Bwd packets Bwd IAT Total Bwd Header Length
29595	Total Bwd packets Bwd IAT Total	3270	Packet Length Variance Bwd Packet/Bulk Avg	57	Flow IAT Std Fwd IAT Mean PSH Flag Count
9948	Flow IAT Min Fwd IAT Min Bwd IAT Min	2627	Fwd IAT Total PSH Flag Count Bwd Packet/Bulk Avg	49	Total Bwd packets Bwd Header Length
20					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
11103	Bwd IAT Min	3566	PSH Flag Count Down/Up Ratio	181	Total Bwd packets Bwd IAT Total Bwd Header Length
10191	PSH Flag Count Down/Up Ratio	3259	Packet Length Variance Bwd Segment Size Avg Bwd Packet/Bulk Avg	92	Total Bwd packets Bwd IAT Total
8026	Bwd IAT Min SYN Flag Count	2987	Fwd IAT Total Fwd IAT Mean Fwd IAT Std	91	PSH Flag Count Down/Up Ratio Subflow Fwd Packets
30					
0.1		**0.01**		**0.001**	
Samples	**Selected Features**	**Samples**	**Selected Features**	**Samples**	**Selected Features**
11809	PSH Flag Count Down/Up Ratio	3257	Total Length of Bwd Packet PSH Flag Count Down/Up Ratio	85	Flow IAT Std Flow IAT Max
11057	Down/Up Ratio	3097	PSH Flag Count Down/Up Ratio	66	Flow IAT Mean Flow IAT Std Flow IAT Max
9950	Bwd IAT Min	2971	Packet Length Variance Bwd Packet/Bulk Avg	60	Flow IAT Std

5.5 Concluding Remarks

In this chapter, we presented the experimental results of the proposed framework. We provided the results of each attack of the framework one by one for each dataset. We also included the results of the saliency map-based attack against a target model that has undergone the label-flipping attack. The proposed evaluation model has been used to perform a deeper analysis of the saliency map-based attack results. This chapter covered contributions five and six.

In the next chapter, we are going to conclude this thesis by presenting the findings of this research and providing future research directions.

Chapter 6

Conclusion and Future Work

Intrusion detection systems play a crucial role in any cybersecurity infrastructure. However, with the rapid increase in network traffic speed and density, traditional intrusion detection systems need help efficiently detect sophisticated attacks. In recent years, deep neural networks have shown remarkable performance and efficiency in various machine learning tasks, including intrusion detection.

Nevertheless, researchers have discovered that machine learning and deep learning models are susceptible to attacks compromising security and privacy. Some of these attacks occur during the training phase, such as poisoning attacks, while others exploit vulnerabilities in trained models, including membership inference attacks, model inversion attacks, and evasion attacks.

Despite these attacks' significance, most existing research has focused on domains other than network security, leaving a gap in understanding their impact on network-related tasks. Furthermore, studies in the network domain have often borrowed attack techniques from unrelated disciplines without customizing them to suit the unique characteristics of network data. Moreover, evaluations have been conducted using more complex datasets.

Therefore, there is a pressing need for research specifically tailored to the network

domain, considering the peculiarities of network data and employing state-of-the-art attack techniques. Such efforts will help bolster the security of intrusion detection systems and fortify overall cybersecurity measures.

This thesis presents a comprehensive framework comprising two primary components: poisoning and evasion attacks targeting deep learning-based NIDS. For the poisoning attack, we introduce an approach based on label-flipping and we propose two techniques for the evasion attack, using FGSM and saliency maps.

Also, we propose an evaluation model designed explicitly for the saliency map-based evasion attack. To assess the effectiveness of our proposed attacks, we employ three well-known network traffic datasets: CIC-IDS2017, CIC-IDS2018, and CIC-UNSW. The latter dataset is generated by processing UNSW-NB15 raw pcap files and extracting network flows using CICFlowMeter.

In our experiments, we execute the three proposed attacks on each dataset and provide results for the combined label flipping and saliency map-based attack. This thorough evaluation allows us to analyze the impact and effectiveness of our framework on different datasets, showcasing its potential applicability and significance in the context of NIDS.

The results obtained from the label-flipping attack exhibit a significant impact on the performance of the target model across all three datasets. As the number of flipped labels increases, the model's accuracy notably reduces. For instance, in the case of CIC-IDS2017, the average accuracy drops from 98.06% to 29.25% when the percentage of flipped labels reaches 70%. Similarly, for CIC-IDS2018, the average accuracy decreases from 97.36% to 29.2%, and for CIC-UNSW, it falls from 83.8% to 25.97% with the same 70% of flipped labels.

Furthermore, exciting findings are observed for CIC-IDS2017 and CIC-IDS2018 regarding the effect of changing the malicious to the benign ratio of the dataset while keeping the same percentage of flipped labels. It is observed that altering the ratio

does not impact the model's accuracy; however, it increases the F1-score and recall. These results indicate that the label-flipping attack is effective in causing misclassifications and lowering the accuracy of the NIDS model. Moreover, manipulating the malicious-to-benign ratio enhances the model's ability to detect malicious instances, as evidenced by improved F1-score and recall. These findings emphasize the importance of robustly defending against poisoning attacks and considering the dataset's composition to optimize NIDS performance.

The analysis of the FGSM-based attack results reveals that the attack's success rate varies depending on the group of features used against different network attacks. Interestingly, generating many adversarial examples without altering all the features is possible. Across all three datasets, similar patterns are observed, with the most successful feature groups showing consistency.

In most cases, the combination of Forward, Backward, and Flow-based features ranks as the second-best performing group after the group that includes all features, consistently remaining the most effective. With ϵ set to 0.01, the success rate of the mentioned combination is 23.28% for CIC-IDS2017, 8.92% for CIC-IDS2018, and 10.65% for CIC-UNSW.

The evaluation of the saliency map-based attacks demonstrates that the proposed technique is successful and achieves a comparable success rate while using only a few features to generate adversarial examples. When ϵ is set to 0.1, and only one feature is used, the success rates for the three datasets are 13.53%, 8.89%, and 10.07%, considered acceptable. Notably, the top selected features for each dataset remain consistent across all ϵ values, indicating that certain features consistently perform better for generating adversarial samples, regardless of the chosen ϵ value.

Moreover, using the best feature sets makes it possible to generate adversarial samples for the entire dataset, resulting in a success rate that is more than twice as high as the previous results for all three datasets. As the ϵ value decreases, the average

confidence score of the adversarial classes decreases for all three datasets. However, this decrease is more significant for CIC-UNSW, dropping from 91.8% to 32.36%. This indicates that with smaller ϵ values, the generated adversarial examples are less confidently misclassified, but they still maintain a considerable success rate.

Overall, these findings underscore the effectiveness and potential of the saliency map-based attacks in creating adversarial examples with minimal feature manipulation. Additionally, understanding the impact of ϵ values on confidence scores is crucial for evaluating the robustness of the NIDS model against such attacks, especially when dealing with different datasets.

The analysis of the combined attack reveals some similarities among the three datasets, but drawing a general conclusion is challenging. When an evasion attack is executed on a model trained on a poisoned dataset, several factors affect the attack's success rate.

One crucial factor is the reduced accuracy of the target model due to the poisoning attack. As the target model's accuracy decreases, the number of detected samples available for generating adversarial examples also reduces, potentially impacting the success rate of the evasion attack. On the other hand, the decreased accuracy of the target model might make it more susceptible to being fooled by adversarial samples. This could favor the evasion attack, increasing its success rate.

With these two opposing effects at play, the results of the combined attack become inconclusive. In some instances, increasing the percentage of flipped labels in the poisoning attack might lead to a higher success rate for the evasion attack. However, in other cases, it may not significantly impact the success rate.

In conclusion, our comprehensive experimental results and evaluations demonstrate the effectiveness of the proposed poisoning and evasion attacks against deep learning-based NIDS models, which were trained on diverse datasets encompassing different characteristics and attack categories. The success of these attacks highlights the

vulnerability of NIDS models to adversarial manipulation.

Our research sheds light on crucial aspects of adversarial attacks in the context of NIDS, particularly focusing on selecting and utilizing features in generating adversarial samples. By exploring different feature combinations and attack techniques, we provide valuable insights into the intricacies of these attacks.

In summary, our study adds significant value to the field of cybersecurity by exposing the vulnerabilities of deep learning-based NIDS to adversarial attacks and offering crucial insights into effective defense strategies. Our hope is that this research will inspire further exploration and foster the development of advanced techniques to fortify NIDS against evolving threats in the ever-changing landscape of network security.

6.1 Future Work

In this thesis, we focused on investigating two prominent attack categories: poisoning and evasion attacks, both of which pose significant threats to deep learning models in the network domain. However, it is important to acknowledge that the landscape of adversarial attacks is vast and continuously evolving. Numerous other attack techniques against deep learning models exist, which may also impact the security and performance of NIDS.

Instead of using label flipping, poisoning attacks offer alternative techniques such as data injection and poisoning that researchers should consider for future investigations. These attacks aim to generate malicious synthetic samples and inject them into the training set of the target model.

The topic of adversarial attacks on deep learning models in the network domain has received significant attention. However, most existing works, including ours, are confined to the feature space, which is susceptible to changes to the nature of

the flow. In the future, our focus in the feature space should be on ensuring that attackers' modifications to the flow's features do not alter the nature of that flow. Moreover, an alternative direction worth exploring is conducting deep packet adversarial attacks instead of restricting ourselves to the feature space. This approach could open up new possibilities and avenues for research in the field of adversarial attacks on deep learning models in the network domain.

Furthermore, we have focused solely on attacks against deep learning-based NIDS without providing any defense mechanisms against such attacks. To address this crucial gap, a promising future direction is to delve into existing defense methods designed to counter these attacks. We can gain insights into their effectiveness and limitations by exploring and studying these defense techniques.

References

[1] *Types of intrusion detection systems (ids)*, `https://www.omnisecu.com/security/infrastructure-and-email-security/types-of-intrusion-detection-systems.php`, December 2022.

[2] Naveed Akhtar and Ajmal Mian, *Threat of adversarial attacks on deep learning in computer vision: A survey*, IEEE Access **6** (2018), 14410–14430.

[3] Elie Alhajjar, Paul Maxwell, and Nathaniel Bastian, *Adversarial machine learning in network intrusion detection systems*, Expert Systems with Applications **186** (2021), 115782.

[4] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, and Mirco Marchetti, *Addressing adversarial attacks against security systems based on machine learning*, 2019 11th international conference on cyber conflict (CyCon), vol. 900, IEEE, 2019, pp. 1–18.

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

[6] Rana Aamir Raza Ashfaq, Xi-Zhao Wang, Joshua Zhexue Huang, Haider Abbas, and Yu-Lin He, *Fuzziness based semi-supervised learning approach for intrusion detection system*, Information Sciences **378** (2017), 484–497.

[7] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici, *Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers*, International Journal of Security and Networks **10** (2015), no. 3, 137–150.

[8] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar, *The security of machine learning*, Machine Learning **81** (2010), 121–148.

[9] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar, *Can machine learning be secure?*, Proceedings of the 2006 ACM Symposium on Information, computer and communications security, 2006, pp. 16–25.

[10] Yoshua Bengio et al., *Learning deep architectures for ai*, Foundations and trends® in Machine Learning **2** (2009), no. 1, 1–127.

[11] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle, *Greedy layer-wise training of deep networks*, Advances in neural information processing systems **19** (2006).

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Learning long-term dependencies with gradient descent is difficult*, IEEE transactions on neural networks **5** (1994), no. 2, 157–166.

[13] Battista Biggio, Blaine Nelson, and Pavel Laskov, *Support vector machines under adversarial label noise*, Asian conference on machine learning, PMLR, 2011, pp. 97–112.

[14] ———, *Poisoning attacks against support vector machines*, arXiv preprint arXiv:1206.6389 (2012).

[15] Battista Biggio and Fabio Roli, *Wild patterns: Ten years after the rise of adversarial machine learning*, Pattern Recognition **84** (2018), 317–331.

[16] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet, *Deep learning techniques for music generation–a survey*, arXiv preprint arXiv:1709.01620 (2017).

[17] Anna L Buczak and Erhan Guven, *A survey of data mining and machine learning methods for cyber security intrusion detection*, IEEE Communications surveys & tutorials **18** (2015), no. 2, 1153–1176.

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

[19] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh, *Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models*, Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 2017, pp. 15–26.

[20] Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot, *Label-only membership inference attacks*, International conference on machine learning, PMLR, 2021, pp. 1964–1974.

[21] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber, *Flexible, high performance convolutional neural networks for image classification*, Twenty-second international joint conference on artificial intelligence, Citeseer, 2011.

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

[23] Ronan Collobert and Jason Weston, *A unified architecture for natural language processing: Deep neural networks with multitask learning*, Proceedings of the 25th international conference on Machine learning, 2008, pp. 160–167.

[24] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma, *Adversarial classification*, Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, 2004, pp. 99–108.

[25] Hervé Debar, Marc Dacier, and Andreas Wespi, *Towards a taxonomy of intrusion-detection systems*, Computer networks **31** (1999), no. 8, 805–822.

[26] Stuart Dreyfus, *The computational solution of optimal control problems with time lag*, IEEE Transactions on Automatic Control **18** (1973), no. 4, 383–385.

[27] Vasisht Duddu, *A survey of adversarial machine learning in cyber warfare.*, Defence Science Journal **68** (2018), no. 4.

[28] Fahimeh Farahnakian and Jukka Heikkonen, *A deep auto-encoder based approach for intrusion detection system*, 2018 20th International Conference on Advanced Communication Technology (ICACT), IEEE, 2018, pp. 178–183.

[29] Fang Feng, Xin Liu, Binbin Yong, Rui Zhou, and Qingguo Zhou, *Anomaly detection in ad-hoc networks based on deep learning model: A plug and play device*, Ad Hoc Networks **84** (2019), 82–89.

[30] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart, *Model inversion attacks that exploit confidence information and basic countermeasures*, Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1322–1333.

[31] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart, *Privacy in pharmacogenetics: An end-to-end case*

study of personalized warfarin dosing, 23rd {USENIX} Security Symposium ({USENIX} Security 14), 2014, pp. 17–32.

[32] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov, *Property inference attacks on fully connected neural networks using permutation invariant representations*, Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, 2018, pp. 619–633.

[33] Ni Gao, Ling Gao, Quanli Gao, and Hai Wang, *An intrusion detection model based on deep belief networks*, 2014 Second International Conference on Advanced Cloud and Big Data, IEEE, 2014, pp. 247–252.

[34] Xianwei Gao, Chun Shan, Changzhen Hu, Zequn Niu, and Zhen Liu, *An adaptive ensemble machine learning model for intrusion detection*, Ieee Access **7** (2019), 82512–82521.

[35] Kinan Ghanem, Francisco J Aparicio-Navarro, Konstantinos G Kyriakopoulos, Sangarapillai Lambotharan, and Jonathon A Chambers, *Support vector machine for network intrusion and cyber-attack detection*, 2017 sensor signal processing for defence conference (SSPD), IEEE, 2017, pp. 1–5.

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

[37] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, *Generative adversarial nets*, Advances in neural information processing systems, 2014, pp. 2672–2680.

[38] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, *Explaining and harnessing adversarial examples*, arXiv preprint arXiv:1412.6572 (2014).

[39] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel, *Adversarial examples for malware detection*, European Symposium on Research in Computer Security, Springer, 2017, pp. 62–79.

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

[41] Mohammad J Hashemi and Eric Keller, *Enhancing robustness against adversarial examples in network intrusion detection systems*, 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2020, pp. 37–43.

[42] Samer Hijazi, Rishi Kumar, Chris Rowen, et al., *Using convolutional neural networks for image recognition*, Cadence Design Systems Inc.: San Jose, CA, USA **9** (2015), 1.

[43] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh, *A fast learning algorithm for deep belief nets*, Neural computation **18** (2006), no. 7, 1527–1554.

[44] Sorami Hisamoto, Matt Post, and Kevin Duh, *Membership inference attacks on sequence-to-sequence models: Is my data in your machine translation system?*, Transactions of the Association for Computational Linguistics **8** (2020), 49–63.

[45] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al., *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, 2001.

[46] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[47] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Christos Tachtatzis, and Robert Atkinson, *Shallow and deep networks intrusion detection system: A taxonomy and survey*, arXiv preprint arXiv:1701.02145 (2017).

[48] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, *Multilayer feedforward networks are universal approximators*, Neural networks **2** (1989), no. 5, 359–366.

[49] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang, *Membership inference attacks on machine learning: A survey*, ACM Computing Surveys (CSUR) **54** (2022), no. 11s, 1–37.

[50] Weiqing Huang, Xiao Peng, Zhixin Shi, and Yuru Ma, *Adversarial attack against lstm-based ddos intrusion detection system*, 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2020, pp. 686–693.

[51] Olakunle Ibitoye, Omair Shafiq, and Ashraf Matrawy, *Analyzing adversarial attacks against deep learning for intrusion detection in iot networks*, 2019 IEEE Global Communications Conference (GLOBECOM), IEEE, 2019, pp. 1–6.

[52] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li, *Manipulating machine learning: Poisoning attacks and countermeasures for regression learning*, 2018 IEEE symposium on security and privacy (SP), IEEE, 2018, pp. 19–35.

[53] Sravan Kumar Jonnalagadda and Ravi Prakash Reddy, *A literature survey and comprehensive study of intrusion detection*, International Journal of Computer Applications **81** (2013), no. 16, 40–47.

[54] Gozde Karatas, Onder Demir, and Ozgur Koray Sahingoz, *Increasing the performance of machine learning-based idss on an imbalanced and up-to-date dataset*, IEEE access **8** (2020), 32150–32162.

[55] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim, *Long short term memory recurrent neural network classifier for intrusion detection*, 2016 international conference on platform technology and service (PlatCon), IEEE, 2016, pp. 1–5.

[56] Jin Kim, Nara Shin, Seung Yeon Jo, and Sang Hyun Kim, *Method of intrusion detection using deep neural network*, 2017 IEEE international conference on big data and smart computing (BigComp), IEEE, 2017, pp. 313–316.

[57] Diederik P Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 2012, pp. 1097–1105.

[59] Aditya Kuppa, Slawomir Grzonkowski, Muhammad Rizwan Asghar, and Nhien-An Le-Khac, *Black box attacks on deep anomaly detectors*, Proceedings of the 14th International Conference on Availability, Reliability and Security, 2019, pp. 1–10.

[60] Alexey Kurakin, Ian Goodfellow, and Samy Bengio, *Adversarial examples in the physical world*, arXiv preprint arXiv:1607.02533 (2016).

[61] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani, *Characterization of tor traffic using time based features.*, ICISSp, 2017, pp. 253–262.

[62] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, nature **521** (2015), no. 7553, 436–444.

[63] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation **1** (1989), no. 4, 541–551.

[64] Bo Li and Yevgeniy Vorobeychik, *Scalable optimization of randomized operational decisions in adversarial classification settings*, Artificial Intelligence and Statistics, PMLR, 2015, pp. 599–607.

[65] Shaofeng Li, Hui Liu, Tian Dong, Benjamin Zi Hao Zhao, Minhui Xue, Haojin Zhu, and Jialiang Lu, *Hidden backdoors in human-centric language models*, Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 3123–3140.

[66] Gaoyang Liu, Chen Wang, Kai Peng, Haojun Huang, Yutong Li, and Wenqing Cheng, *Socinf: Membership inference attacks on social media health data with machine learning*, IEEE Transactions on Computational Social Systems **6** (2019), no. 5, 907–921.

[67] Yajun Liu and Xuan Zhang, *Intrusion detection based on idbm*, 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/-DataCom/CyberSciTech), IEEE, 2016, pp. 173–177.

[68] Manuel Lopez-Martin, Antonio Sanchez-Esguevillas, Juan Ignacio Arribas, and Belen Carro, *Network intrusion detection based on extended rbf neural network with offline reinforcement learning*, IEEE Access **9** (2021), 153153–153170.

[69] Teresa F Lunt, *A survey of intrusion detection techniques*, Computers & Security **12** (1993), no. 4, 405–418.

[70] Shona McCombes, *Sampling methods — types, techniques & examples*, `https://www.scribbr.com/methodology/sampling-methods/`, December 2022.

[71] Larry Medsker and Lakhmi C Jain, *Recurrent neural networks: design and applications*, CRC press, 1999.

[72] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov, *Exploiting unintended feature leakage in collaborative learning*, 2019 IEEE symposium on security and privacy (SP), IEEE, 2019, pp. 691–706.

[73] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur, *Recurrent neural network based language model.*, Interspeech, vol. 2, Makuhari, 2010, pp. 1045–1048.

[74] Seonwoo Min, Byunghan Lee, and Sungroh Yoon, *Deep learning in bioinformatics*, Briefings in bioinformatics **18** (2017), no. 5, 851–869.

[75] Marvin Minsky and Seymour A Papert, *Perceptrons: An introduction to computational geometry*, MIT press, 2017.

[76] Nikhil Kumar Mittal, *A survey on wireless sensor network for community intrusion detection systems*, 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), IEEE, 2016, pp. 107–111.

[77] Hesamodin Mohammadian, Ali A Ghorbani, and Arash Habibi Lashkari, *A gradient-based approach for adversarial attack on deep learning-based network intrusion detection systems*, Applied Soft Computing **137** (2023), 110173.

[78] Hesamodin Mohammadian., Arash Lashkari., and Ali A. Ghorbani., *Evaluating label flipping attack in deep learning-based nids*, Proceedings of the 20th Inter-

national Conference on Security and Cryptography - SECRYPT, INSTICC, SciTePress, 2023, pp. 597–603.

[79] Hesamodin Mohammadian., Arash Habibi Lashkari., and Ali A. Ghorbani., *Evaluating deep learning-based nids in adversarial settings*, Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP, INSTICC, SciTePress, 2022, pp. 435–444.

[80] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, *Deepfool: a simple and accurate method to fool deep neural networks*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2574–2582.

[81] Nour Moustafa and Jill Slay, *Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)*, 2015 military communications and information systems conference (MilCIS), IEEE, 2015, pp. 1–6.

[82] ———, *The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set*, Information Security Journal: A Global Perspective **25** (2016), no. 1-3, 18–31.

[83] Biswanath Mukherjee, L Todd Heberlein, and Karl N Levitt, *Network intrusion detection*, IEEE network **8** (1994), no. 3, 26–41.

[84] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli, *Towards poisoning of deep learning algorithms with back-gradient optimization*, Proceedings of the 10th ACM workshop on artificial intelligence and security, 2017, pp. 27–38.

[85] Kuniaki Noda, Yuki Yamaguchi, Kazuhiro Nakadai, Hiroshi G Okuno, and Tetsuya Ogata, *Audio-visual speech recognition using deep learning*, Applied intelligence **42** (2015), 722–737.

[86] Seong Joon Oh, Bernt Schiele, and Mario Fritz, *Towards reverse-engineering black-box neural networks*, Explainable AI: Interpreting, Explaining and Visualizing Deep Learning (2019), 121–144.

[87] Pavlos Papadopoulos, Oliver Thornewill von Essen, Nikolaos Pitropakis, Christos Chrysoulas, Alexios Mylonas, and William J Buchanan, *Launching adversarial attacks against network intrusion detection systems for iot*, Journal of Cybersecurity and Privacy **1** (2021), no. 2, 252–273.

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

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

[90] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami, *The limitations of deep learning in adversarial settings*, 2016 IEEE European symposium on security and privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[91] Ye Peng, Jinshu Su, Xiangquan Shi, and Baokang Zhao, *Evaluating deep learning based network intrusion detection system in adversarial environment*, 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), IEEE, 2019, pp. 61–66.

[92] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas, *A taxonomy and survey of attacks against machine learning*, Computer Science Review **34** (2019), 100199.

[93] Howard E Poston, *A brief taxonomy of intrusion detection strategies*, 2012 IEEE National Aerospace and Electronics Conference (NAECON), IEEE, 2012, pp. 255–263.

[94] Sasanka Potluri and Christian Diedrich, *Accelerated deep neural networks for enhanced intrusion detection system*, 2016 IEEE 21st international conference on emerging technologies and factory automation (ETFA), IEEE, 2016, pp. 1–8.

[95] Han Qiu, Tian Dong, Tianwei Zhang, Jialiang Lu, Gerard Memmi, and Meikang Qiu, *Adversarial attacks against network intrusion detection in iot systems*, IEEE Internet of Things Journal (2020).

[96] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Cun, *Efficient learning of sparse representations with an energy-based model*, Advances in neural information processing systems **19** (2006).

[97] Maria Rigaki, *Adversarial deep learning against intrusion detection classifiers*, 2017.

[98] Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), no. 6, 386.

[99] ———, *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, Tech. report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[100] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes, *Ml-leaks: Model and data independent membership in-*

ference attacks and defenses on machine learning models, arXiv preprint arXiv:1806.01246 (2018).

[101] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani, *Toward generating a new intrusion detection dataset and intrusion traffic characterization.*, ICISSP, 2018, pp. 108–116.

[102] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani, *Developing realistic distributed denial of service (ddos) attack dataset and taxonomy*, 2019 International Carnahan Conference on Security Technology (ICCST), IEEE, 2019, pp. 1–8.

[103] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov, *Membership inference attacks against machine learning models*, 2017 IEEE symposium on security and privacy (SP), IEEE, 2017, pp. 3–18.

[104] Jimmy Shun and Heidar A Malki, *Network intrusion detection system using neural networks*, 2008 fourth international conference on natural computation, vol. 5, IEEE, 2008, pp. 242–246.

[105] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, arXiv preprint arXiv:1312.6034 (2013).

[106] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556 (2014).

[107] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai, *One pixel attack for fooling deep neural networks*, IEEE Transactions on Evolutionary Computation (2019).

[108] Fnu Suya, Saeed Mahloujifar, Anshuman Suri, David Evans, and Yuan Tian, *Model-targeted poisoning attacks with provable convergence*, International Conference on Machine Learning, PMLR, 2021, pp. 10000–10010.

[109] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, *Intriguing properties of neural networks*, arXiv preprint arXiv:1312.6199 (2013).

[110] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho, *Deep learning approach for network intrusion detection in software defined networking*, 2016 international conference on wireless networks and mobile communications (WINCOM), IEEE, 2016, pp. 258–263.

[111] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani, *A detailed analysis of the kdd cup 99 data set*, 2009 IEEE symposium on computational intelligence for security and defense applications, Ieee, 2009, pp. 1–6.

[112] Zhiyi Tian, Lei Cui, Jie Liang, and Shui Yu, *A comprehensive survey on poisoning attacks and countermeasures in machine learning*, ACM Computing Surveys **55** (2022), no. 8, 1–35.

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

[114] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart, *Stealing machine learning models via prediction apis.*, USENIX security symposium, vol. 16, 2016, pp. 601–618.

[115] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei, *Towards demystifying membership inference attacks*, arXiv preprint arXiv:1807.09173 (2018).

[116] ———, *Demystifying membership inference attacks in machine learning as a service*, IEEE Transactions on Services Computing **14** (2019), no. 6, 2073–2089.

[117] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin, *Intrusion detection by machine learning: A review*, expert systems with applications **36** (2009), no. 10, 11994–12000.

[118] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol, *Extracting and composing robust features with denoising autoencoders*, Proceedings of the 25th international conference on Machine learning, 2008, pp. 1096–1103.

[119] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li, *Deep learning for content-based image retrieval: A comprehensive study*, Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 157–166.

[120] Binghui Wang and Neil Zhenqiang Gong, *Stealing hyperparameters in machine learning*, 2018 IEEE symposium on security and privacy (SP), IEEE, 2018, pp. 36–52.

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

[122] Zhibo Wang, Jingjing Ma, Xue Wang, Jiahui Hu, Zhan Qin, and Kui Ren, *Threats to training: A survey of poisoning attacks and defenses on machine learning systems*, ACM Computing Surveys **55** (2022), no. 7, 1–36.

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

[124] Paul J Werbos, *Applications of advances in nonlinear sensitivity analysis*, System modeling and optimization, Springer, 1982, pp. 762–770.

[125] Guoxing Wu, Wenjie Lu, Guangwei Gao, Chunxia Zhao, and Jiayin Liu, *Regional deep learning model for visual tracking*, Neurocomputing **175** (2016), 310–323.

[126] Han Xiao, Huang Xiao, and Claudia Eckert, *Adversarial label flips attack on support vector machines*, ECAI 2012, IOS Press, 2012, pp. 870–875.

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

[128] Binghao Yan and Guodong Han, *Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system*, IEEE Access **6** (2018), 41238–41248.

[129] Kaichen Yang, Jianqing Liu, Chi Zhang, and Yuguang Fang, *Adversarial examples against the deep learning based network intrusion detection systems*, MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM), IEEE, 2018, pp. 559–564.

[130] Haipeng Yao, Danyang Fu, Peiying Zhang, Maozhen Li, and Yunjie Liu, *Msml: A novel multilevel semi-supervised machine learning framework for intrusion detection system*, IEEE Internet of Things Journal **6** (2018), no. 2, 1949–1959.

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

[132] Dong Yu and Li Deng, *Deep learning and its applications to signal and information processing [exploratory dsp]*, IEEE Signal Processing Magazine **28** (2010), no. 1, 145–154.

[133] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals, *Understanding deep learning (still) requires rethinking generalization*, Communications of the ACM **64** (2021), no. 3, 107–115.

[134] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song, *The secret revealer: Generative model-inversion attacks against deep neural networks*, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 253–261.

Appendix A

Attacks source code

```
def LabelFlipping(dataset_labels, num_classes, previous_flipped):

    label_count = dataset_labels.value_counts()
    poisoned_label = dataset_labels.copy()
    labels = list(range(num_classes))

    labels_count = {}
    for i in range(num_classes):
        labels_count[i] = int(label_count[i] / 10)

    dataset_labels = dataset_labels.drop(previous_flipped)

    for l in labels:
        label_flip = dataset_labels[dataset_labels['Label'] == l]
        label_flip = label_flip.sample(n=labels_count[l], axis=0)
        previous_flipped += label_flip.index.to_list()

        for i, r in label_flip.iterrows():
            new_label = choice([la for la in labels if la != r['Label']])
            poisoned_label.iloc[i, 0] = new_label

    return poisoned_label
```

Listing 1: Label flipping attack source code

```
def FGSM(model, loss, epsilon, x, y, clip_min=0, clip_max=1, mask=None):

    adv_x = np.copy(x.detach().numpy())

    if the mask is None:
        mask = np.ones_like(adv_x)

    output = model(x)
    loss = loss(output, y)
    model.zero_grad()
    loss.backward()

    with torch.no_grad():
        grad_sign = x.grad.data.sign()
        adv_x += epsilon * grad_sign.numpy() * mask

    adv_x = torch.FloatTensor(adv_x)
    adv_x = torch.clamp(adv_x, min=clip_min, max=clip_max)
    return adv_x
```

Listing 2: FGSM-based attack source code

```
def SaliencyMap(model, epsilon, x, y, feature_comb,  clip_min=0, clip_max=1):

    x_copy = x.clone().detach().requires_grad_(True)
    output = model(x_copy)

    num_classes = output.size()[1]
    jacobian = torch.zeros(num_classes, *x_copy.size())
    grad_output = torch.zeros(*output.size())

    for i in range(num_classes):
        zero_gradients(x_copy)
        grad_output.zero_()
        grad_output[:, i] = 1
        output.backward(grad_output, retain_graph=True)
        jacobian[i] = x_copy.grad.data

    jacobian = torch.transpose(jacobian, dim0=0, dim1=1).squeeze()

    all_sum = torch.sum(jacobian, 0).squeeze()
    alpha = jacobian[y].squeeze() * feature_comb
    beta = (all_sum - jacobian[y].squeeze()) * feature_comb

    max_idx = torch.max(torch.mul(torch.mul(-torch.sum(alpha, 1), torch.sum(beta, 1)),
    ↪  torch.mul(torch.sum(alpha, 1) < 0, torch.sum(beta, 1) > 0)), 0)[1].item()

    adv_x = torch.FloatTensor(np.copy(x.detach().numpy()))
    adv_x += feature_comb[max_idx] * epsilon
    adv_x = torch.clamp(adv_x, min=clip_min, max=clip_max)
    return adv_x
```

Listing 3: Saliency map-based attack source code

Vita

Candidate's full name:
Hesamodin Mohammadian

University Attended:

- Ph.D. in Computer Science, University of New Brunswick (UNB), Fredericton, New Brunswick, Canada, *Jan 2019 - Present.*
- M.Sc. in Computer Engineering - Artificial Intelligence, Kharazmi University of Tehran (KHU), Tehran, Iran, *Sep 2012 - July 2015.*
- B.Sc. in Computer Engineering - Software Engineering, Iran University of Science and Technology (IUST), Tehran, Iran, *Sep 2007 - Sep 2012.*

Publications:

- **Hesamodin Mohammadian**, Ali A. Ghorbani, and Arash Habibi Lashkari, *A Gradient-based Approach for Adversarial Attack on Deep Learning-based Network Intrusion Detection Systems*, Applied Soft Computing 137 (2023), 110173.
- **Hesamodin Mohammadian**, Arash Habibi Lashkari, and Ali A. Ghorbani, *Evaluating Label Flipping Attack in Deep Learning-based NIDS*, In Proceedings of the 20th International Conference on Security and Cryptography - SECRYPT, 2023, pp. 597-603.
- **Hesamodin Mohammadian**, Arash Habibi Lashkari, and Ali A. Ghorbani, *Evaluating Deep Learning-based NIDS in Adversarial Settings*, In Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP, 2022, pp. 435–444.
- Leila Rashidi, Windhya Hansinie Rankothge, **Hesamodin Mohammadian**, Rashid Hussain Khokhar, Brian Frei, Shawn Ellis, Lago Freitas, and Ali A. Ghorbani, *Securing Supply Chain: A Comprehensive Blockchain-based Framework and Risk Assessment*, In Proceedings of the 20th International Conference on Privacy, Security, and Trust - PST, 2023.
- Leila Rashidi, Windhya Hansinie Rankothge, **Hesamodin Mohammadian**, Rashid Hussain Khokhar, Brian Frei, Shawn Ellis, Lago Freitas, and Ali A. Ghorbani, *A Survey on Supply Chain Management: Exploring Physical and Cyber Security Challenges, Threats, Critical Applications, and Innovative Technologies*, International Journal of Supply and Operations Management (2023).