A Dissertation

entitled

A Study on Behaviors of Machine Learning-Powered Intrusion Detection Systems under Normal and Adversarial Settings

by

Medha Rani Pujari

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Computer Science

Dr. Weiqing Sun, Committee Chair

Dr. Ahmad Y. Javaid, Committee Co-Chair

Dr. Mohammed Niamat, Committee Member

Dr. Devinder Kaur, Committee Member

Dr. Junghwan Kim, Committee Member

Dr. Scott Molitor, Acting Dean
College of Graduate Studies

The University of Toledo
March 2023

An Abstract of

A Study on Behaviors of Machine Learning-Powered
Intrusion Detection Systems under Normal and Adversarial Settings

by

Medha Rani Pujari

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Doctor of Philosophy Degree in Computer Science

The University of Toledo
March 2023

Intrusion detection systems (IDSs) have evolved significantly since the first time they were introduced and have become one of the most essential defenses in a network. With the advent of machine learning (ML), several improvements and enhancements have been made to the capabilities of traditional IDSs. However, every advancement brings with it a range of new challenges and threats. Although ML expanded the abilities of IDSs, there are certain problems that need to be investigated and this research attempts to highlight and address some of the existing problems. One of the problems is that a major portion of the research progress involving IDSs has been achieved using decades-old datasets. This work aims to study recently published research IDS datasets and analyze the performances of ML-based IDS models when trained with such datasets. Another problem focused on in this research is the vulnerabilities of ML models to adversarial environments. The work identifies that a majority of research progress achieved relevant to ML-powered IDSs is toward the direction of improving the performance efficiency of the IDS models under

normal settings, i.e., toward optimizing the detection rates with genuine data. Relatively little progress is made towards making the IDS models robust to adversarial environments and deceptive inputs that target the IDSs rather than the premises (networks or hosts) guarded by them. This is a serious concern in cybersecurity which needs more investigation and problem-solving. In regard to this concern, various types of adversarial attacks are studied, and the behaviors of IDSs in certain white-box adversarial settings are assessed when the models are trained with modern research datasets. The study extends further by developing a defense mechanism against a white-box evasion attack which is considered to be very powerful for image-classification-based models. As the IDS models deployed in real-world environments are more susceptible to black-box attacks, this research pursues to develop a defense for IDSs against adversarial black-box evasion attacks, in particular, the query-based attacks that rely on the decisions made by their target model(s).

Query-based black-box evasion attacks have become more popular lately, targeting MLaaS (ML as a Service) models, and those deployed in commercial/real-time environments. The latest module of work in this research focuses on studying the performance of an IDS model when provided with the data generated by an adversarial black-box attack algorithm called genetic algorithm (GA). It extends further by proposing a defense mechanism to mitigate the impact of such powerful black-box algorithms. Experimental and evaluation results are presented and analyzed.

This work and dissertation are dedicated to my dear husband and the best mentor of my life, Yeshwanth Rangineni. He is the one who trusts me the most and stands by me in all my endeavors.

Acknowledgments

I would like to express my appreciation and gratitude to my advisor Dr. Weiqing Sun, who has been a great mentor for me throughout my Master's and Doctoral studies. I honestly could not have asked for a better advisor in life to succeed. He is not only a great advisor and mentor but is also my inspiration and someone I look up to. I would also like to especially thank my research co-advisor, Dr. Ahmad Javaid, for his immense support and guidance throughout my journey at the University of Toledo (UT). I have been very fortunate to have obtained guidance and support from two wonderful people. Their support in offering me teaching and research assistantships and believing in my capabilities is also greatly appreciated. I am grateful to all my dissertation committee members who have been very supportive and inspiring. Their service has been very valuable to me and I very much appreciate their roles in my learning process. I have deeply enjoyed my journey with UT and would like to express my special thanks to the entire family of UT!

Finally, I would like to thank my husband, Yeshwanth Rangineni, who is my greatest support system in life. He has been the driving force that enabled me to pursue my interests and follow my dreams. I would also like to thank my family for their constant support and encouragement. The confidence they have in me has been a boosting power for me to be persistent in my efforts and keep going forward.

Contents

List of Tables

List of Figures

List of Abbreviations

AML	Adversarial machine learning
AUC	Area under the curve
CIC	Canadian institute of cybersecurity
CSE	Communications security establishment
CW	Carlini Wagner
DL	Deep learning
DT	Decision tree
EA	Evolutionary Algorithm
FGSM	Fast gradient sign method
GA	Genetic algorithm
GAN	Generative adversarial network
GB	Gradient Boosting
IDS	Intrusion detection system
JSMA	Jacobian-based saliency map attack
ML	Machine learning
MLaaS	ML as a service
MLP	Multi layer perceptron
NN	Neural network
RF	Random forest
SVM	Support vector machine
UNSW	University of New South Wales
XGBoost	Xtreme gradient boost
ZOO	Zeroth Order Optimization

Chapter 1

Introduction

Intrusion detection systems (IDSs) were first designed to proactively monitor a network's traffic and raise alerts when suspicious traffic enters the network, or if an unusual event occurs [1]. Since then, the IDS technology underwent several enhancements [2]. However, despite the advancements, there was no significant improvement in the detection rates of IDSs and the number of false alarms. To address such performance issues, research emerged in the late 1990s to use machine learning (ML) techniques in IDS development [3]. The IDS models powered by ML could take decisions on unseen patterns of data, in addition to the ones they are aware of, unlike the traditional IDSs. Furthermore, the traditional detection systems took a long time to analyze and respond to new or complex signatures, whereas the ML-based models could do that in very less time [4]. Thereafter, research progressed toward improving the efficiency of ML-based IDSs while also bringing down false detections. Nevertheless, a system is never totally secure because the concept of security is only relative. There are always ways for adversaries to exploit and get through a secured perimeter. This research emphasizes some of the existing problems in relevance to

the area of IDSs, and analyzes their impacts. It also puts forth approaches that can reduce the impact and make the IDSs more immune to the discussed problems. The remainder of this chapter elaborates on the problems and discusses the work done (and in progress) to address them.

For a long time, the experiments were based on research datasets such as DARPA 98, KDD99, and NSL-KDD, which were created in the late 1990s. Although they had been very useful in research experiments and findings, with rapidly changing network behaviors and attacks growing more sophisticated day by day, these datasets become less usable as they do not reflect modern traffic behaviors. As significant as a research experiment is, its applicability will be in question because the data it has learned and understood does not include contemporary data patterns and scenarios. Therefore, generating more datasets that reflect modern attack scenarios and are realistic enough to make a model fit in a real-time network is necessary. As there are recently published datasets published for the research community, a study on their properties and influence on the performance of IDS models is beneficial to the researchers. This work is discussed in detail in Chapter 2.

In the early 2000s, a study by N. Dalvi et al. [5] revealed a concerning vulnerability of ML algorithms when exposed to adversarial inputs. Later, it was shown that deep learning (DL) and neural networks (NNs) are more vulnerable to adversarial perturbations [6–12]. Various adversarial attack scenarios were developed, and classifiers were evaluated by subjecting them to the adversarial samples generated through attack algorithms. Defense mechanisms had also been proposed to reduce the impact of adversarial perturbations on the models [13]. However, much of this

progress was made in image-based areas, like computer vision, image processing, etcetera. Relatively lesser progress has been made in the IDS domain [14]. One of the major concerns about training IDSs is datasets. The performance of an IDS hugely depends on the quality of the data it learns from.

Three adversarial white-box evasion attack algorithms – Jacobian-based Saliency Map Attack (JSMA), Fast Gradient Sign Method (FGSM), and Carlini Wagner (CW) – were used to study their impact on the performance of the models. The novelty of this part of the work lies in the combination of the components such as the contemporary datasets, attack algorithms, and the domain itself. Unlike image data, network traffic data is more complex to analyze, process, and work with. The experiment is discussed in detail in Chapter 3.

White-box evasion attacks are considered the attacks with the maximum impact on the target models because the attack algorithms have almost complete knowledge of the model and control over its dataset. This research proposes a defense that can reduce the impact of a powerful white-box evasion attack, i.e., the CW. The defense is based on a generative adversarial network (GAN). The CSE-CIC-IDS2018 [15–17] dataset is used for the experiment, which is discussed in Chapter 4.

White-box attacks are mostly implemented in controlled, and in particular, research or experimental environments. They are less common to occur in real-world networks because it is not likely for an adversary to have complete knowledge of the target model. In practice, an adversary has zero to partial knowledge of a target model, which is the case of a black-box or a gray-box attack, respectively. This research moves forward toward developing a defense against adversarial black-box

evasion attacks. The goal is to detect query-based black-box attacks and block such attack inputs to prevent the model from processing the deceptive inputs.

The outline of the remainder of this document is as follows. Chapter 2 covers the evaluation of contemporary datasets using various efficient classification algorithms. Chapter 3 discusses the analysis of the impact of adversarial white-box evasion attack algorithms on the model's performance when it is trained with contemporary datasets. Chapter 4 is on the evaluation of the proposed defense mechanism to make IDS models resistant to the CW attack. Chapter 5 is about the work that is currently in progress to develop a defense technique to tackle adversarial black-box evasion attacks. Chapter 6 concludes the proposal document by discussing the dissertation milestones.

Chapter 2

A Comparative Study on Contemporary Intrusion Detection Datasets

The datasets used for this study are UNSW-NB15, Bot-IoT, and CSE-CIC-IDS2018. Some of the factors behind choosing these datasets are the modern attack scenarios they include, the number of protocols, and the types of attacks. Based on the efficiency demonstrated in the literature, four classification algorithms – random forest (RF), support vector machine (SVM), keras-based deep learning (DL), and xtreme gradient boost (XGBoost) – are used.

2.1 Datasets

2.1.1 UNSW-NB15

This dataset was created in a synthetic environment at the University of New South Wales (UNSW) cybersecurity lab. The dataset has 49 features, with 9 types of attacks, and normal traffic. The data records are pre-split into training and test sets [18]. It has a total of 2,218,761 malicious traffic instances and 321,283 normal traffic instances. Since this dataset has a relatively smaller difference between the number of normal and malicious instances than that of the other two datasets, we consider it to be less imbalanced (or more balanced) than the others. This dataset's balance (normal:malign) ratio is approximately 1:0.14. The categories of attacks it includes are – fuzzers, analysis, backdoor, DoS, exploits, generic, reconnaissance, shellcode, and worms [19].

2.1.2 Bot-IoT

This dataset was generated in 2018. It consists of simulated IoT network traffic, with various types of IoT attacks. It is available in the form of pre-split training and test sets, with a total of about 9,543 normal traffic instances and 73,360,900 malicious traffic instances, resulting in a huge imbalance between the number of normal and malicious traffic instances. The balance ratio in this dataset is approximately 1:7687.

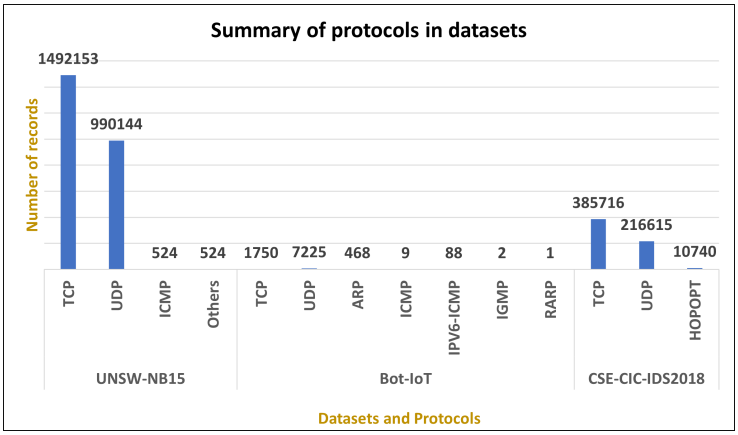

Figure 2-1: Protocols in the datasets.

2.1.3 CSE-CIC-IDS2018

This dataset is a collaborative effort of the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC). Created in the year 2018, this dataset was generated using a vast network of machines to make it close to real-world traffic data. The testbed has a victim network with 420 computers and 30 servers. The dataset includes the network traffic and logs from each of the machines present in the victim network. A fair fraction of this dataset consists of 1,048,010 normal scenario records with only 567 malicious scenario records, resulting in a balance ratio of 1848:1. It does not have available pre-split training and test sets. The summary of protocols in the three datasets is shown in Figure 2-1 [20].

2.2 Classification Algorithms

2.2.1 Random Forest

The RF classifier, as the name suggests, is an ensemble learning technique that consists of a large number of individual decision trees. Each tree in the RF performs the classification of a label and the label with the majority votes becomes the outcome of a prediction. A reason it is a powerful algorithm is that it results in trees that are not only trained on different sets of data (bagging) but also use different sets of features to make decisions. The main idea is to combine hundreds or thousands of decision trees and train each one on a slightly different set of observations while splitting nodes in each tree [21]. The final predictions of the algorithm are made by averaging the predictions of each tree. RF is also used as a feature selection algorithm as each tree predicts random features and gives out high accuracy based on those random selections of features.

2.2.2 Support Vector Machine

An SVM is described as a discriminative classifier that works by generating a hyperplane. In other words, given labeled training data (supervised learning), this algorithm outputs an optimal hyperplane that categorizes new examples. In two-dimensional space, this hyperplane is a line that divides the plane into two parts where each of the classes lay on either side. When perfect separation is not possible, the classifier maximizes the margins and minimizes the misclassifications. It tries to maintain the slack variable to zero while maximizing the margin. A slack variable is

used to penalize the objective function when the SVM misclassifies [22].

2.2.3 Deep Learning

DL is a class of ML algorithms that uses multiple layers to extract progressively higher-level features from the raw input. The deep in "deep learning" is the various layers through which the data is transformed. It allows training to predict outputs based on a set of inputs. It uses a neural network to imitate human intelligence. The neural network consists of three or more layers of neurons [23].

2.2.4 Xtreme Gradient Boost

XGBoost is an ensemble technique that sequentially adds predictors and corrects previous models. Its implementation offers multiple advanced features such as model tuning, computing environments, and algorithm enhancement. In addition to standard gradient boosting (GB), this algorithm is capable of performing two other major forms of GB - stochastic GB and regularized GB. XGBoost supports fine-tuning and the addition of regularization parameters.

2.3 Experimental Setup

Figure 2-2 shows the workflow of the experiment.

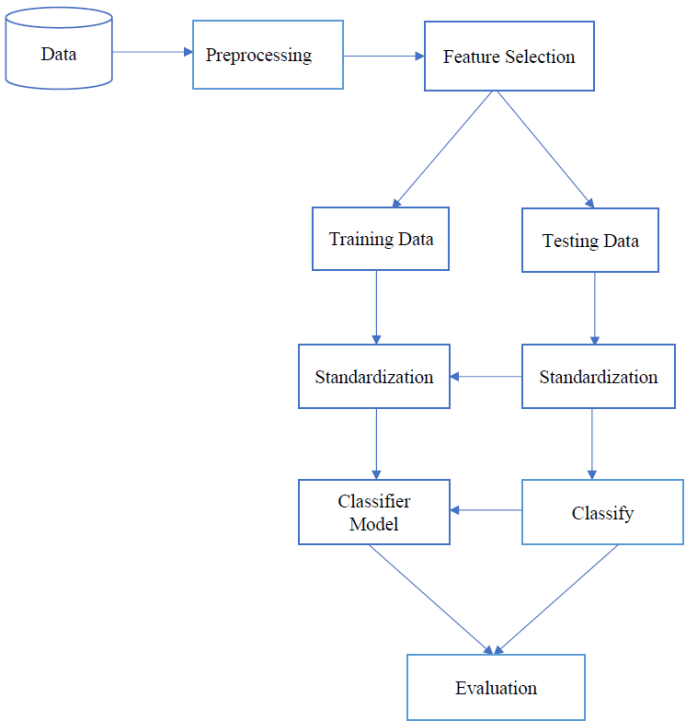

Figure 2-2: workflow of the experiment.

2.3.1 Software Specification

Spyder workspace, which comes in-built with Anaconda software, was used for coding the model. All the necessary packages, such as pandas, numpy, and other classification models were imported from the scikit-learn package.

2.3.2 Preprocessing

The impute module of the scikit-learn package was used to replace the NULL/-NaN values in the datasets with mean or median, based on the nature of values in the columns in which they were found. Feature selection is performed to determine which features contribute the most to the output. For this experiment, the ten best features were selected from each dataset, and then the columns were divided into features and labels. Label encoding was also applied to categorical features and classes to make them more suitable for analysis.

2.3.3 Standardization and classifier model

After feature selection, the dataset was split into train and test sets in a ratio of 75:25. K-fold-cross validation was used to form randomized validation sets. The results from five-, eight-, and ten-fold cross-validations suggested that the five-fold validation was almost as good as eight- and ten-fold, and the latter ones took much longer execution time than five-fold. Additionally, due to resource constraints, five-fold turned out to be a fair choice. The training data was standardized before training the model to make data in all the columns fall in the same range. The model was

trained in an under-sampling manner, instead of over-sampling, to avoid any bias that the model can learn during the training process. The training data used from the CSE-CIC-IDS2018 dataset was only about 10 percent of its total available training-set size, due to limited hardware resources. For the other two datasets, the entire pre-split training sets of data were used. The portion from the CSE-CIC-IDS2018 dataset was carefully chosen to make sure it has all the types of attacks covered, but with a relatively fewer number of records. It was made sure that this portion of the dataset has a similar balance ratio to that of the complete dataset.

2.4 Experimental Setup

There are two scenarios followed to evaluate the datasets. The first scenario is to assess each dataset by training the model with each of the selected algorithms. In the second scenario, as an additional step, a class of attacks is separated, and the model is trained with the remaining classes and is then evaluated to know whether it can identify the untrained attacks. This methodology lets us understand if the trained ML models can identify new and unseen attacks.

2.4.1 Analysis of Performance using RF and SVM

Table 2.1 [20] shows the outcomes of the classification of unseen malicious profiles. The results show that RF performs better overall in classifying the untrained malicious instances for all three datasets. Bot-IoT has all the predictions correct. The reason could be that the majority of profiles in Bot-IoT are about malicious

data and the model might be biased towards malicious instances. In the case of CSE-CIC-IDS2018, the majority of its profiles belong to normal data, and hence the model tends to predict the test inputs as normal, particularly when the SVM algorithm is used. Similarly, the classification seems to be unbiased for UNSWNB-15 in identifying the Worms although the prediction accuracy is not so impressive.

A summary of the number of false predictions made when trained with each of the datasets is presented at the end of this section.

Table 2.1: Predictions on unseen malicious data using RF and SVM .

Datasets	**Malicious Profiles**	**RF**	**SVM**
UNSW-NB15	Worms (44)	27 Malicious	28 Malicious
		17 Normal	16 Normal
Bot-IoT	Theft (65)	65 Malicious	63 Malicious
			2 Normal
CSE-CIC-IDS2018	SQL Injection (53)	22 Malicious	7 Malicious
		31 Normal	46 Normal

Figures 2-3 - 2-8 are the confusion matrices showing the classification by RF and SVM algorithms when trained with the UNSW-NB15 dataset.

2.4.2 Analysis of performance using DL implementation

The prediction results on unseen malicious instances as seen in Table 2.2 show how the performance of the model varies depending on the balance ratio of the profiles in the datasets. The classification of the untrained Worms profile in UNSW-NB15 is fair with 26 identified as malicious and 18 as normal. For the Bot-IoT dataset, which has a dominating number of malicious profiles, the classification of Theft is

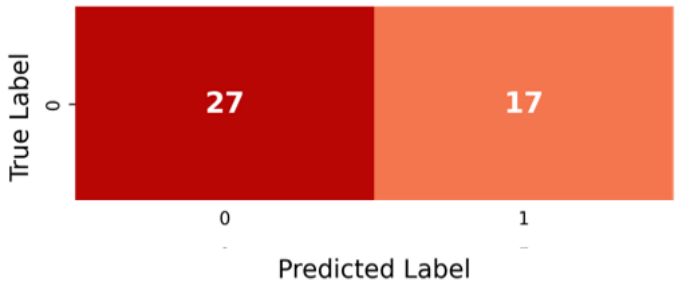

Figure 2-3: Confusion matrix for RF classifier on 44 malicious records of UNSW-NB15.

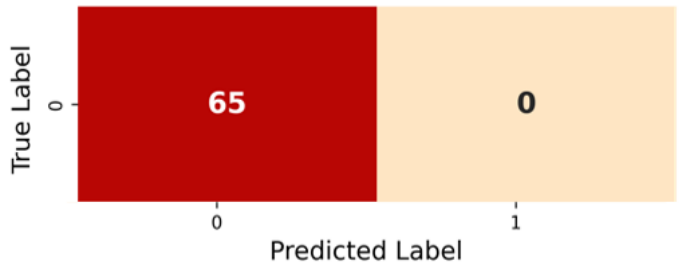

Figure 2-4: Confusion matrix for RF classifier on 65 malicious records of Bot-IoT.

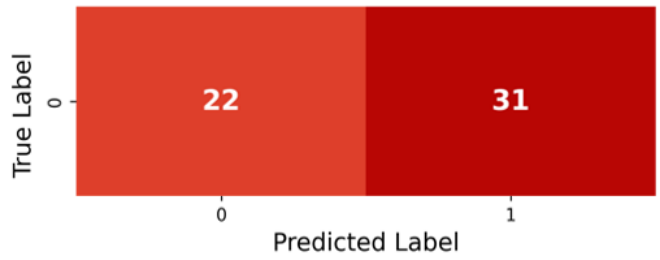

Figure 2-5: Confusion matrix for RF classifier on 53 malicious records of CSE-CIC-IDS2018.

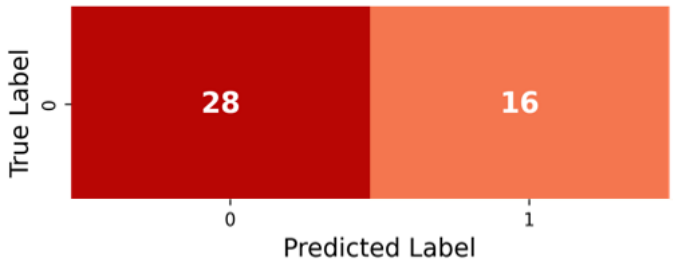

Figure 2-6: Confusion matrix for SVM classifier on 44 malicious records of UNSW-NB15.

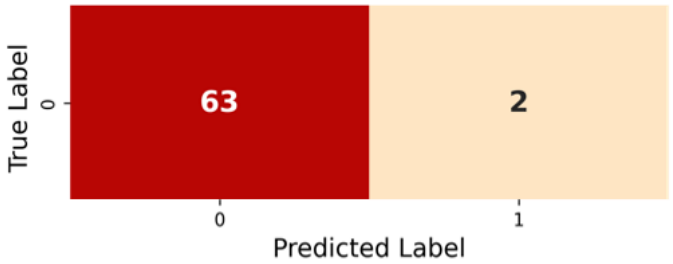

Figure 2-7: Confusion matrix for SVM classifier on 65 malicious records of Bot-IoT.

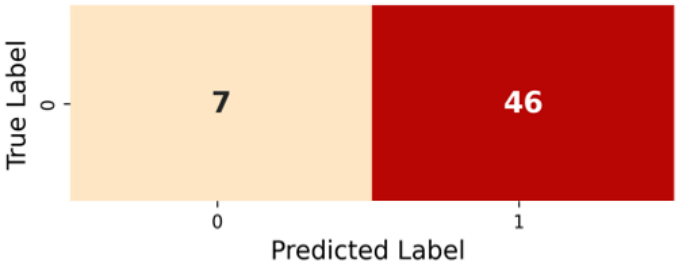

Figure 2-8: Confusion matrix for SVM classifier on 53 malicious records of CSE-CIC-IDS2018.

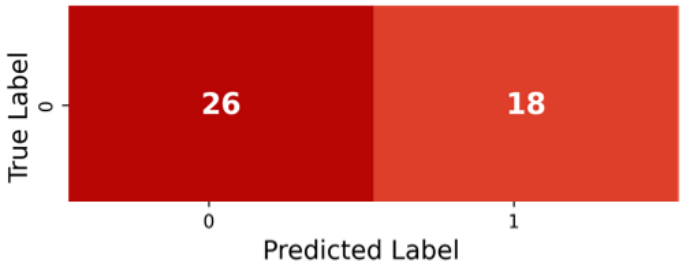

Figure 2-9: Confusion matrix for DL classifier on 44 malicious records of UNSW-NB15.

perfect, with no false prediction. Similarly, for CSE-CIC-IDS2018 data, which has a dominating number of normal profiles, the classification of the SQL injection turns out to be normal with only 5 instances identified as malicious. The deep learning implementation adds more weight to the statement that balanced datasets are better for training the ML-based IDS.

Table 2.2: Predictions on unseen malicious data using DL.

Datasets	**Malicious Profiles**	**DL**
UNSW-NB15	Worms (44)	26 Malicious
		18 Normal
Bot-IoT	Theft (65)	65 Malicious
CSE-CIC-IDS2018	SQL Injection (53)	5 Malicious
		48 Normal

The confusion matrices for the results shown in Table 2.2 are presented in Figures 2-9 - 2-11.

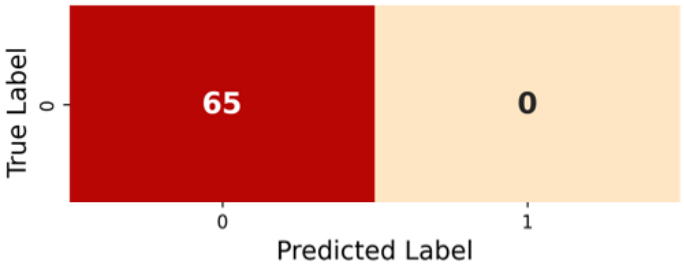

Figure 2-10: Confusion matrix for DL classifier on 65 malicious records of Bot-IoT.

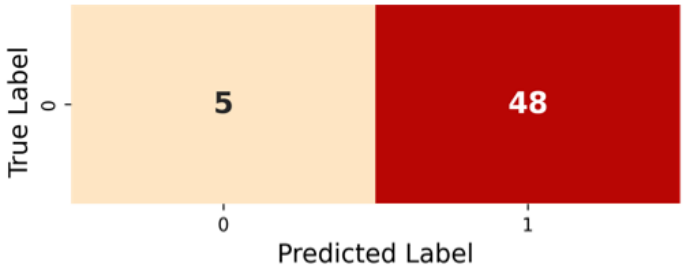

Figure 2-11: Confusion matrix for DL classifier on 53 malicious records of CSE-CIC-IDS2018.

2.4.3 Analysis of performance using XGBoost

It can be seen in Table 2.3 that the classes with a higher number of instances in imbalanced datasets make the model biased and affect the prediction of malicious attacks. XGBoost predicts the untrained malicious profiles from UNSW-NB15 and BotIoT datasets accurately. However, in the case of CSE-CICIDS2018, it has predicted none of the profiles as an attack. Although the training data was under-sampled, the model when trained with the CSE-CIC-IDS2018 dataset is biased towards the normal profile (majority class in the dataset) and hence, has made biased predictions.

Table 2.3: Predictions on unseen malicious data using XGBoost.

Datasets	**Malicious Profiles**	**XGBoost**
UNSW-NB15	Worms (44)	42 Malicious
		2 Normal
Bot-IoT	Theft (65)	65 Malicious
CSE-CIC-IDS2018	SQL Injection (53)	0 Malicious
		53 Normal

Figures 2-12 - 2-14 are the confusion matrices for the results shown in Table 2.3.

2.4.4 Analysis of false predictions

Figure 2-3 shows a summary of the false predictions (false positives and false negatives) made by the model when trained with each of the datasets (The XGBoost classifier is represented as XGB in the figure). Results show that the balance ratio of the datasets can influence the number of false predictions made by a model. The

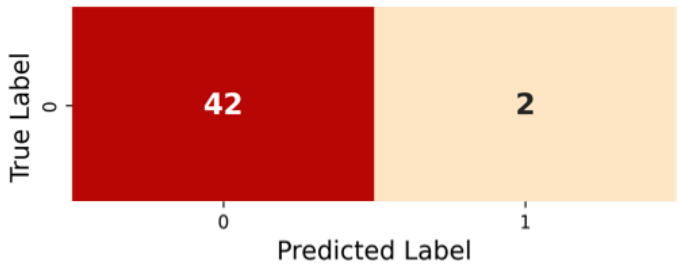

Figure 2-12: Confusion matrix for XGBoost classifier on 44 malicious records of UNSW-NB15.

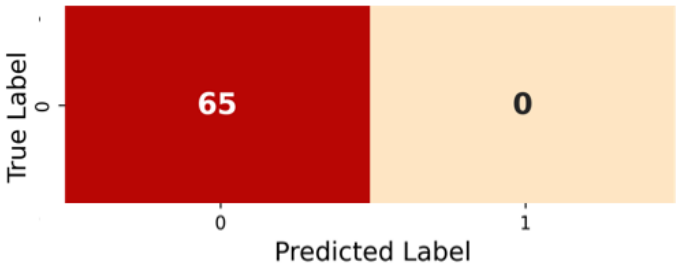

Figure 2-13: Confusion matrix for XGBoost classifier on 65 malicious records of Bot-IoT.

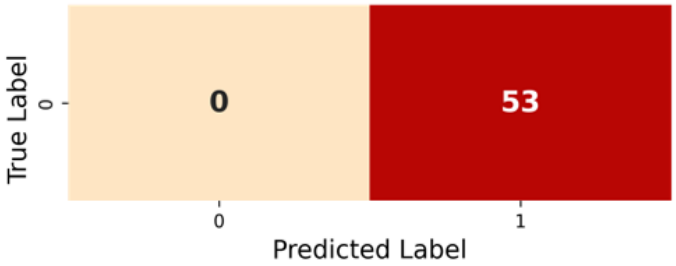

Figure 2-14: Confusion matrix for XGBoost classifier on 53 malicious records of CSE-CIC-IDS2018.

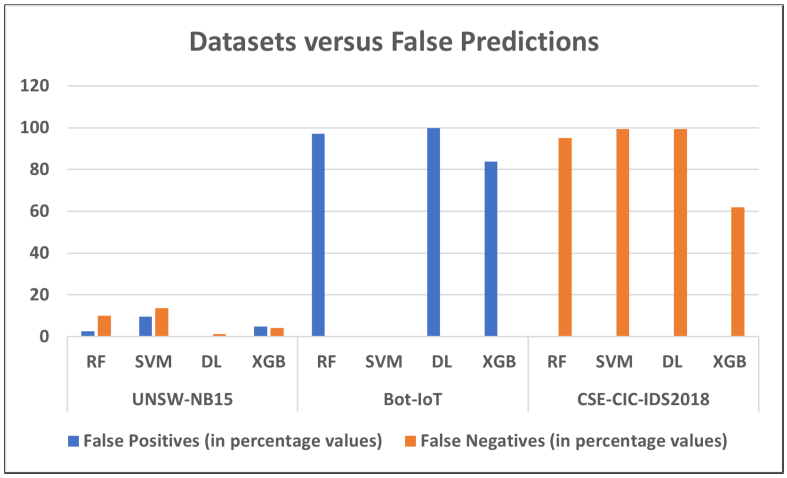

Figure 2-15: Summary of false predictions.

model rendered relatively fewer and more balanced false predictions when trained with a well-balanced (or relatively more balanced) dataset, i.e., UNSW-NB15, unlike when it is trained with more imbalanced datasets.

2.5 Conclusion and Future Work

In this experiment, the performance of three contemporary IDS datasets is compared by training a model with each of them using a set of algorithms that are well-known for their efficiency and robustness. Many existing ML-based IDS models are trained with old benchmark datasets and there is a need to study the recent ones and analyze their characteristics so that the researchers can switch to the latest datasets by choosing one which satisfies their requirements, and this stands as the motive behind this work. The evaluation results suggest that the imbalance in the datasets can influence the number of false predictions made by a model.

A thorough analysis of the research datasets is essential as the performance of research models and their applicability to real-world scenarios hugely depend on the quality of data they learn or work with. Data generated in a typical simulated environment is, in multiple ways, different from the network traffic observed in a real network. However, the gap between the two can be reduced by carefully incorporating the properties that make data more realistic. ML techniques can also be used to understand the detection patterns of the network traffic and bypass the IDS by manipulating the values in the dataset. Hence, as future work, more research needs to be done in two directions - toward improving the datasets in a way that they can not only enhance the quality of research but also be used in real-life networks, and in developing mechanisms to handle the IDS evasion techniques.

Chapter 3

Impact of Adversarial Machine Learning Attacks on Contemporary Intrusion Detection Datasets

Adversarial Machine Learning (AML) is the concept of deceiving an ML model by perturbing an input to make the model render an incorrect prediction. The perturbed input is crafted in a way that it is imperceptible to humans but makes a considerable difference to a neural network. Neural networks are vulnerable to adversarial attacks during training as well as testing/validation phases. Attack techniques can vary based on factors like the time of occurrence (training, testing, etc.), the knowledge of the target model that the attacker has, the target of the attack, the influence of the attacker, etc. The attacks carried out in the training phase are known as

Poisoning attacks and those launched during the testing phase are called Evasion attacks. As highlighted in [24], three major properties of an attack are the influence, the focus of violation (confidentiality, integrity, availability), and the specificity of the target. For example, based on some of the factors stated above, an evasion attack can be classified as either a white-box attack, for which an attacker needs to have complete knowledge of the model (including details like training dataset, parameters, etcetera), or a black-box attack, where the attacker has almost no knowledge of the model, or a gray-box attack, for which the attacker should have partial knowledge of the target. The attacks used for this experiment are all white-box evasion attacks.

3.1 Adversarial Attack Algorithms

3.1.1 Jacobian-based Saliency Map Attack

The JSMA, introduced in [10], is one of the attack techniques evaluated in this study. It is an evasion attack that works by modifying the inputs based on the gradients of the target model with respect to the perturbed features. It iteratively generates a saliency map using which a feature is chosen that will have a maximum prediction error when a perturbation is added [25]. The attack aims to perturb the least possible number of features to cause misclassification by setting a limited perturbation budget. For each feature selected, the perturbation is adjusted and the iterations are continued until misclassification in the target class is achieved or the limit for a maximum number of perturbed features is met [10]. If it fails to achieve this, the algorithm selects the next feature and repeats the process with

it [11]. Although the success rates achieved by JSMA and FGSM are almost similar, the number of features modified is relatively lesser and the computational costs are higher with JSMA, than with FGSM [25]. Algorithm 1 outlines a generic flow of steps involved in JSMA attack.

Algorithm 1: JSMA attack

Require: Target model M, input x, target label t, maximum perturbation budget k, saliency threshold γ, perturbation multiplier θ
Ensure: Adversarial example x_{adv}
Set $x_{adv} \leftarrow x$
Set $i \leftarrow 0$ ▷ i is the counter for iteration
while the predicted label of x_{adv} is not equal to t and $i < k$ **do**
Compute the Jacobian matrix J of M at x_{adv}
Compute the saliency map S of x_{adv} using J and the current predicted label of x_{adv}
Sort the features of x_{adv} in descending order of their saliency values
for each feature f in x_{adv} **do**
if the saliency value of f is less than γ **then**
Skip to the next feature
end if
Compute the gradient of M with respect to f at x_{adv}
if the gradient points in the direction of decreasing f **then**
Perturb f by $-\theta$
else
Perturb f by θ
end if
Clip the perturbed value of f to the range $[0, 1]$
Update x_{adv} with the perturbed value of f
end for
Increment i by 1
end while
return x_{adv}

3.1.2 Fast Gradient Sign Method

The FGSM attack was proposed by [8] for adversarial data generation. The algorithm uses a loss function and aims to minimize it [14]. Unlike the JSMA attack, the FGSM attack does not aim at generating minimal adversarial perturbations. However, it tries to speed up the adversarial data generation process [8], and this is why it takes less time for adversarial data generation than the JSMA. Algorithm 2 provides a generic pseudocode for FGSM attack.

Algorithm 2: FGSM attack

Require: Input x, target label t, step size ϵ, gradient function G, perturbation η
Ensure:
Perturbed input x'
Calculate the gradient $g \leftarrow G(x, t)$
Compute the perturbation $\eta \leftarrow \epsilon \cdot \text{sign}(g)$
Generate the perturbed input $x' \leftarrow \text{clip}(x + \eta, 0, 1)$ ▷ Perturbed input is clipped to the range [0,1]
return x'

3.1.3 Carlini Wagner

The CW attack, proposed by [7], is considered to be one of the most powerful attacks in defeating neural network models. It is often used as a benchmark algorithm to evaluate the vulnerability of a model, and also to assess the strength of an adversarial data generation technique. The evaluations conducted by the authors show that the CW attack fails the popular defensive distillation mechanism, which is another potential reason for its robustness. The L2 attack, implemented in this

work, is available in the Cleverhans library [26]. Algorithm 3 presents the flow of steps involved in the L2 attack.

3.2 Contemporary Datasets

The datasets used for this experiment are the same as the ones discussed in Chapter 2, which are the UNSW-NB15, Bot-IoT, and CSE-CIC-IDS2018. However, for this experiment, around 20% of the CSE-CIS-IDS2018 was used, which is more than the size of data that was used for the experiment in Chapter 1.

3.3 Classification Algorithms

The study summarized in this chapter works with multiclass classification as all the datasets used in the experiment have multiple classes. To sync well with the nature of the datasets, four efficient classification algorithms have been chosen, namely, multi-layer perceptron (MLP), decision tree (DT), random forest (RF), and support vector machine (SVM). Tables 3.1 - 3.4 summarize the hyperparameters used for the classifiers [27]. The *criterion* parameter signifies how the algorithm chooses to split the decision tree. In other words, it helps the algorithm choose the best feature to split the data at each node of the tree. The *max_depth* parameter represents the maximum number of levels in the tree. The *n_estimators* hyperparameter represents the number of decision trees used in the random forest algorithm. The value of parameter C indicates the width of the margin used to separate the classes and controls the trade-off between obtaining low training and testing errors. The *random_state*

parameter is used to control the random number generation. When set to an integer value, it usually represents the seed for the random number generator. The *loss* parameter represents the type of loss function used to penalize misclassifications made by the SVM algorithm. To perform multi-class classification, the OneVsRestClassifer function is used to fit one classifier per class.

Table 3.1: Hyperparameters for multi-layer perceptron algorithm.

Parameter	**Value**
dropout	0.4
dense layer 1	256
dense layer 2	128
activation	relu
loss	categorical_crossentropy
optimizer	adam
output layer activation	softmax

Table 3.2: Hyperparameters for decision tree algorithm.

Parameter	**Value**
criterion	gini
max_depth	12

Table 3.3: Hyperparameters for random forest algorithm.

Parameter	**Value**
n_estimators	200
random_state	4
min_samples_split	10

Table 3.4: Hyperparameters for support vector machine algorithm.

Parameter	Value
C	1
random_state	42
loss	hinge

3.3.1 OnevsRest Classification

This is a method that enables multi-class classification by using binary classification algorithms. The process involves splitting a multi-class dataset into multiple binary classification subsets/problems. Each binary classifier is then trained with each of the binary classification subsets. Meaning, the model gets trained with each class, to learn all the classes present in a dataset.

3.4 Experimental Setup and Evaluation

3.4.1 Software Specifications

The programming setup is based on Python 3.6.5, Scikit-learn v0.19.1 library [28], Tensorflow v1.13.2 [29], and Keras v2.1.5 [30]. For the implementation of the attack algorithms, Cleverhans v3.0.1 library [26] has been used.

3.4.2 Data Pre-Processing

There are two steps of pre-processing implemented in this work - the OneHot Encoding, and the Min-Max Normalization.

3.4.2.1 One-Hot Encoding

This step was performed to convert the entire data to a numerical format. There are some features in each dataset that have non-numerical values, for example, categorical data. The One-Hot encoding method helps address this scenario.

3.4.2.2 Min-Max Normalization

This technique was applied to all the datasets to scale the values in each of them between 0 and 1. Since different features in a dataset might have values distributed on different scales, this technique helps convert all the values to a common scale and eliminate outliers, if any. Additionally, the attack methods require that all the features are within a common range, to be effective [25].

3.4.3 Experiment

There are two stages implemented in the experiment: 1) training a learning algorithm with original data; 2) generating adversarial samples from the original data. In the first stage, training and testing phases are carried out, as shown in Figure 3-1 [31]. In both phases, the original data is pre-processed. MLP is used as the baseline learning algorithm. Therefore, baseline results are obtained when MLP is tested with the data (original or adversarial), and for evaluation purposes, each of the other algorithms (DT, RF, and SVM) are implemented over the baseline algorithm.

Figure 3-2 [31] outlines the second stage of the experiment. There are training and testing phases in the second stage as well. The main difference here is that,

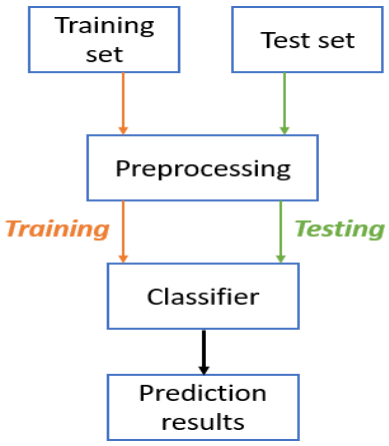

Figure 3-1: Stage 1 of the experiment.

in the testing phase, after the test data is pre-processed, it is fed to the MLP, and each of the attack algorithms is implemented to add adversarial perturbations to the test data. The obtained adversarial test set is forwarded to the classifier for final predictions. The attacks are performed by targeting the malign classes in the chosen datasets, under white-box settings. Tables 3.5 - 3.7 specify the parameters set for each of the attacks. The parameter *theta* is the perturbation added for modification of data. *Gamma* denotes the maximum percentage of perturbed features. The *clip_min* parameter minimum value for clipping the modified data, and the *clip_max* parameter is the maximum value for clipping. The *eps* (epsilon) parameter indicates the attack step size, in other words, it is a factor for scaling perturbations. For the CW attack, the *binary_search_steps* parameter indicates the number of times binary search is performed to find an optimal tradeoff-constant between the norm of a perturbation and the confidence score of the classification. *Max_iterations* parameter represents the maximum number of iterations; a larger value usually produces lower distortion

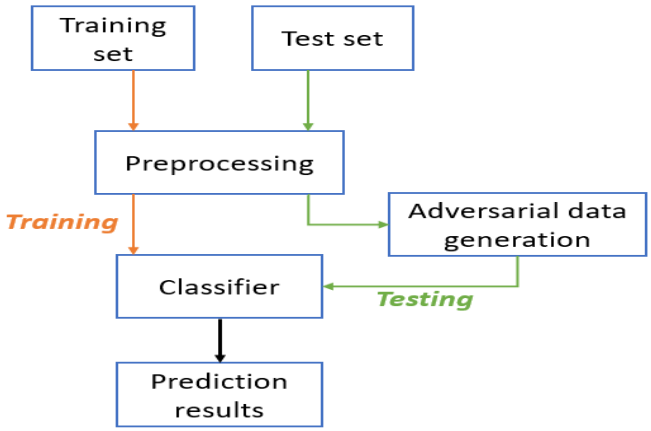

Figure 3-2: Stage 2 of the experiment.

results. The *learning_rate* parameter is the learning rate for the attack. *Batch_size* indicates the number of attacks to be run simultaneously. The *initial_const* value is the initial tradeoff-constant between the size of a perturbation and the confidence score of the classification.

Table 3.5: Hyperparameters for JSMA attack.

Parameter	**Value**
theta	1
gamma	0.1
clip_min	0
clip_max	1

3.4.4 Evaluation Metrics

As the last step in the evaluation, each classifier is tested with the original test set and then with the adversarial samples. The same process is followed for every

Table 3.6: Hyperparameters for FGSM attack.

Parameter	**Value**
eps	0.3

Table 3.7: Hyperparameters for CW attack.

Parameter	**Value**
binary_search_steps	2
max_iterations	100
learning_rate	0.2
batch_size	1
initial_const	10

dataset. The metrics used for evaluation are Accuracy, Area Under the Curve (AUC), F1 score, and Recall.

3.4.5 Analysis of Evaluation Results

Tables 3.8 - 3.11 present the performance results for the evaluation metrics chosen for this experiment when the model is trained with the UNSW-NB15 dataset. Tables 3.12 - 3.15 present the results when the model is trained with the Bot-IoT dataset. The results of the model when trained with the CSE-CIC-IDS2018 dataset are shown in Tables 3.16 - 3.19 [31]. The remaining paragraphs in this section analyze the results from the following perspectives - dataset, adversarial algorithm, and classification model.

The scores of the model, when trained with UNSW-NB15 dataset, went down

Table 3.8: Summary of accuracy scores in normal (baseline) and adversarial settings for UNSW-NB15 dataset.

Classifier	**Baseline Accuracy**	**JSMA**	**FGSM**	**CW**
MLP	0.72	0.39	0.38	0.21
SVM	0.59	0.22	0.26	0.23
DT	0.64	0.57	0.19	0.15
RF	0.64	0.64	0.25	0.28

Table 3.9: Summary of F1-scores in normal (baseline) and adversarial settings for UNSW-NB15 dataset.

Classifier	**Baseline F1-score**	**JSMA**	**FGSM**	**CW**
MLP	0.73	0.45	0.35	0.33
SVM	0.68	0.30	0.29	0.31
DT	0.69	0.62	0.32	0.24
RF	0.73	0.68	0.38	0.43

with respect to all the performance metrics. The JSMA algorithm modified 11% of the features in this dataset to implement the attack and took 8 minutes on average for adversarial data generation. The FGSM algorithm modified 78% of the features in this dataset for the attack but took only about 5 seconds to generate adversarial data. The main goal of FGSM is to quickly generate adversarial samples without worrying much about the number of features perturbed, unlike JSMA which attempts to perturb the least possible number of features to cause desired misclassification. The results obtained justify the respective behaviors of these algorithms. On the other hand, the CW attack, which is computationally heavier than the other two algorithms, modified around 65% of the features and took about 50 minutes to

Table 3.10: Summary of recall scores in normal (baseline) and adversarial settings for UNSW-NB15 dataset.

Classifier	**Baseline Recall**	**JSMA**	**FGSM**	**CW**
MLP	0.72	0.42	0.40	0.25
SVM	0.60	0.23	0.31	0.26
DT	0.66	0.62	0.38	0.22
RF	0.65	0.56	0.24	0.28

Table 3.11: Summary of AUC scores in normal (baseline) and adversarial settings for UNSW-NB15 dataset.

Classifier	**Baseline AUC**	**JSMA**	**FGSM**	**CW**
MLP	0.90	0.58	0.62	0.55
SVM	0.89	0.31	0.63	0.61
DT	0.84	0.80	0.53	0.51
RF	0.92	0.92	0.83	0.78

generate adversarial samples. From the standpoint of the classification algorithms under each of the attacks, the SVM model was the most affected one under the JSMA attack whereas the RF classifier was the most robust one among the three. DT model was the most affected one under the settings of both FGSM and CW attacks whereas the RF model was the most robust in both cases. To summarize, the RF classifier was more resistant to all three attack settings than the other two classifiers and was followed by MLP, SVM, and DT. The highest baseline (normal) accuracy was obtained using the MLP classifier while SVM gave the least baseline accuracy. CW attack had the highest overall impact on the model whereas JSMA had the least impact of all.

Table 3.12: Summary of accuracy scores in normal (baseline) and adversarial settings for Bot-IoT dataset.

Classifier	**Baseline Accuracy**	**JSMA**	**FGSM**	**CW**
MLP	0.91	0.39	0.36	0.34
SVM	0.94	0.48	0.40	0.48
DT	0.99	0.45	0.48	0.65
RF	0.99	0.86	0.47	0.60

Table 3.13: Summary of F1-scores in normal (baseline) and adversarial settings for Bot-IoT dataset.

Classifier	**Baseline F1-score**	**JSMA**	**FGSM**	**CW**
MLP	0.99	0.76	0.40	0.55
SVM	1.0	0.77	0.33	0.58
DT	0.99	0.61	0.46	0.67
RF	0.99	0.96	0.42	0.57

Tables 3.12 - 3.15 show the decline in the performance scores of the model when trained with the Bot-IoT dataset. JSMA algorithm modified 43% of the features in this dataset to implement the attack and took 14 minutes on average for adversarial data generation. The FGSM algorithm modified 52% of the features in this dataset for the attack and took about 20 seconds to generate adversarial data. The CW attack also modified around 52% of the features and took about 2 hours to generate adversarial samples. As Bot-IoT dataset is larger than UNSW-NB15, all three adversarial algorithms took longer times to generate adversarial data from Bot-IoT. From the standpoint of the classification algorithms under each of the attacks, the DT model was the most affected one under the JSMA attack while the RF classifier

Table 3.14: Summary of recall scores in normal (baseline) and adversarial settings for Bot-IoT dataset.

Classifier	**Baseline Recall**	**JSMA**	**FGSM**	**CW**
MLP	0.99	0.93	0.39	0.57
SVM	1.0	0.93	0.41	0.59
DT	1.0	0.45	0.40	0.60
RF	0.99	0.95	0.41	0.57

Table 3.15: Summary of AUC scores in normal (baseline) and adversarial settings for Bot-IoT dataset.

Classifier	**Baseline AUC**	**JSMA**	**FGSM**	**CW**
MLP	0.98	0.48	0.97	0.96
SVM	0.99	0.50	0.98	0.95
DT	0.99	1.0	0.97	0.97
RF	0.99	0.50	0.98	0.95

was the least affected one. SVM model was the most affected one under the settings of both FGSM and CW attacks. Both DT and RF classifiers were almost equally resistant, and the least affected under FGSM attack, while DT was the least affected under CW attack. To summarize, the SVM classifier was overall the most affected model under the attack settings and the RF classifier was the least affected one. The highest baseline accuracy scores were obtained using RF and DT classifiers and MLP gave the least scores. FGSM attack had the highest impact on this dataset overall while JSMA had the least impact of all.

Tables 3.16 - 3.19 show the decline in the performance scores of the model when trained with the CSE-CIC-IDS2018 dataset. This dataset is the largest of all three,

Table 3.16: Summary of accuracy scores in normal (baseline) and adversarial settings for CSE-CIC-IDS2018 dataset.

Classifier	**Baseline Accuracy**	**JSMA**	**FGSM**	**CW**
MLP	0.62	0.39	0.38	0.35
SVM	0.61	0.58	0.61	0.20
DT	0.88	0.47	0.85	0.38
RF	0.92	0.84	0.91	0.81

Table 3.17: Summary of F1-scores in normal (baseline) and adversarial settings for CSE-CIC-IDS2018 dataset.

Classifier	**Baseline F1-score**	**JSMA**	**FGSM**	**CW**
MLP	0.89	0.43	0.85	0.51
SVM	0.66	0.39	0.64	0.47
DT	0.91	0.11	0.89	0.69
RF	0.94	0.59	0.92	0.83

therefore, took the adversarial algorithms very long to generate adversarial data. JSMA algorithm modified 42% of the features in this dataset to implement the attack and took around 10 hours on average for adversarial data generation. The FGSM algorithm modified 85% of the features in this dataset for the attack and took around 6 hours to generate adversarial data. The CW attack modified 86% of the features and took about 14 hours to generate adversarial samples. The DT model was the most affected one under the JSMA attack while the SVM classifier was the least affected one. RF model was the least affected one under the settings of both FGSM and CW attacks. SVM was the most affected model under FGSM, and DT was the most affected under the CW attack. To summarize, the DT classifier was

Table 3.18: Summary of recall scores in normal (baseline) and adversarial settings for CSE-CIC-IDS2018 dataset.

Classifier	**Baseline Recall**	**JSMA**	**FGSM**	**CW**
MLP	0.90	0.58	0.85	0.81
SVM	0.97	0.64	0.92	0.95
DT	0.94	0.12	0.93	0.82
RF	0.91	0.57	0.91	0.81

Table 3.19: Summary of AUC scores in normal (baseline) and adversarial settings for CSE-CIC-IDS2018 dataset.

Classifier	**Baseline AUC**	**JSMA**	**FGSM**	**CW**
MLP	1.0	0.42	0.97	0.99
SVM	1.0	0.44	0.91	0.99
DT	1.0	0.44	1.0	0.99
RF	1.0	0.44	1.0	0.99

overall the most affected model under the attack settings and the RF classifier was the least affected one. The highest baseline accuracy scores were obtained using RF classifiers and MLP gave the least scores. CW attack had the highest impact on this dataset overall while FGSM had the least impact of all.

Although all three attack algorithms influenced the performances of the IDS models, the variations in their impacts can help investigate the characteristics of the datasets used. Based on the results, the overall impact on the CSE-CIC-IDS2018 dataset is relatively lesser, which is followed by the UNSW-NB15 dataset, and then the Bot-IoT. One possible reason behind this pattern is the number of features in the datasets. With a lesser number of features, the vulnerabilities may increase.

Looking at the overall results from the classifiers' end, the RF classifier stood almost steadily robust against all three attacks, with all three datasets. Further, the RF classifier produced the highest baseline accuracy scores for two out of three datasets. Another significant behavior is that the impact patterns are not uniform among different evaluation metrics. It means an adversary should target a performance metric and design the attack accordingly to bring down the performance as desired. Showing a severe impact on a metric need not produce the same level of impact on a different metric. The influence of the CW attack, which is considered one of the most sophisticated and powerful algorithms to deceive image classification models, does not seem much stronger than that of the other two attack techniques for IDS data, as all of them had nearly the same level of impact.

3.5 Conclusion and Future Work

There is a need to study the properties of the available modern IDS datasets and switch from old and outdated datasets to contemporary ones. As important as it is to analyze how useful modern datasets are in machine learning-based research, it is essential to know how useful they are under adversarial settings. This work studies three recently published IDS datasets, namely, UNSW-NB15, BotIoT, and CIC-IDS2018 under the light of three adversarial attack algorithms, namely, JSMA, FGSM, and CW. The performance is evaluated using multiple classifiers - SVM, DT, and RF - while using MLP as the baseline classifier. The experimental results have shown that RF is relatively more robust in adversarial environments, and in terms of the datasets, CIC-IDS2018 has offered more resilience to the classifiers. The impacts

of the attacks have been varying with the datasets and classifiers.

I would like to extend this study in multiple directions. One of them is to develop defense mechanisms to tackle black-box attacks. Another direction is to analyze how successful adversarial attacks can be on IDS research datasets, in general. This direction is a relatively long-term research plan I have. The real success of an adversarial attack lies in generating valid deceptive samples that can bypass detection and launch the attacks they are meant for. I would like to investigate whether the adversarial data samples generated using the attacks are retaining the original attack properties of the genuinely malign data. As highlighted in [32], some factors might help validate the adversarial samples to a certain extent. Further experimentation is needed to investigate more into this avenue.

Algorithm 3: CW L2 attack

Require: Input: Model M, clean input image x, target class t, confidence parameter c, L2 weight parameter κ, binary search iterations n_{binary}, maximum iterations $n_{\max}$, learning rate α, and initial perturbation δ_0

Ensure:

Output: Adversarial image x'

$l_{\min} \leftarrow 0, l_{\max} \leftarrow \infty$ ▷ $l_{\min}$ and $l_{\max}$ are initial lower and upper bounds for binary search threshold.

$x' \leftarrow x + \delta_0$

for $i \leftarrow 1$ to n_{binary} **do**

  mid $\leftarrow (l_{\min} + l_{\max})/2$

  $\delta \leftarrow \text{clamp}(\text{project}x, \kappa(x' - x))$ ▷ The clamp function is to make sure the perturbation values are within the allowed range.

  $x\_\text{adv} \leftarrow \text{clip}(x + \delta, 0, 1)$

  for $j \leftarrow 1$ to $n_{\max}$ **do**

    $g \leftarrow \nabla_x L(M(x_{\text{adv}}), t) - \nabla_x L(M(x_{\text{adv}}), \hat{t})$ ▷ $L()$ refers to the loss function.

    $\hat{g} \leftarrow \frac{g}{\|g\|_2 + \epsilon}$

    $\delta \leftarrow \delta + \alpha \cdot \text{sign}(\hat{g})$

    $\delta \leftarrow \text{clamp}(\text{project}x, \kappa(x' - x))$

    $x\_\text{adv} \leftarrow \text{clip}(x + \delta, 0, 1)$

    if $M(x_{\text{adv}}) = t$ **then**

      $l_{\max} \leftarrow$ mid

      $x' \leftarrow x_{\text{adv}}$

    else

      $l_{\min} \leftarrow$ mid

    end if

  end for

end for

return x'

Chapter 4

An Approach to Improve the Robustness of Machine Learning based Intrusion Detection System Models Against the Carlini-Wagner Attack

This work proposes and evaluates a defense mechanism that can improve the resistance of IDSs against adversarial white-box evasion attacks. The experiment is evaluated with the CSE-CIC-IDS2018 dataset against the Carlini-Wagner (CW) attack. The defense framework is based on a generative adversarial network (GAN). The following sections describe this work in detail.

4.1 A Brief Background

4.1.1 Dataset Overview

Some essential parameters that determine the quality of an IDS dataset are - the framework used for generating the traffic scenarios; diversity of attack scenarios captured; anonymity; the variety of protocols included; capturing complete network interactions; configurations in the network; feature set; labeled data samples; heterogeneity; and metadata [33]. We have chosen the CSE-CIC-IDS2018 dataset for this effort, considering all these characteristics.

4.1.2 The Carlini Wagner Attack

In addition to the explanation provided for this attack in Chapter 3, this section briefly describes the attack algorithm, in simpler terms. CW is a targeted evasion attack, proposed by [7] to counter defensive distillation, a popular defense mechanism. This white-box attack has turned out to be more potent than many other white-box attack algorithms in the research community, rendering most defensive methodologies ineffective. Formulating an optimization problem to produce misclassification is the foundation of an adversarial attack algorithm. In the CW attack, the problem of creating adversarial examples is represented as follows:

$$\begin{aligned} &minimize \;\; d(x, x+\eta) \\ such \;\; that \;\; &Y(x+\eta) = T \;\; (this \;\; is \;\; constraint \;\; 1) \\ where \;\; &x+\eta \;\; \epsilon \;\; [0,1]^n \;\; (this \;\; is \;\; constraint \;\; 2) \end{aligned} \tag{4.1}$$

In equation 4.1 [32], x is an input data point, η is the perturbation, d is the metric of the distance between a real input and its corresponding adversarial form, Y is the classification function, T is the target class chosen by the adversary, and n is the number of dimensions in the feature space. Constraint-1 ensures the misclassification of the data point, while constraint-2 ensures the adversarial sample generated is within the normalized boundaries of the dataset [34] [35]. The authors define the objective function in seven different ways and choose the optimal one, based on which is the closest to the target-class misclassification. The distance metrics are specified using Lp norms (i.e., L0, L2, L∞) [32] [35].

4.1.3 Generative Adversarial Network

A GAN is a deep learning (DL)-based generative model that was first described by Ian Goodfellow et al. [36]. Its purpose is to build adversarial samples, very similar to original data, from an input dataset. A GAN is implemented using two neural networks that challenge each other, as in a two-player game. It attempts to mimic a data distribution and allows a model to learn more from available data.

A random number (random noise) is given as input to the Generator, which generates the samples that are comparable to those in the dataset and forwards them to the Discriminator, which examines the samples and predicts whether they are original data or generated ones [37]. The Discriminator learns the original data characteristics and, based on this knowledge, makes decisions on the data passed to it by the Generator. In simple terms, the Generator keeps improving its adversarial samples to make it difficult for the Discriminator to identify them. The Discriminator

tries to learn more and correctly identify the adversarial samples introduced by the Generator. There are some limitations to GAN's capacities. Its Vanishing Gradient Problem is one of them, where the Generator reaches a saturation point and can no longer produce new samples to trick the Discriminator. Meaning, the Discriminator identifies the samples created by Generator with high confidence values leaving no gradient for the Generator [38]. This issue can be averted by carefully balancing the race/training between the Generator and Discriminator networks and making sure the Discriminator is not over-trained. Ensuring that the Discriminator is trained to an optimal level for every iteration of Generator training is critical in avoiding such issues. Various approaches for GAN training stabilization are discussed and analyzed in [39].

Using GAN within its capable boundaries, two conventional loss metrics are considered - Generator loss and Discriminator loss. The following sections extend the discussion on how GAN and its characteristics are used in this experiment.

4.1.4 Classification Algorithms

The classification algorithms used in this work are decision tree (DT), random forest (RF), and support vector machine (SVM).

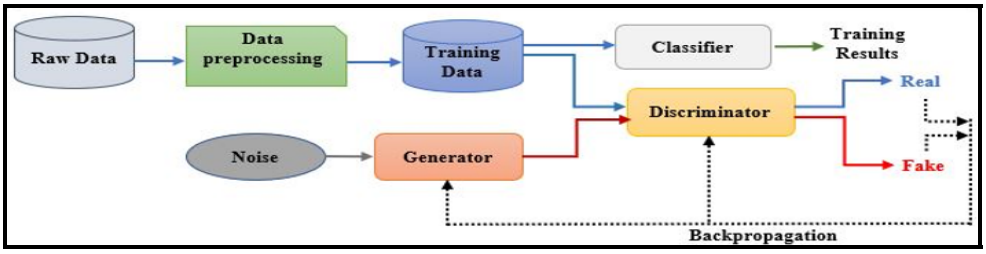

Figure 4-1: Training phase of the experiment.

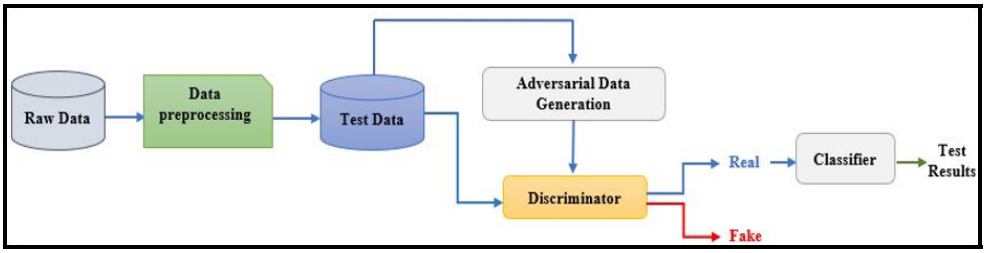

Figure 4-2: Testing phase of the experiment.

4.2 Architecture and Workflow of the Proposed Approach

This experiment consists of a series of independent evaluations. At first, the performance of the baseline model is evaluated in non-adversarial settings, i.e., with the original dataset, using each of the chosen classification algorithms. Next, the model is evaluated by implementing the CW attack. Then, the defense technique is incorporated into the model, after which its performance is evaluated. Figures 4-1 and 4-2 [32] outline the training and testing phases, respectively. The remainder of this section discusses classification behaviors and technical specifications involved in the experiment, and briefly describes the components that constitute the architecture.

4.2.1 Classification Goal

The following are the classification goals of the experimental setup: a) the discriminator is to classify adversarial and nonadversarial data, therefore, this is binary classification, b) the final IDS is to classify whether the data instances it receives are benign or malign, which is a binary classification, too.

4.2.2 Resources Used for the Experiment

The computer that was used for the experiment had the following specifications: Intel Xeon Processor E5-2697 @ 2.6 GHz, 128 GB RAM, and Microsoft Windows 64-bit. The software specifications include Python 3.6.5, Scikit-learn 0.24.2, Tensorflow 1.13.2, and Keras 2.1.5. The CW attack was implemented using the Cleverhans 3.0.1 library.

4.2.3 Configuration of the Internal Components

4.2.3.1 Generator Configuration

The Generator is made up of a five-layer NN with a random noise input layer of size 78 and three internal layers with 128, 128, and 256 nodes respectively, and uses the ReLU activation function for learning the real data distribution. It has an output layer that generates a data sample of size 78, which is comparable to the original data. A generator loss function is computed based on the prediction results provided by the Discriminator and is fed back to the Generator for it to improve in a way to minimize the loss value.

4.2.3.2 Discriminator Configuration

The Discriminator network has an input layer that accepts inputs from the Generator and training dataset during the learning phase, and four hidden layers with 128, 128, 64, and 64 nodes respectively, with ReLU activation function and one output layer with sigmoid function to result in an output value between 0 and 1. The output value represents how confident the Discriminator is about the sample being fake or real. Based on this score, the records with a value of less than 0.5 are classified as fake and are separated from the test data, while the ones classified as real are forwarded to the IDS to get classified further as benign or malign. If the Discriminator correctly classifies a fake sample, the loss function assists the Generator in adjusting its weights to craft more deceptive samples for the Discriminator. If the Discriminator fails to detect a fake sample, the loss function assists the Discriminator in adjusting its weights to improve the detection of fake samples.

4.2.3.3 The Core IDS

This classifier is trained on the training data and is the IDS model that receives the data that is forwarded by the Discriminator, and decides whether each input is benign or malign.

4.3 Implementation and Evaluation

4.3.1 Preprocessing

The preprocessing stage involves the removal of records with NaN and Infinity values, and the deletion of the Timestamp column from the dataset. The next step in this stage is to use OneHot encoding to convert nominal values to numerical data. The CSE-CIC-IDS2018 dataset has a total of 79 features, and after One-Hot encoding, the number of total features becomes 94. Then, min-max normalization is applied to all features to scale the data between 0 and 1. This step is essential because the dataset has numeric features whose values might have been derived from various distributions, have varied scales, and are occasionally affected by outliers, which can make some classifiers render incorrect results. After normalization, 70% of the dataset is separated for training and the remaining 30% for testing. The training set is used to train the ML model. In the testing phase, prediction results are obtained by providing the test-set instances as inputs to the model. To let the model learn from data that has multiple classes the OneVsRest classification is used.

4.3.2 Evaluation of Baseline Model in Adversarial Settings

The generated adversarial data, along with the original test data is presented to the model for evaluation in adversarial settings. Tables 4.1 - 4.4 summarize the results obtained in normal and adversarial settings.

Tables 4.1 - 4.4 show that all the classifiers exhibit a drop in performance scores in the presence of adversarial inputs. The accuracy drop in the case of SVM classifier

Table 4.1: Accuracy scores in normal and adversarial settings.

Classifier	**Baseline Accuracy**	**Adversarial**
DT	0.72	0.49
RF	0.91	0.81
SVM	0.61	0.25

Table 4.2: F1 scores in normal and adversarial settings.

Classifier	**Baseline F1-Score**	**Adversarial**
DT	0.86	0.71
RF	0.94	0.83
SVM	0.66	0.51

is the highest of all three, with a decline of around 59%, while the drop for DT and RF are 32% and 11% respectively. The highest baseline accuracy is obtained from the RF classifier and the least accuracy is from the SVM model. Further, the drop in performance when trained using RF algorithm is the least among the three. The results suggest that RF algorithm has the potential to render promising performance while also showing in-built resistance to adversarial inputs. This gives a good motive to use RF for future experiments toward enhancing the resistance of IDS models to adversarial environments.

The F1 scores, as shown in Table 4.2, also drop in a similar fashion to that of the accuracy scores with respect to the classifiers' standpoint. The SVM classifier has the highest drop of 23%, followed by DT with 17% and RF with 12%. The RF classifier has the highest baseline F1 score and the least performance drop of all. Another

Table 4.3: Recall scores in normal and adversarial settings.

Classifier	**Baseline Recall**	**Adversarial**
DT	0.94	0.80
RF	0.91	0.81
SVM	0.97	0.88

Table 4.4: AUC in normal and adversarial settings.

Classifier	**Baseline AUC**	**Adversarial**
DT	1	0.99
RF	1	0.99
SVM	1	0.99

observation here is that the impact of the CW attack on F1 scores is relatively lower than the impact on the accuracy scores based on the overall percentages of drop between the two scenarios.

The performance scores in terms of recall show different behavior. SVM classifier gives the highest baseline recall value and is followed by DT with RF being the model with the least recall. The percentage drop is the least - 9% - when trained using SVM. After SVM, RF has the least drop of 11% in the score while the DT model's score dropped by 15%. The percentages of drop are further lower for recall scores than those for F1 scores. This implies that the impact of the attack varies with metrics.

The baseline AUC and the percentage drop are almost the same for all three classifiers, as shown in Table 4.4.

4.3.3 Evaluation of the Defense

The architecture for the proposed defense mechanism uses GAN to identify and separate the adversarial samples and pass the non-adversarial data to the core IDS classifier, which determines if the data is benign or malign. The data is separated into training and testing sets during the preprocessing phase. In the training phase, random noise is provided to the Generator to produce fake samples. The Discriminator is trained with the training set and hones its ability to detect adversarial samples by learning the fake data produced by the Generator. The core classifier is also trained with the training set. The GAN, as a whole, becomes more competent as the number of learning/training iterations increase, to not only generate complex fake data but also to identify it correctly. In the testing phase, the Discriminator is tested with test data. The test data comprises the data from the test set and the adversarial samples that are generated by running the CW attack on the test set. The instances that the Discriminator predicts as real are carried forward to the core IDS classifier, separating the instances that are classified as fake.

4.4 Evaluation Results and Performance Comparison

Tables 4.5 - 4.8 outline the performance of the model when tested with a combination of original and adversarial test data before and after the defense is incorporated. The results presented in Section 4.3.2 show a clear drop in the performance of the baseline classifier when it is tested with adversarial samples. The results in this sec-

tion summarize how the performance changes when the defense mechanism is added to the IDS model.

Table 4.5: Accuracy scores in adversarial settings without and with defense.

Classifier	**Adversarial without Defense**	**Adversarial with Defense**
DT	0.49	0.60
RF	0.81	0.82
SVM	0.25	0.36

Table 4.6: F1 scores in adversarial settings without and with defense.

Classifier	**Adversarial without Defense**	**Adversarial with Defense**
DT	0.71	0.67
RF	0.83	0.84
SVM	0.51	0.43

Table 4.5 shows the improvement in the accuracy scores in the presence of the defense. There is approximately 22% performance gain for DT model, only around 1% gain for RF model, and around 44% for SVM model, the highest gain of all. The GAN-based defense has enhanced the accuracy scores with all three classifiers but the improvement is very little for the RF classifier. The changes in the performance with respect to the remaining metrics are not as favorable as the accuracy scores. The F1 scores for DT and SVM classifiers, as shown in Table 4.6, have declined further in the presence of defense. The same behavior can be noticed for recall and AUC scores as shown in Tables 4.7 and 4.8. However, when trained using RF algorithm,

Table 4.7: Recall scores in adversarial settings without and with defense.

Classifier	**Adversarial without Defense**	**Adversarial with Defense**
DT	0.80	0.65
RF	0.81	0.83
SVM	0.88	0.74

Table 4.8: AUC in adversarial settings without and with defense.

Classifier	**Adversarial without Defense**	**Adversarial with Defense**
DT	0.99	0.90
RF	0.99	0.90
SVM	0.99	0.90

the accuracy, F1, and recall scores have improved in the presence of defense. This behavior is to be investigated to figure out why the scores have dropped further and more importantly, whether the network configuration of the GAN can influence this behavior. Additionally, based on the fake data filtered out by the discriminator during evaluation, on average, 75% of the adversarial data is correctly identified as fake. As part of future work, the work further improves the detection abilities. Figure 4-3 [32] summarizes the improvements in the accuracy scores.

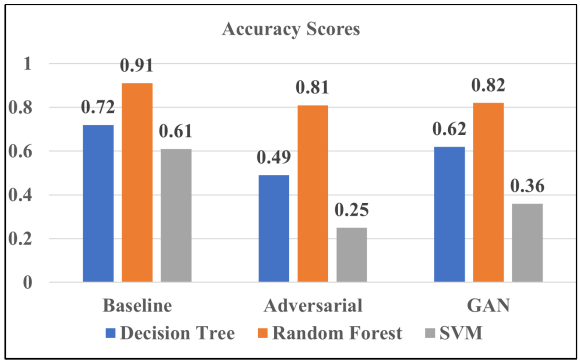

Figure 4-3: An overall comparison of accuracy scores.

4.4.1 Comparison with Related Work

Rui Shu et al. [40] propose a technique called Omni, an ensemble of unexpected models to tackle adversarial environments. Their ideology behind employing unexpected models is to keep the distance between their core prediction mechanism and the adversary's target model's mechanism as large as possible. The authors present the experimentation results conducted using five different adversarial white-box evasion attacks on five different cybersecurity datasets. The CW is one of the attacks, and the CSE-CIC-IDS2018 dataset is one of the datasets they have evaluated their approach with. The results from this approach were superficially compared with those presented in [40]. From the results presented by Rui et al., the baseline accuracy, i.e., under normal settings is 94.48%, and the final accuracy after implementing the Omni defense on the model is 75.23%. The highest baseline accuracy in this approach is 91% and is from the RF classifier, and the corresponding final accuracy (with the proposed defense in place) is 82%. However, since Omni is designed to be agnostic about the type of adversarial evasion attack used, it creates an avenue in

this approach to extend its applicability to all kinds of adversarial evasion attacks.

4.5 Conclusion

This part of the work proposes a defense mechanism to improve the resistance of ML-based IDSs against the CW attack by using a GAN-based architecture. The dataset used for the work is CSE-CIC-IDS2018, as it reflects a good number of modern network characteristics. The experimentation results show that there is an improvement in the performance of the IDS model with the proposed mechanism. Evaluation results of two scenarios when subjected to adversarial inputs are presented - the baseline model alone, and the baseline model with GAN. The results indicate that this approach of defense improves the accuracy scores in the adversarial environment created by a powerful white-box attack such as the CW. One of the ideas for future work is to improve the detection rate of the proposed GAN-based network in identifying the adversarial data while also investigating how an imbalanced dataset like CSE-CIC-IDS2018 can impact the performance in adversarial environments. The architecture will be extended by introducing feature squeezing methods and the performance will be thoroughly evaluated.

Chapter 5

Towards the Defense of Machine Learning based Intrusion Detection Systems Against Adversarial Black-box Attacks

Cyberattacks are ever-evolving and given the rapid advancements in science and technology, the levels of difficulty in tackling the attacks keep going up. Bad actors are developing more sophisticated techniques to implement their malicious activities such as disrupting the operations of organizations, stealing sensitive information, getting hold of resources by unauthorized means, etcetera. Therefore, any research progress in the domain of cybersecurity should be achieved by consciously considering the bad actors and their strategies.

Most of the machine learning (ML)-based intrusion detection systems (IDSs)

proposed in the literature have been designed for performance and relatively very little consideration has been given to their robustness and security. This becomes the greatest advantage to the bad actors. With the concept of adversarial machine learning (AML), which is one of the well-known vulnerabilities of ML techniques, it has become relatively easier for adversaries to deceive and break the ML-based systems without having much knowledge of the target models [41]. Such a scenario can be accomplished through adversarial black-box attacks. AML and its black-box variant of attacks are a serious concern in cybersecurity considering how crucial the network defenders are, and their increasing dependency on ML mechanisms. The AML algorithms are capable of affecting the functionalities of ML-powered IDSs deployed in real-time environments. Therefore, with the motivation to develop mechanisms that can mitigate the impact of adversarial black-box attacks, this research moves forward in the direction of analyzing the impact of such attacks on ML-based IDSs, and towards proposing defense techniques to make IDSs more robust to black-box attacks. This chapter discusses the experiments performed using the CSE-CIC-IDS2018 dataset and the evaluation of the defense using the Evolutionary black-box attack algorithm. The generative adversarial network (GAN)-based defense framework proposed in Chapter 4 is used for countering the black-box attack in this portion of the work. The remainder of this chapter is organized as follows. Section 5.1 covers the relevant background of the elements involved in the experimentation. Section 5.2 discusses the architecture of the proposed defense in the current context. Section 5.3 presents the experimentation setup and methodology. Section 5.4 discusses and analyzes the evaluation results. Section 5.5 concludes the chapter

and discusses ideas for future work.

5.1 Background

5.1.1 Adversarial Black-box Attacks

Unlike a white-box attack, an adversarial black-box attack does not need any information about its target model to implement the attack. A black-box attack algorithm can proceed in two ways, broadly classified, either on a transferability basis [42], or on a query basis [43]. For this work, a query-based black-box attack is chosen.

5.1.2 Query-Based Black-Box Attack: Genetic Algorithm

A query-based black-box attack is based solely on the model's input-output behavior, without relying on the gradient calculation or the confidence scores of the target model. In this type of attack, an attacker feeds inputs to the model and observes its outputs, then uses this information to craft adversarial examples that can fool the model into producing incorrect outputs. In a practical setting, an adversary typically has a limited number of queries they can make to the model, meaning they can only make a certain number of input-output observations. This makes the attack more challenging because the adversary has to carefully choose which inputs to use for each query in order to maximize the chances of crafting a successful adversarial example.

The attack chosen for this experiment is an evolutionary algorithm (EA) called

the genetic algorithm (GA). In this approach, the algorithm starts by selecting a set of initial inputs (or population) that are fed into the target model to observe its output. The attacker then applies genetic operators such as mutation, crossover, and selection to these inputs to generate a new set of candidate adversarial examples. The candidate examples are then evaluated by feeding them to the target model to see if they are successful in fooling it. The best-performing candidate examples are then selected and used to create the next generation of inputs, and the process is repeated until a possibly optimal adversarial example is generated. Evolutionary algorithms, in general, can generate adversarial examples that are highly tailored to the specific model and data being used. However, they can also be computationally expensive and time-consuming.

The following are the generalized steps involved in an evolutionary algorithm:

- Generate initial population: Create a population of N individuals with randomly assigned values for the decision variables.
- Evaluate each individual: A fitness value of each individual is determined by querying the target model with the individual and observing the output provided by the model. It can be the classification accuracy of the model on each individual.
- Select high-performing individuals: Select individuals with high fitness values.
- Recombination: Combine the decision variables of the selected individuals to create new individuals.
- Mutation: Randomly perturb the decision variables of some individuals.

- Evaluation of new individuals: Evaluate the fitness of the new individuals.
- Replacement: Replace the least fit individuals in the population with the new individuals.

The steps of evaluation, recombination, mutation, and replacement are repeated until a pre-determined stopping criterion is met. The stopping criterion can be based on various factors such as reaching a certain number of generations, achieving a satisfactory fitness level, or exceeding a set computation time.

5.1.3 Classification Algorithms

Random Forest (RF) is the classification algorithm used for attack data generation and evaluation. RF, Decision Tree (DT) and Support Vector Machine (SVM) are used for training and evaluating the IDS model. The reason behind choosing the RF classifier for attack generation is its efficiency and robustness observed in the evaluation results from the previous experiments of this work, as presented in the previous chapters.

5.1.4 Dataset

As highlighted in Chapter 4, considering various characteristics of the data [33], CSE-CIC-IDS2018 is used throughout the experiment, i.e., for training the IDS model and for adversarial data generation.

5.2 Architecture of the Proposed Defense

The defense framework used for this experiment uses a generative adversarial network (GAN) along with adversarial training as a reinforcement. Figure 5-1 outlines the higher-level idea of this work.

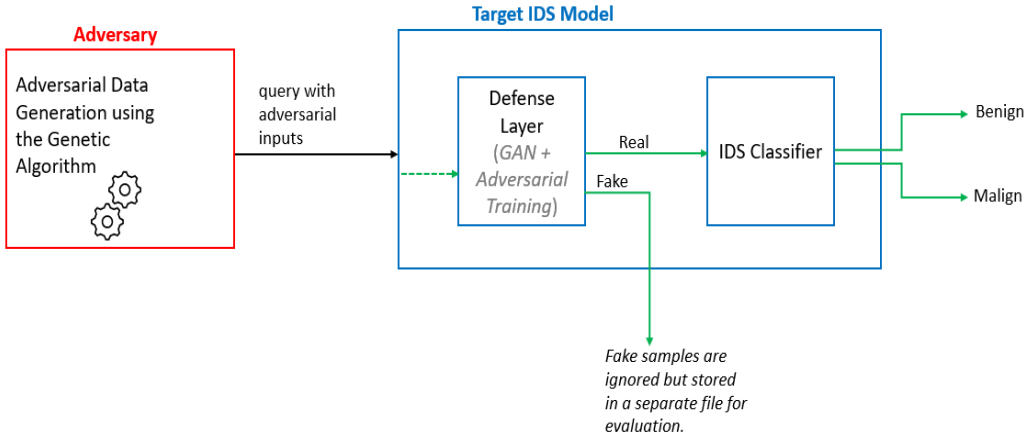

Figure 5-1: An outline of the architecture.

The defense layer consists of a generator neural network (NN) and a discriminator NN, which compete with each other resembling a two-player game. Figure 5-2 and Figure 5-3 show the internal mechanism of the GAN-based defense layer, in two phases - training and testing. During the training, there is an additional step of adversarial training involved, for which, some portion of the adversarial data generated using the genetic algorithm is separated and provided to the discriminator for training. The idea behind this additional training is to make the discriminator more aware of the fake samples and to evaluate whether such additional knowledge would make the target model more robust to a wide range of black-box attacks.

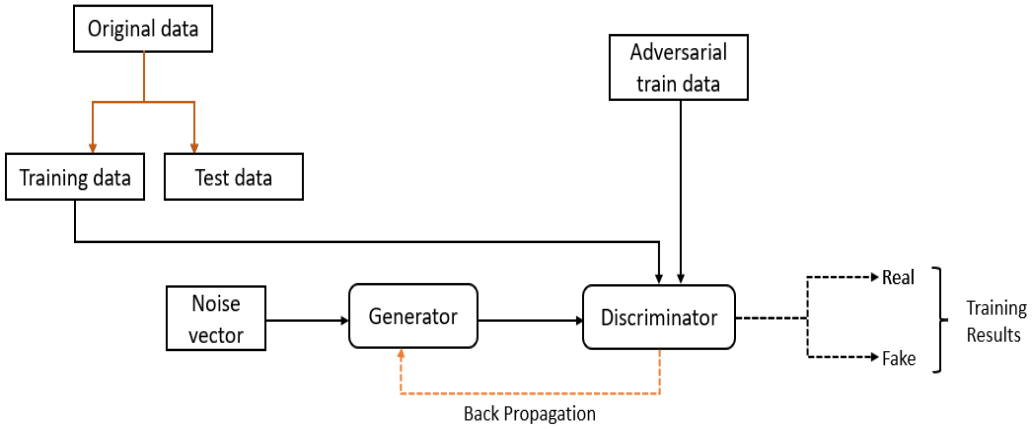

Figure 5-2: Inside the GAN-based defense layer: Training phase

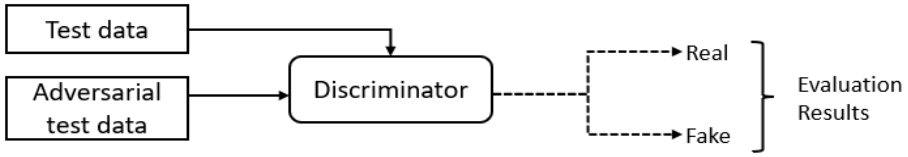

Figure 5-3: Inside the GAN-based defense layer: Testing phase

During the training phase, the generator, discriminator, and IDS model are all trained well. A random noise vector is provided to the generator network to generate fake samples that fit within the data distribution of the original dataset. The discriminator is trained with the original train set and the fake samples created by the generator. To summarize, during the training, the generator trains towards efficiency in creating complex fake samples, the discriminator trains towards efficiency in classifying the real and fake samples, and the IDS model trains towards efficiency in classifying the benign and malicious samples.

5.2.1 Ideology for the Detection of Adversarial Inputs

The fundamental concept around which a query-based attack is built is that it takes several queries for such an attack to create a successful adversarial sample. The key is that each of the several queries is designed in a way that the sample is very close to the original input in a data distribution. An adversarial sample can only be designed if all or most of the iterative queries are answered by the target model. If a model can identify and not respond to the queries that are part of an attack, the adversary will not be able to craft an adversarial input, at least not in a feasible amount of time. This is the thought process driving the development of this defense.

5.3 Experiment Setup and Methodology

This section provides details of the methodology followed for this experiment in multiple modules.

5.3.1 Data Preprocessing

The original dataset is preprocessed before providing it to the ML model, either for training or validation purposes, to improve the reliability of its data interpretation. The following steps are performed for preprocessing in this work.

- Removal of records with NaN and Infinity values.
- Deletion of the Timestamp column from the dataset.

- One-Hot encoding to convert nominal values to numerical data.
- Separation of data into training (70%) and test (30%) sets.
- Separation of features and labels from training and test sets.
- Min-Max normalization to all features in the train and test sets to scale the data between 0 and 1.

5.3.2 Train the IDS

The IDS classifier is trained well with the original dataset using the three learning algorithms specified in Section 5.3.1, as shown in Figure 5-4. During this training, the IDS classifier becomes efficient in classifying benign and malicious data. Tables 5.1 - 5.3 specify the hyperparameter values of the classification algorithms during model training. As mentioned in Section 3.3 of Chapter 3, the *criterion* parameter signifies how the algorithm chooses to split the decision tree. The *max_depth* parameter represents the maximum number of levels in the tree. The *n_estimators* value represents the number of decision trees used in the random forest algorithm. The parameter C determines the width of the margin used to separate the classes and controls the trade-off between obtaining low training and testing errors. *Random_state* parameter is used to control the random number generation. When set to an integer value, it usually represents the seed for the random number generator. The *loss* parameter represents the type of loss function used to penalize misclassifications made by the SVM algorithm.

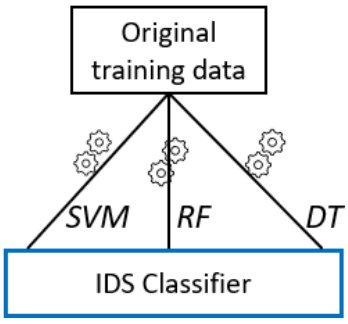

Figure 5-4: Training the IDS classifier with original data.

Table 5.1: Hyperparameters for decision tree algorithm.

Parameter	**Value**
criterion	gini
max_depth	12

5.3.3 Adversarial Data Generation

Adversarial data generation involves a series of steps. The very first step is to preprocess the dataset as described in Section 5.3.1. A substitute classification model is used for the adversarial data generation and its partial evaluation to reduce the number of queries made to the target model. The substitute model is trained with the preprocessed training data using the RF classification algorithm. The core process of the attack starts with the creation of an initial population. A random population is created based on the data distribution of the dataset used. Each data point, hereafter referred to as an individual, in the population looks similar to a record in the dataset. Each individual is evaluated using a fitness function to determine how well it performs against the target model. This evaluation is achieved by passing each individual to the substitute RF model as a test input, and retrieving its accuracy,

Table 5.2: Hyperparameters for random forest algorithm.

Parameter	**Value**
n_estimators	200
criterion	gini
max_depth	20

Table 5.3: Hyperparameters for support vector machine algorithm.

Parameter	**Value**
C	1
random_state	42
loss	hinge

F1 score, recall, and AUC values. The higher the values are, the more the fitness of an individual is, which thereby means more misclassified adversarial examples. Each batch of individuals, i.e., the population, is passed to the genetic algorithm wherein the individuals are combined with one another, mutated, and new ones are generated. The fitness values of new ones are measured and the individuals with the least fitness values in the previous population are replaced with high-performing individuals from the new batch. For this experiment, the attack took 20 iterations, meaning, 20 generations of individuals were created to arrive at one best individual. Each of such best individuals can be considered an adversarial perturbation to be added to the original data. Adding the perturbation to the original data provides adversarial data. The resultant adversarial data is clipped to make sure it is present within the valid range of the original dataset. In this case, the adversarial data is clipped to be between 0 and 1.

Figure 5-5: Adversarial data is separated into training and test sets.

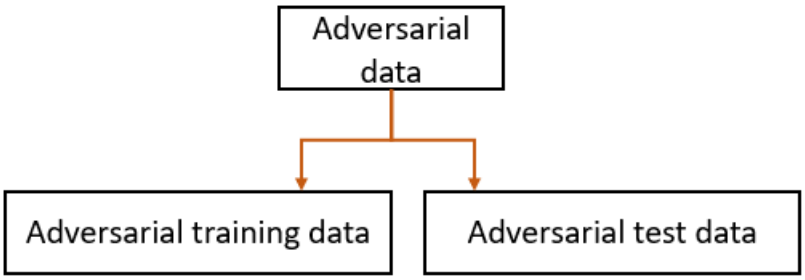

Figure 5-6: Adversarial data is generated by interacting with the target black-box model and using the local model.

5.3.4 Querying the Target Model with Adversarial Data

Once a batch of adversarial data is ready, it is divided into training and test sets for this experiment. The algorithm queries the target model with the adversarial test set and observes the outputs provided by the model. The target model here represents only the IDS classifier without a defense layer. Therefore, the GA interacts directly with the IDS classifier in this module of the experiment, and the performance scores of the IDS classifier are presented later in this paper. Figure 5-5 and Figure 5-6 show an overview of the process described in this section.

The hyperparameters of the RF algorithm used for the local substitute model during the adversarial data generation are provided in Table 5.4.

Table 5.4: The hyperparameters for random forest algorithm used during adversarial data generation.

Parameter	**Value**
n_estimators	100
criterion	gini
max_depth	10
random_state	0

5.3.5 Prepare the Defense Layer

This module is for training the components present in the defense layer. As mentioned earlier in this paper, the proposed defense framework uses GAN. A GAN has two neural networks (NN) namely, a generator and a discriminator. The role of a generator is primarily in the training phase of a GAN. Its purpose is to create fake data that resembles the given original dataset, in this case, the CSE-CIC-IDS2018, by taking a random noise vector as input. The purpose of a discriminator is to distinguish between the fake data created by the generator and the original data. During the training phase, the generator network is trained to craft more and more realistic samples to increase the error rates of the discriminator; while the discriminator is trained to correctly classify the fake samples of the generator and the real data from the dataset. During this training, the discriminator sends feedback to the generator about how close its fake samples are to the original data, which is used by the generator to adjust its parameters and create more realistic data in the next iteration. Meanwhile, the discriminator also keeps adjusting its parameters in every iteration to reduce its error rates in detecting fake samples.

An additional step introduced in the proposed defense process is augmented ad-

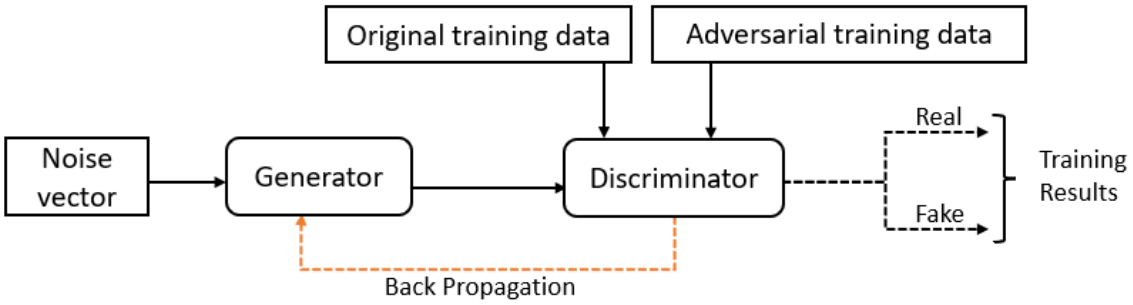

Figure 5-7: Preparation of the defense layer by training the GAN.

versarial training. The discriminator, in addition to being trained with the original training set and the fake data coming from the generator, is also trained with the adversarial training set that was separated after adversarial data generation. The reason behind performing the augmented training is to provide additional knowledge of adversarial inputs to the discriminator and make it more prepared for the validation phases. Such additional knowledge coming from the adversarial data generated by a powerful algorithm can promote the discriminator's ability to distinguish between real and fake data. Figure 5-7 outlines the GAN training. Tables 5.5 and 5.6 summarize the network configurations of the generator and discriminator respectively.

Table 5.5: Architecture of GAN's generator network.

Layer	**Number of neurons**	**Activation**
Dense_1	256	LeakyReLU
Dense_2	512	LeakyReLU
Dense_3	1024	LeakyReLU
Dense_4	2048	LeakyReLU
Output	79	tanh

Table 5.6: Architecture of GAN's discriminator network.

Layer	**Number of neurons**	**Activation**
Dense_1	2048	LeakyReLU
Dense_2	1024	LeakyReLU
Dense_3	512	LeakyReLU
Dense_4	256	LeakyReLU
Output	1	sigmoid

5.3.6 Evaluate the Target Model with Defense

In this module, the defense framework is plugged into the target model and the performance of the target model is assessed by providing the adversarial test inputs. In this scenario, the IDS classifier processes only the data that is forwarded to it by the discriminator of the defense layer. Meaning, the adversarial samples that are correctly classified by the discriminator are not acted upon, therefore, the adversarial algorithm does not receive any response from the target model.

5.4 Evaluation Results: Discussion and Analysis

This section presents evaluation results from the modules of the experiment discussed in Section 5.3.

Tables 5.7 - 5.10 summarize the performance scores of the IDS classifier when tested with original test data and later with adversarial test data. The scores presented here are the average values of five rounds of this experiment. Under adversarial settings, there is a significant drop in the performance scores overall, with values of almost all the metrics dropping by at least 50%. Tables 5.11 - 5.14 summarize the

Table 5.7: Comparison of accuracy scores of IDS classifier in normal and adversarial settings.

Classifier	**Normal Accuracy**	**Adversarial**
RF	0.95	0.39
DT	0.87	0.26
SVM	0.72	0.23

Table 5.8: Comparison of F1 scores of IDS classifier in normal and adversarial settings.

Classifier	**Normal F1-score**	**Adversarial**
RF	0.91	0.37
DT	0.91	0.29
SVM	0.77	0.28

evaluation results of the target model under adversarial settings before and after the incorporation of the defense layer.

Figures 5-8 - 5-11 show the changes observed in various performance metrics of the IDS classifier under the following three scenarios - when tested with original test data; when tested with a combination of original and adversarial test data in the

Table 5.9: Comparison of recall scores of IDS classifier in normal and adversarial settings.

Classifier	**Normal Recall**	**Adversarial**
RF	0.91	0.36
DT	0.90	0.25
SVM	0.91	0.30

Table 5.10: Comparison of AUC scores of IDS classifier in normal and adversarial settings.

Classifier	**Normal AUC**	**Adversarial**
RF	1	0.5
DT	1	0.47
SVM	1	0.49

Table 5.11: Comparison of accuracy scores of IDS classifier in adversarial settings without and with defense.

Classifier	**Accuracy in Adversarial Settings without Defense**	**Accuracy in Adversarial Settings with Defense**
RF	0.39	0.84
DT	0.26	0.79
SVM	0.23	0.61

Table 5.12: Comparison of F1 scores of IDS classifier in adversarial settings without and with defense.

Classifier	**F1-scores in Adversarial Settings without Defense**	**F1-scores in Adversarial Settings with Defense**
RF	0.37	0.82
DT	0.29	0.82
SVM	0.28	0.65

Table 5.13: Comparison of recall scores of IDS classifier in adversarial settings without and with defense.

Classifier	**Recall in Adversarial Settings without Defense**	**Recall in Adversarial Settings with Defense**
RF	0.36	0.83
DT	0.29	0.82
SVM	0.30	0.87

Table 5.14: Comparison of AUC scores of IDS classifier in adversarial settings without and with defense.

Classifier	**AUC in Adversarial Settings without Defense**	**AUC in Adversarial Settings with Defense**
RF	0.5	0.89
DT	0.47	0.90
SVM	0.49	0.92

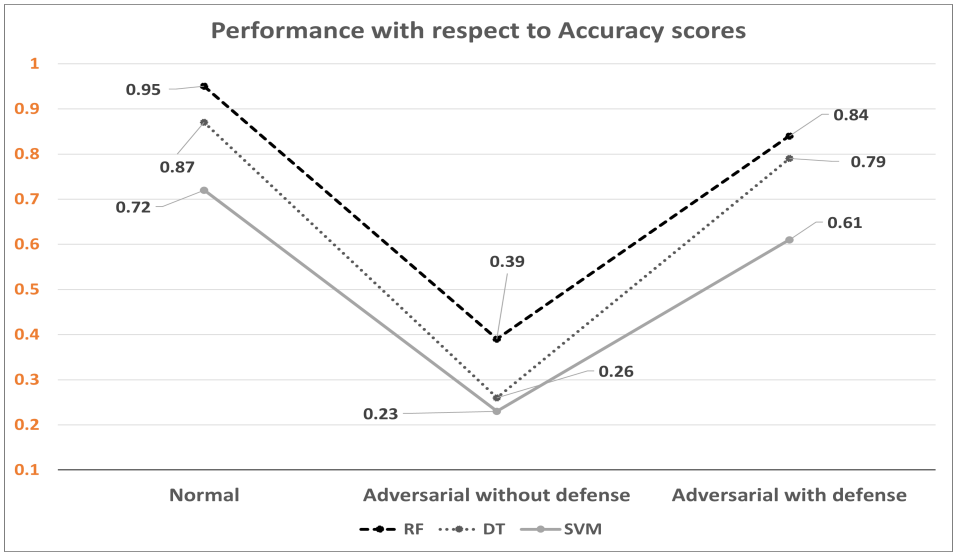

Figure 5-8: Changes in the performance of target model with respect to accuracy scores.

absence of defense; when tested with a combination of original and adversarial test data in the presence of defense.

The results from defense show a promising improvement in the resistance of the target system to adversarial data, enabling it to perform almost as efficiently as it did in the absence of adversarial data. An important note here is that the strange behavior observed with the defense proposed in Chapter 4 is not repeated in this experiment. There is a significant performance gain with respect to all the metrics considered. Although the configurations of generator and discriminator networks seem to influence the said behavior, it is necessary to further experiment and analyze how impactful GAN's configuration can be on the quality of defense, which is one of the ideas for future work.

The attack results show that the adversarial data has brought down the perfor-

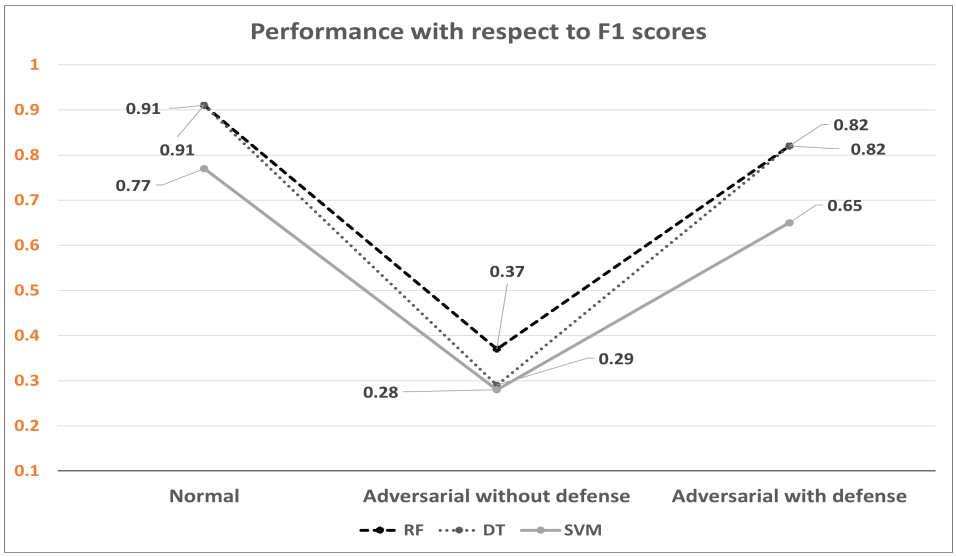

Figure 5-9: Changes in the performance of target model with respect to F1 scores.

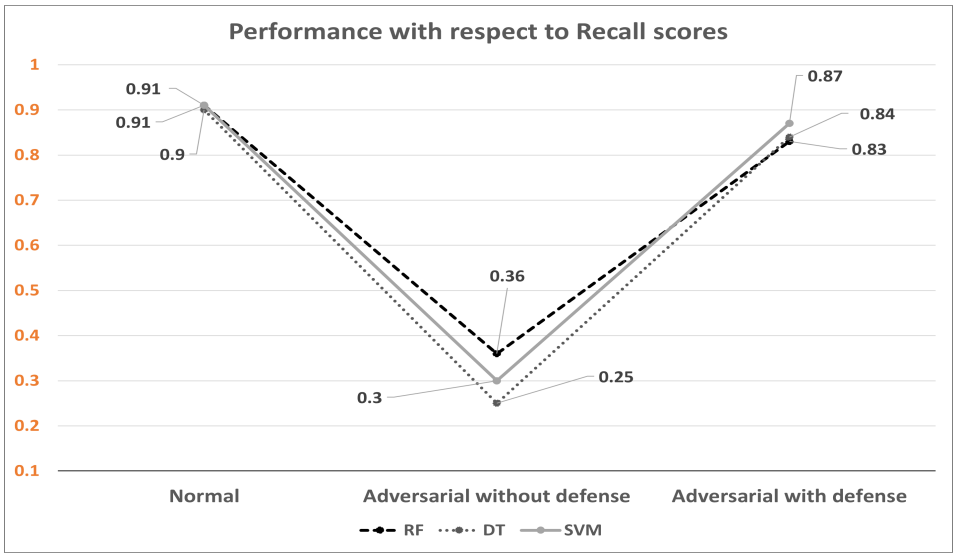

Figure 5-10: Changes in the performance of target model with respect to recall scores.

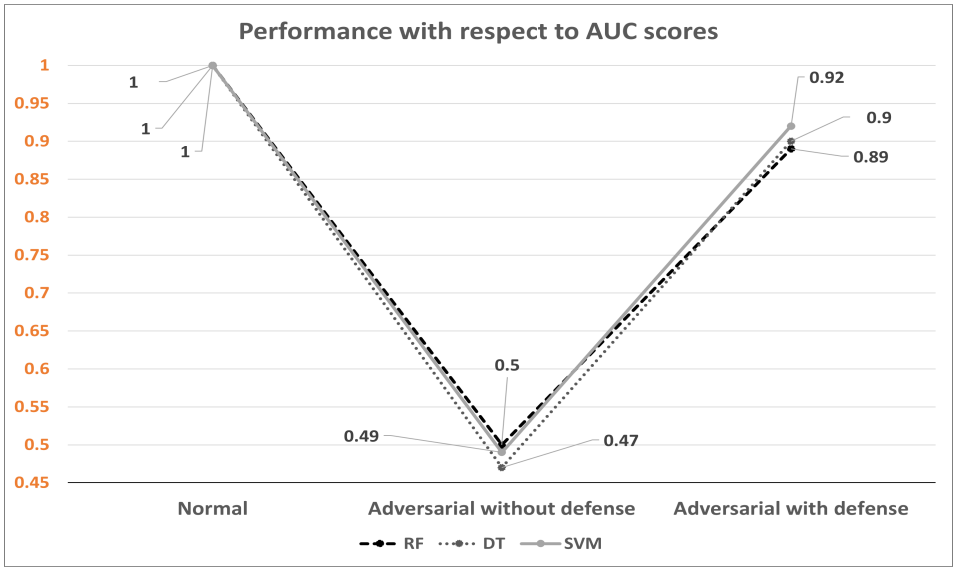

Figure 5-11: Changes in the performance of target model with respect to AUC scores.

mance of the target model by a huge extent, and interestingly, the impact of this attack was more severe than that of the white-box attack performed in earlier stages of this research, even at the very first attempt of evaluation. This attack did not need many optimizations to increase the impact on the performance. The values presented in the results are the average values from five rounds of optimizations in adversarial data generation. However, the proposed defense mechanism has produced promising improvements in the performance of the model when evaluated with all three classification algorithms. The approach can be further extended as future work to make it more robust and applicable against a broad class of black-box attack algorithms.

Analyzing the performance from the perspective of classification algorithms, scores were the highest and the percentage drop in the evaluation metrics overall was the least when the IDS classifier was trained with the RF algorithm. Based on the eval-

uation of the defense layer's performance, on average, 87% of the adversarial samples were correctly classified as fake and filtered out. The performance of the defense layer is promising and may be further improved through our future work in this direction.

The work presented by Muhammad Usama et al. [44] discusses creating a GAN-based adversarial attack and also proposes a GAN-based defense to thwart their attack. However, the evaluation of their work was performed using KDD99 dataset [45]. Based on the evaluation results of their GAN-based attack and defense mechanisms, the approach of this work gives a much better performance considering the following factors - a) the normal/baseline scores of the target model produced in this work are much higher than those from their experiment; b) the drop in the scores from the attack performed in the current work is far greater than that presented in their results; c) the final scores after incorporating the current defense proposed are, for almost all the metrics, higher. The work presented by [46] is relevant to this work and it uses the same dataset for evaluations. The attack used in their paper is ZOO (zeroth order optimization) algorithm which is also a black-box algorithm but is different from an evolutionary attack in its approach and characteristics. ZOO is a type of optimization algorithm that estimates the gradients (or derivatives) using just the function evaluations instead of using explicit gradient or derivative values of the objective function. The evolutionary attack used in this work and the ZOO algorithm are fundamentally different approaches. The comparison of the current results with those of [46] and other recent related works is one of the ideas for future work.

5.5 Conclusion and Future Work

This chapter discusses the work to mitigate the influence of adversarial black-box attacks on ML-based IDSs. A powerful evolutionary attack algorithm known as the genetic algorithm is implemented to analyze the performance of the IDS model under normal settings where it has no defense mechanism in place to resist the attack. The defense proposed here follows a GAN-based approach combined with additional training wherein the discriminator network of GAN is trained with a subset of the adversarial data generated using the genetic algorithm. The experiment results are presented summarizing the performance of the model under normal and adversarial settings, followed by the evaluation results with the proposed defense in place.

For future work, the defense shall be evaluated using other sophisticated black-box attacks. Another idea is to use an IDS model that is deployed in a real-time environment, for example, a publicly accessible cloud-based IDS model to compare its resistance to black-box attacks with the model (equipped with defense) evaluated in this work. The work also proceeds in the direction of enhancing the detection capacity of the proposed defense framework and thereby making it more reliable and harder to penetrate.

Chapter 6

Conclusion and Future Work

This Ph.D. dissertation is primarily about identifying the challenges concerning machine learning (ML) based intrusion detection systems (IDSs) and proposing approaches towards addressing those challenges. This dissertation is a study and analysis of the behaviors of ML-based IDSs in various normal and adversarial settings where a normal setting is defined as the scenario where the IDS model encounters only original or genuine test data, and an adversarial setting is defined as a scenario where the IDS model receives adversarial (deceptive) data in addition to original test data.

The following are the concerns related to ML-based IDSs highlighted in this work:

- Majority of research work is performed and evaluated using decades-old datasets that lack modern traffic behaviors and characteristics.
- A relatively little focus is given to the security and reliability of ML-based IDSs as compared to their detection efficiencies.

6.1 Contributions

To address the above-listed concerns, here are the contributions made by this research through various phases.

- While emphasizing the need to generate datasets that reflect modern traffic scenarios and to study the characteristics of such modern/contemporary datasets available to research community, this work studies three recently published benchmark datasets and evaluates the performance of IDS models when trained with those datasets.

- Based on the evaluations, this work identifies that class imbalance in datasets can influence the performance of IDS models by making them render biased predictions and more false predictions. Through this observation, this work emphasizes the need to focus on the quality of datasets as the performance of a model hugely depends on the quality of data it is trained upon.

- This work studies the problems - in particular, the concept of adversarial ML (AML) - caused by the vulnerabilities of ML-based techniques in the domain of IDSs.

- Two types of adversarial attacks, namely, white-box and black-box algorithms are studied and their impact on the behaviors of IDSs is analyzed.

- Two defense mechanisms - one addressing adversarial white-box attacks and the other for adversarial black-box attacks - are proposed.

- Experimentation methodologies and evaluation results are presented in detail for all the phases of research conducted. The results are sufficiently discussed and analyzed.

6.2 Directions for Future Work

There are multiple directions for future work because of the vastness of the topic pursued in this research. Listed below are the major directions in which this research continues.

- Development of a research and educational testbed to conduct various experiments related to this research, as also listed further below.
- Further understand the concerns or limitations related to the existing contemporary IDS datasets.
- Using the above-mentioned testbed, create a standard dataset that captures the modern behaviors of benign and malicious traffic in a way that addresses the identified limitations in existing IDS datasets.
- Conduct a deeper investigation into the viability of AML algorithms in modifying network traffic data.
- Improve the proposed defense mechanisms to make them more robust and applicable to a wide range of adversarial attacks.

References

[1] R. A. Kemmerer and G. Vigna, "Intrusion detection: a brief history and overview," *Computer*, vol. 35, no. 4, pp. supl27–supl30, 2002.

[2] P. Innella *et al.*, "The evolution of intrusion detection systems," *Tetrad Digital Integrity*, pp. 1–15, 2001.

[3] A. Z. Li and D. Barton, "A brief history of machine learning in cybersecurity," Available at https://www.securityinfowatch.com/cybersecurity/article/21114214/a-brief-history-of-machine-learning-in-cybersecurity (2019/11/14).

[4] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, "Intrusion detection model using machine learning algorithm on big data environment," *Journal of Big Data*, vol. 5, no. 1, sep 2018. [Online]. Available: https://doi.org/10.1186/s40537-018-0145-4

[5] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 99–108.

[6] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in

Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2013, pp. 387–402. [Online]. Available: https://doi.org/10.1007/978-3-642-40994-3_25

[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2017. [Online]. Available: https://doi.org/10.1109/sp.2017.49

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[9] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[10] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, mar 2016. [Online]. Available: https://doi.org/10.1109/eurosp.2016.36

[11] M. Rigaki, "Adversarial deep learning against intrusion detection classifiers," 2017.

[12] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[13] K. Sadeghi, A. Banerjee, and S. K. S. Gupta, "A system-driven taxonomy of attacks and defenses in adversarial machine learning," *IEEE Transactions on*

Emerging Topics in Computational Intelligence, vol. 4, no. 4, pp. 450–467, aug 2020. [Online]. Available: https://doi.org/10.1109/tetci.2020.2968933

[14] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38 367–38 384, 2018. [Online]. Available: https://doi.org/10.1109/access.2018.2854599

[15] AWS. (2018) A realistic cyber defense dataset (cse-cic-ids2018). [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018/

[16] V. Kanimozhi and T. P. Jacob, "Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing," in *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, apr 2019. [Online]. Available: https://doi.org/10.1109/iccsp.2019.8698029

[17] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications, 2018. [Online]. Available: https://doi.org/10.5220/0006639801080116

[18] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*. IEEE, nov 2015. [Online]. Available: https://doi.org/10.1109/milcis.2015.7348942

[19] M. S. et. al., "Network based intrusion detection using the UNSW-NB15 dataset," *International Journal of Computing and Digital Systems*, jan 2019.

[20] S. Dwibedi, M. Pujari, and W. Sun, "A comparative study on contemporary intrusion detection datasets for machine learning research," in *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2020, pp. 1–6.

[21] C. Mantas, S. Castellano, J.G. andMoral-García *et al.*, "A comparison of random forest based algorithms: random credal random forest versus oblique random forest," *Soft Computing*, vol. 23, p. 10739–10754, 2019. [Online]. Available: https://doi.org/10.1007/s00500-018-3628-5

[22] V. Kecman, "Support vector machines–an introduction," in *Support vector machines: theory and applications*. Springer, 2005, pp. 1–47.

[23] N. Ketkar, *Introduction to Deep Learning*. Berkeley, CA: Apress, 2017, pp. 1–5. [Online]. Available: https://doi.org/10.1007/978-1-4842-2766-4_1

[24] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 16–25.

[25] N. Martins, J. M. Cruz, T. Cruz, and P. H. Abreu, "Analyzing the footprint of classifiers in adversarial denial of service contexts," in *Progress in Artificial Intelligence*. Springer International Publishing, 2019, pp. 256–267. [Online]. Available: https://doi.org/10.1007/978-3-030-30244-3_22

[26] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, P. McDaniel *et al.*, "cleverhans v2. 0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, vol. 10, 2016.

[27] Y. Pacheco and W. Sun, "Adversarial machine learning: A comparative study on contemporary intrusion detection datasets," in *Proceedings of the 7th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications, 2021. [Online]. Available: https://doi.org/10.5220/0010253501600171

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[30] F. Chollet *et al.* (2021) Keras. [Online]. Available: https://github.com/keras-team/keras

[31] M. Pujari, P. Yulexis *et al.*, "A comparative study on the impact of adversarial machine learning attacks on contemporary intrusion detection datasets," *Springer Nature Computer Science*, vol. 3, 2022.

[32] M. Pujari, B. P. Cherukuri, A. Y. Javaid, and W. Sun, "An approach to improve

the robustness of machine learning based intrusion detection system models against the carlini-wagner attack," in *2022 IEEE International Conference on Cyber Security and Resilience)*. IEEE, 2022.

[33] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2018, no. 1, pp. 177–200, 2018.

[34] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[35] B. Poudel, "Explaining the carlini & wagner attack algorithm to generate adversarial examples." [Online]. Available: https://medium.com/@iambibek/explanation-of-the-carlini-wagner-c-w-attack-algorithm-to-generate-adversarial-examples-6c1db8669fa2

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[37] S. Msika, A. Quintero, and F. Khomh, "Sigma: Strengthening ids with gan and metaheuristics attacks," *arXiv preprint arXiv:1912.09303*, 2019.

[38] S. A. Barnett, "Convergence problems with generative adversarial networks (gans)," *arXiv preprint arXiv:1806.11382*, 2018.

[39] M. Wiatrak, S. V. Albrecht, and A. Nystrom, "Stabilizing generative adversarial networks: A survey," *arXiv preprint arXiv:1910.00927*, 2019.

[40] R. Shu, T. Xia, L. Williams, and T. Menzies, "Omni: Automated ensemble with unexpected models against adversarial evasion attack," *arXiv preprint arXiv:2011.12720*, 2020.

[41] D. Shu, N. O. Leslie, C. A. Kamhoua, and C. S. Tucker, "Generative adversarial attacks against intrusion detection systems using active learning," in *Proceedings of the 2nd ACM workshop on wireless security and machine learning*, 2020, pp. 1–6.

[42] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[43] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[44] M. Usama, M. Asim, S. Latif, J. Qadir *et al.*, "Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems," in *2019 15th international wireless communications & mobile computing conference (IWCMC)*. IEEE, 2019, pp. 78–83.

[45] M. S. Al-Daweri, K. A. Zainol Ariffin, S. Abdullah, and M. F. E. Md. Senan,

"An analysis of the kdd99 and unsw-nb15 datasets for the intrusion detection system," *Symmetry*, vol. 12, no. 10, p. 1666, 2020.

[46] S. Alahmed, Q. Alasad, M. M. Hammood, J.-S. Yuan, and M. Alawad, "Mitigation of black-box attacks on intrusion detection systems-based ml," *Computers*, vol. 11, no. 7, 2022. [Online]. Available: https://www.mdpi.com/2073-431X/11/7/115

Appendix A

Source Code Snippets for Adversarial White-box Attacks and Defense

A.1 JSMA

```
def Adversarial_JSMA(X_train_scaled, X_test_scaled, y_train, y_test, classes,
batch, predictions, sess, x, y, model, target_class):

    source_samples = X_test_scaled.shape[0]


    # Jacobian-based Saliency Map
    results = np.zeros((classes, source_samples), dtype='i')
    # Rate of perturbed features for each test set example and target class
    perturbations = np.zeros((classes, source_samples), dtype='f')
    wrap = KerasModelWrapper(model)

    results = np.zeros((classes, source_samples), dtype='i')
```

```
  # Rate of perturbed features for each test set example and target class
perturbations = np.zeros((classes, source_samples), dtype='f')
X_adv = np. zeros (( source_samples , X_test_scaled . shape [1]) )

jsma = SaliencyMapMethod(wrap, sess=sess)
jsma_params = {'theta': 1., 'gamma': 0.1,
                   'clip_min': 0., 'clip_max': 1.,
                   'y_target': None}
X_adv = np. zeros (( source_samples , X_test_scaled . shape [1]) )

for sample_ind in range (0, source_samples ):
    # We want to find an adversarial example for each possible target class

    current_class = int (np. argmax ( y_test [ sample_ind ]))
    sample = X_test_scaled[sample_ind:(sample_ind + 1)]

    # Only target the normal class
    for target in [target_class]:
        if current_class == target_class:
            break
        # This call runs the Jacobian-based saliency map approach
        one_hot_target = np.zeros((1, classes), dtype=np.float32)
        one_hot_target[0, target] = 1
        jsma_params['y_target'] = one_hot_target
        x_advinputs = jsma.generate_np(sample, **jsma_params)
```

```
            # Check if success was achieved
            res = int(model_argmax(sess, x, predictions, x_advinputs) == target)

            # Compute number of modified features
            adv_x_reshape = x_advinputs.reshape(-1)
            test_in_reshape = X_test_scaled[sample_ind].reshape(-1)
            nb_changed = np.where(adv_x_reshape != test_in_reshape)[0].shape[0]
            percent_perturb = float(nb_changed) / x_advinputs.reshape(-1).shape[0]

            # Update the arrays for later analysis
            results[target, sample_ind] = res
            perturbations[target, sample_ind] = percent_perturb
            X_adv[sample_ind] = x_advinputs


# Compute the number of adversarial examples that were successfully found
nb_targets_tried = ((classes - 1) * source_samples)
succ_rate = float(np.sum(results)) / nb_targets_tried


# Compute the average distortion introduced by the algorithm
percent_perturbed = np.mean(perturbations[np.where(perturbations!=0)])

# Compute the average distortion introduced for successful samples only
percent_perturb_succ = np.mean(perturbations[np.where(perturbations!=0)] *
(results[np.where(perturbations!=0)] == 1))
```

```
    eval_params = {'batch_size': batch }
    accuracy_adv = model_eval (sess , x, y, predictions , X_adv , y_test ,
                               args = eval_params )
    return X_adv, accuracy_adv, succ_rate, percent_perturbed, percent_perturb_succ
```

A.2 FGSM

```
def Adversarial_FGSM(X_train_scaled, X_test_scaled, y_train, y_test, classes,
batch, predictions,sess,x,y, model,target_class):

    source_samples = X_test_scaled . shape [0]

    # Rate of perturbed features for each test set example and target class
    wrap = KerasModelWrapper(model)

    adv_inputs = X_train_scaled[:source_samples]

    # Rate of perturbed features for each test set example and target class
    X_adv = np. zeros (( source_samples , X_test_scaled . shape [1]) )
    adv_inputs = X_test_scaled[:source_samples]

    one_hot_target = np.zeros((source_samples, classes), dtype=np.float32)
    one_hot_target[np.arange(source_samples), target_class] = 1
```

```
    # Craft adversarial examples using FGSM approach
    fgsm =  FastGradientMethod(wrap, sess=sess)
    fg_params = {'eps':0.3, 'y_target':one_hot_target}
    X_adv = fgsm.generate_np(adv_inputs, **fg_params)
    eval_params = {'batch_size': np.minimum(classes, source_samples)}
    accuracy_adv = model_eval (sess , x, y, predictions , X_adv , y_test ,
                                    args = eval_params )
    return X_adv, accuracy_adv
```

A.3 CW

```
def Adversarial_CW(X_train_scaled, X_test_scaled, y_train, y_test, classes,
batch, predictions,sess,x,y, model,target_class):
    source_samples = X_test_scaled . shape [0]

    # Rate of perturbed features for each test set example and target class
    wrap = KerasModelWrapper(model)
    adv_inputs =  X_test_scaled[:source_samples]

    one_hot_target = np.zeros((source_samples, classes), dtype=np.float32)
    one_hot_target[np.arange(source_samples), target_class] = 1
    adv_ys = None

    # Craft adversarial examples using Carlini and Wagner's approach
    # Instantiate a CW attack object
    cw = CarliniWagnerL2(wrap, sess=sess)
    cw_params_batch_size = 10
```

```
cw_params = {'binary_search_steps': 2,
        'max_iterations': 100,
        'learning_rate': 0.2,
        'batch_size': 1,
        'initial_const': 10,
        'y_target': adv_ys}

X_adv = cw.generate_np(adv_inputs, **cw_params)

eval_params = {'batch_size': np.minimum(classes, source_samples)}
err = model_eval(sess, x, y, predictions, X_adv,
y_test[:source_samples],args=eval_params)
adv_accuracy = 1 - err
accuracy_adv = model_eval (sess , x, y, predictions , X_adv , y_test ,
                                    args = eval_params )
return X_adv, accuracy_adv
```

Appendix B

Source Code Snippet for GAN-based Defense

```
#Training PArams
learning_rate = 0.0002
batch_size = 78
epochs = 500

#Network params
data_dim = 78
gen_hidd_dim = 128
gen_hidd2 = 128
gen_hidd3 = 256
disc_hidd_dim  = 128
disc_hidd2 = 128
disc_hidd3 = 64
disc_hidd4 = 64
z_noise_dim = 78

def xavier_init(shape):
  return tf.random_normal(shape = shape, stddev= 1./tf.sqrt(shape[0]/2.0))
```

```
weights = {
    "disc_H" : tf.Variable(xavier_init([data_dim,disc_hidd_dim])),
        "disc_H2" : tf.Variable(xavier_init([disc_hidd_dim,disc_hidd2])),
        "disc_H3" : tf.Variable(xavier_init([disc_hidd2,disc_hidd3])),
        "disc_H4" : tf.Variable(xavier_init([disc_hidd3,disc_hidd4])),
    "disc_final": tf.Variable(xavier_init([disc_hidd4,1])),

    "gen_H": tf.Variable(xavier_init([z_noise_dim, gen_hidd_dim])),
        "gen_H2": tf.Variable(xavier_init([gen_hidd_dim,gen_hidd2])),
        "gen_H3": tf.Variable(xavier_init([gen_hidd2,gen_hidd3])),
        "gen_final": tf.Variable(xavier_init([gen_hidd3, data_dim]))
}
bias = {
    "disc_H" : tf.Variable(xavier_init([disc_hidd_dim])),
        "disc_H2" : tf.Variable(xavier_init([disc_hidd2])),
        "disc_H3" : tf.Variable(xavier_init([disc_hidd3])),
        "disc_H4" : tf.Variable(xavier_init([disc_hidd4])),
        "disc_final": tf.Variable(xavier_init([1])),

        "gen_H": tf.Variable(xavier_init([gen_hidd_dim])),
        "gen_H2": tf.Variable(xavier_init([gen_hidd2])),
        "gen_H3": tf.Variable(xavier_init([gen_hidd3])),
    "gen_final": tf.Variable(xavier_init([data_dim]))
}
#defining the Discriminator
def Discriminator(x):
```

```
  hidden_layer = tf.nn.relu(tf.add(tf.matmul(x, weights["disc_H"]),
  bias["disc_H"]))
  hidden_layer2 = (tf.add(tf.matmul(hidden_layer, weights["disc_H2"]),
  bias["disc_H2"]))
  hidden_layer3 = (tf.add(tf.matmul(hidden_layer2, weights["disc_H3"]),
  bias["disc_H3"]))
  hidden_layer4 = (tf.add(tf.matmul(hidden_layer3, weights["disc_H4"]),
  bias["disc_H4"]))
  final_layer = (tf.add(tf.matmul(hidden_layer4, weights["disc_final"]),
  bias["disc_final"]))
  disc_output = tf.nn.sigmoid(final_layer)
 #print(disc_output)
  return  disc_output

# Defining the Generator NW
def Generator(x):
  hidden_layer = tf.nn.relu(tf.add(tf.matmul(x, weights["gen_H"]),
  bias["gen_H"]))
  hidden_layer2 = (tf.add(tf.matmul(hidden_layer, weights["gen_H2"]),
  bias["gen_H2"]))
  hidden_layer3 = (tf.add(tf.matmul(hidden_layer, weights["gen_H3"]),
  bias["gen_H3"]))
  final_layer = (tf.add(tf.matmul(hidden_layer3, weights["gen_final"]),
  bias["gen_final"]))
  gen_output = tf.nn.sigmoid(final_layer)
  return gen_output
```

```
#define placeholders for external input
z_input = tf.placeholder(tf.float32, shape = [None, z_noise_dim], name =
"input_noise")
x_input = tf.placeholder(tf.float32, shape = [None, data_dim], name =
"real_noise")

#Building the Generator NW
with tf.name_scope("Generator") as scope:
  output_Gen = Generator(z_input) #G(z)
 # Building the Disc NW
with tf.name_scope("Discriminator") as scope:
    real_output_disc = Discriminator(x_input) #implements D(x)
    fake_output_disc = Discriminator(output_Gen) # implements D(G(x))

with tf.name_scope("Discriminator_Loss") as scope:
  Discriminator_Loss = -tf.reduce_mean(tf.log(real_output_disc+
  0.0001)+tf.log(1.- fake_output_disc+0.0001))

with tf.name_scope("Genetator_Loss") as scope:
  Generator_Loss = -tf.reduce_mean(tf.log(fake_output_disc+ 0.0001))
  Disc_loss_total = tf.summary.scalar("Disc_Total_loss", Discriminator_Loss)
  Gen_loss_total = tf.summary.scalar("Gen_loss", Generator_Loss)

#Making a separate dictionary for the current weights and biases
Generator_var = [weights["gen_H"], weights["gen_final"], bias["gen_H"],
bias["gen_final"]]
Discriminator_var = [weights["disc_H"], weights["disc_final"], bias["disc_H"],
```

```
bias["disc_final"]]

#Define the optimizer
with tf.name_scope("Optimizer_Discriminator") as scope:
  Discriminator_optimize = tf.train.AdamOptimizer(learning_rate =
  learning_rate).minimize(Discriminator_Loss, var_list = Discriminator_var)

with tf.name_scope("Optimizer_Generator") as scope:
  Generator_optimize = tf.train.AdamOptimizer(learning_rate =
  learning_rate).minimize(Generator_Loss, var_list = Generator_var)
```

Appendix C

Source Code Snippets for Adversarial Black-box Attack and its Defense

C.1 Genetic Algorithm

```
def evolutionary_attack_v2():

    # Load and preprocess the CSE-CIC-IDS2018 dataset
    X_train_scaled, X_test_scaled, y_train, y_test = preprocessing()

    # Define the machine learning model
    model = RandomForestClassifier()

    # Define the fitness function
    def fitness(individual, X_train_scaled, X_test_scaled, y_train, y_test):
        selected_features = [idx for idx, gene in enumerate(individual) if gene]
        x_test_individual = X_test_scaled[:, selected_features]

        model = RandomForestClassifier(n_estimators=200, random_state=0)
```

```
    model.fit(X_train_scaled[:, selected_features], y_train)

    y_pred = model.predict(x_test_individual)

    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    roc = roc_auc_score(y_test, y_pred, average='weighted', multi_class='ovo')
    return (accuracy, f1, recall, roc)

POP_SIZE = 50
GEN_NUM = 20
CXPB = 0.5
MUTPB = 0.2

# Create the genetic algorithm toolbox
creator.create("FitnessMax", base.Fitness, weights=(1.0, 1.0, 1.0, 1.0))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_bool, n=X_test_scaled.shape[1])
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", fitness, X_train_scaled=X_train_scaled,
X_test_scaled=X_test_scaled, y_train=y_train, y_test=y_test)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=MUTPB)
```

```
    toolbox.register("select", tools.selTournament, tournsize=3)

    # Create the initial population
    pop = toolbox.population(n=POP_SIZE)
    # Run the EA
    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean, axis=0)
    stats.register("min", np.min, axis=0)
    stats.register("max", np.max, axis=0)

    pop, log = algorithms.eaSimple(pop, toolbox, cxpb=CXPB, mutpb=MUTPB,
    ngen=GEN_NUM, stats=stats, halloffame=hof, verbose=True)

    # Print the best individual
    best_individual = hof[0]

    # Evaluate the best individual
    fitness_values = fitness(best_individual, X_train_scaled, X_test_scaled,
    y_train, y_test)
```

C.2 GAN-based Defense

```
# Define the size of the latent space for the generator
latent_dim = 100

# Define the generator
```

```
generator = Sequential()
generator.add(Dense(256, input_dim=latent_dim))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Dropout(0.2))
generator.add(Dense(512))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Dropout(0.2))
generator.add(Dense(1024))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Dropout(0.2))
generator.add(Dense(2048))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Dropout(0.2))
generator.add(Dense(79, activation='tanh'))

# Define the discriminator
discriminator = Sequential()
discriminator.add(Dense(2048, input_dim=79))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.2))
discriminator.add(Dense(1024))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.2))
discriminator.add(Dense(512))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.2))
discriminator.add(Dense(256))
```

```
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.2))
discriminator.add(Dense(1, activation='sigmoid'))

gan_input = Input(shape=(latent_dim,))
gan_output = discriminator(generator(gan_input))
gan = Model(gan_input, gan_output)

# Compile the discriminator
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002,
beta_1=0.5))

# Compile the GAN
gan.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002,
beta_1=0.5))

# Define a function to generate random samples from the latent space
def generate_latent_points(latent_dim, n_samples):
    x_input = np.random.randn(latent_dim * n_samples)
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

# Define a function to train the GAN
def train_gan(gan, discriminator, generator, X_train_adv, epochs=100,
batch_size=128):

    # Calculate the number of batches per epoch
```

```
batch_per_epoch = int(X_train_adv.shape[0] / batch_size)
half_batch = int(batch_size / 2)

# Loop through each epoch
for i in range(epochs):

    # Loop through each batch
    for j in range(batch_per_epoch):

        # Generate random samples from the latent space
        X_batch = generate_latent_points(latent_dim, half_batch)

        # Generate samples from the generator
        X_fake = generator.predict(X_batch)

        # Combine real and fake samples
        X_real = X_train_adv[j * half_batch:(j + 1) * half_batch]
        X = np.concatenate([X_real, X_fake])

        # Create labels for real and fake samples
        y_real = np.ones((half_batch, 1))
        y_fake = np.zeros((half_batch, 1))
        y = np.concatenate([y_real, y_fake])

        # Train the discriminator on the real and fake samples
        discriminator.trainable = True
        d_loss = discriminator.train_on_batch(X, y)
```