GENERATION OF CYBER ATTACK DATA USING GENERATIVE TECHNIQUES

by

Nidhi N. Sakhala

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

May 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. John Springer, Chair

Department of Computer and Information Technology

Dr. Eric Dietz

Department of Computer and Information Technology

Dr. Julia Taylor Rayz

Department of Computer and Information Technology

Approved by:

Dr. Eric T. Matson

Head of the Graduate Program

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. John Springer, for all the guidance he has provided throughout this process. His guidance has played a major role in the completion of this thesis. I will also like to extend my gratitude to my Committee Members, Dr. Julia Rayz and Dr. Eric Dietz for their helpful comments.

Lastly, I am grateful to my friends for always motivating me.

TABLE OF CONTENTS

LIST OF TABLES vii
LIST OF FIGURES viii
LIST OF ABBREVIATIONS ix
GLOSSARY x
ABSTRACT xi
CHAPTER 1. INTRODUCTION 1
1.1 Significance 1
1.2 Scope 3
1.3 Research Question 3
1.4 Assumptions 3
1.5 Limitations 3
1.6 Delimitations 4
1.7 Summary 4
CHAPTER 2. REVIEW OF LITERATURE 5
2.1 Types of Cyber Attacks 5
2.2 Intrusion Detection Systems 6
2.2.1 Machine Learning based Intrusion Detection Systems 7
2.2.2 Problems with ML based IDS 8
2.3 Data Imbalance 9
2.3.1 Techniques to handle Data Imbalance 10
2.4 Generative Adversarial Networks 12
2.5 Summary 14
CHAPTER 3. METHODOLOGY 15
3.1 Dataset 15
3.1.1 Data Extraction and Cleaning 16
3.1.2 Data Transformation 17
3.2 Experiment Design 19
3.2.1 Performance Metrics 19

3.3 Models used . . . 20
3.3.1 Machine Learning models . . . 20
3.3.1.1 Decision Tree . . . 21
3.3.1.2 Support Vector Machine . . . 24
3.3.1.3 Neural Network . . . 25
3.3.2 Generative Adversarial Network models . . . 26
3.3.3 Wasserstein Generative Adversarial Network . . . 28
3.4 Hypotheses . . . 30
3.5 Summary . . . 31
CHAPTER 4. EXPERIMENT RESULTS AND CONCLUSION . . . 32
4.1 Results with IP . . . 32
4.1.1 Result Analysis . . . 34
4.2 Results without IP . . . 34
4.2.1 Result Analysis . . . 35
4.3 Evaluation of the results . . . 36
4.4 Summary . . . 38
CHAPTER 5. DISCUSSION AND FUTURE PLAN . . . 39
5.1 Discussion . . . 39
5.2 Future Work . . . 40
5.3 Summary . . . 41
REFERENCES . . . 42
APPENDIX A. CODES . . . 46
A.1 Data Cleaning . . . 46
A.2 Decision Tree . . . 47
A.3 Support Vector Machine . . . 48
A.4 Neural Network . . . 49
A.5 Wasserstein GAN . . . 50
A.6 Data generation . . . 53
A.7 Combining datasets . . . 54
A.8 Hypothesis Testing . . . 55

A.9 PCA plots 56

LIST OF TABLES

3.1 Confusion Matrix 20
3.2 Decision Tree Confusion Matrix for Original Data (with IP) 22
3.3 Decision Tree Confusion Matrix for Original Data (without IP) 22
3.4 SVM Confusion Matrix for Original Data (with IP) 25
3.5 SVM Confusion Matrix for Original Data (without IP) 25
3.6 Neural Network Confusion Matrix for Original Data (with IP) 26
3.7 Neural Network Confusion Matrix for Original Data (without IP) 26
4.1 Change in the model metrics by adding the generated samples. 32
4.2 Decision Tree Confusion Matrix for Combined Data (with IP) 32
4.3 SVM Confusion Matrix for Combined Data (with IP) 32
4.4 Neural Network Confusion Matrix for Combined Data (with IP) 33
4.5 Decision Tree Confusion Matrix for Combined Data (without IP) 34
4.6 SVM Confusion Matrix for Combined Data (with IP) 35
4.7 Neural Network Confusion Matrix for Combined Data (with IP) 35

LIST OF FIGURES

3.1 Data composition 17
3.2 Decision Tree 22
3.3 Decision Tree rules for original data (with IP) 23
3.4 Decision Tree rules for original data (without IP) 23
3.5 Pictorial Representation of GAN working 27
3.6 GAN generator and discriminator loss plot 28
3.7 WGAN critic and generator loss plot 30
4.1 Difference in the number of False Negatives (with IP) 33
4.2 Difference in the detection rate of models (with IP) 33
4.3 Difference in the number of False Negatives (without IP) 35
4.4 Difference in the detection rate of models (without IP) 36
4.5 PCA plot of original attacks and generated attacks 37
5.1 PCA plot of data distribution in the original file 39

LIST OF ABBREVIATIONS

IDS	Intrusion Detection Systems
ML	Machine Learning
GAN	Generative Adversarial Network
WGAN	Wasserstein GAN

GLOSSARY

Some terms most commonly used in this document are:

Intrusion Detection System: A system that monitors the traffic flowing through a particular network and tries to identify any malicious traffic coming to the network by any external parties.

Generative Adversarial Network: A system of neural networks that aims to generate new data which resembles the input.

Wasserstein Generative Adversarial Network: A type of GAN that provides some improvement over the basic GAN and makes it easy to train.

ABSTRACT

Author: Sakhala, Nidhi N. M.S.
Institution: Purdue University
Degree Received: May 2019
Title: Generation of Cyber Attack Data using Generative Techniques
Major Professor: John Springer

The presence of attacks in day-to-day traffic flow in connected networks is considerably less compared to genuine traffic flow. Yet, the consequences of these attacks are disastrous. It is very important to identify if the network is being attacked and block these attempts to protect the network system. Failure to block these attacks can lead to loss of confidential information and reputation and can also lead to financial loss. One of the strategies to identify these attacks is to use machine learning algorithms that learn to identify attacks by looking at previous examples. But since the number of attacks is small, it is difficult to train these machine learning algorithms. This study aims to use generative techniques to create new attack samples that can be used to train the machine learning based intrusion detection systems to identify more attacks. Two metrics are used to verify that the training has improved and a binary classifier is used to perform a two-sample test for verifying the generated attacks.

CHAPTER 1. INTRODUCTION

Machine Learning (ML) has seen success in multiple fields, ranging from image identification and classification to text analysis and speech conversion to sound recognition. Various algorithms have been developed over time to accommodate the range of available data. Researchers have then applied these algorithms for cybersecurity data to reap the benefits of machine learning and identified some problems as to why machine learning does not give good results with cybersecurity as compared to other domains. Sommer and Paxson (2010), discuss the shortcomings of ML for Intrusion Detection Systems (IDS). One of the problems discussed is that of not having enough data to train a machine learning model. Even though we have lots of traffic data, the percentage of attack data within it is very small. Machine learning algorithms learn by analyzing examples; the more examples on which we train, the better is the classification accuracy of attacks from non-attacks. But due to insufficient data, we cannot provide enough examples of attack data, and thus the models cannot be trained accurately. Generative Adversarial Networks (GANs) are a new suite of algorithms that help create new samples based on the input samples of such quality that they are indistinguishable from each other (Goodfellow et al., 2014). Using this technique, we can generate new attack data having properties resembling the input data and hence generating more samples of attack data on which to train the machine learning algorithms.

1.1 Significance

With the growth of computer networking, e-commerce applications and web-services, information security has become a global issue of serious concern. Due to the accessibility and openness of the Internet, an increasing number of users are utilizing it for almost all of their needs, including shopping, socializing, storing information, banking, etc. This leads to a large number of transactions with some involving financial transactions between servers. The secure exchange of personal information is very vital to the user as organizations and/or individuals with evil intent can listen on these transactions

and steal confidential information. This is known as a cyber attack. Uma and Padmavathi (2013) provide a listing of various types of cyber attacks by classifying them based on various criteria such as purpose, scope, terminology, and the type of network which is being attacked.

For businesses, network attack can cause major damage because it has a huge impact on the financial expenditures as well as on the customer's trust in the business. Walters (2014) discusses the cyber attacks on big corporations including restaurant chains, hotels, technology companies, government portals, and financial institutions. The attackers stole the user's login credentials, payment details, and personal information. The users whose information is stolen are susceptible to attacks like hacking, fishing, cyber bullying, or identity thefts; their payment information can be used by attackers for illegal transactions and thus encouraging more crime. This leads to the users losing their trust in a company and the company losing its customer base that in extreme cases could lead to the company going out of business and shutting down entirely. The cyber attacks have also caused huge financial losses to the companies (Cashell, Jackson, Jickling, & Webel, 2004) in terms of investigation costs, shareholder losses, attack mitigation costs, etc. All the above cases show that studying and implementing information and network security is imperative.

This thesis will help in the reduction of the cost attached to investigation of cyber attacks as well as to help prevent the networks from the dangers of these attacks. The generative approach will be used to generate more samples of the attack dataset, thereby making available enough training dataset for the classifier, and this will increase the efficiency of the machine learning tools in detecting intrusions. This tool can be used as an automated intrusion detector for any network, hence decreasing the human efforts involved in capturing traffic, identifying, and analyzing malicious traffic targeting their network. More importantly, this thesis will contribute towards reducing the number of unidentified attacks.

1.2 Scope

Imbalanced class distribution affects the performance of the IDS as the classifier tends to favor the majority class (Hido, Kashima, & Takahashi, 2009). Insufficient attack dataset makes it difficult to train the classifier, and this affects the accuracy of the result. With a balanced set, it will be possible to train the machine learning algorithms to detect attacks better. This thesis focuses on generating new attack samples in an effort to balance the dataset by using GANs. The scope of this thesis is restricted to generating the samples for Denial of Serivce (DoS) attacks only, as they are one of the most widespread attack types.

1.3 Research Question

Will the addition of new samples created using generative techniques to balance the dataset, decrease the false negatives, and improve the detection rate for a pre-trained machine learning classifier that identifies attack data from regular traffic flow?

1.4 Assumptions

The assumptions for this study include:

- The simulated attacks are representative of the actual attacks that could infiltrate the system.
- The new generated attacks will cover all the variations possible for DoS.

1.5 Limitations

The limitations for this study include:

- There is no quantitative way to judge the quality of samples created. This will pose a challenge to validate the samples generated as good attack samples.

- The dataset is of approximately 163 gigabytes. The data was sampled to a manageable size of 2 gigabytes. This sample represents data of only 1 week.

1.6 Delimitations

The delimitations for this study include:

- The attack and regular traffic flow discussed throughout the document are constrained to the traffic flows in the 1 week of sampled dataset.
- The dataset contains traffic flow information only about Denial of Serivce attacks. Hence, the generative model will create new samples similar to these attacks, and the classifier trained on this data will be able to better identify these attacks. This does not increase the classifier's ability to detect other kinds of attacks.
- The results of this thesis are only applicable for the classifier built in this thesis. These results can vary for classifiers built using different algorithms or different architectures.

1.7 Summary

This chapter provided the scope, significance, research question, assumptions, limitations and delimitations for the research project.

CHAPTER 2. REVIEW OF LITERATURE

Network attacks can cause a major damage to the business, in terms of finance, customer popularity, or loss of confidential information. This section highlights the types of attacks and the ways previously developed to identify and prevent these attacks. Furthermore, the use of generative techniques is suggested to help improve the same.

2.1 Types of Cyber Attacks

A network can be infiltrated in many ways, some with the help of human interaction, and some without the user even knowing about it. As an example, a user downloads some files from an unknown website where one of the files contains private information and sends it over the internet. In other methods, the attacker pretends to be someone whom the user trusts and then gains private information from the user. In both these cases, there is some interaction with the user. But there are other types of attack that take place without user involvement. These attacks are directly on the network, and look as normal traffic flows. They are known as network attacks and are of two types: active and passive.

Passive attacks are those in which the attacker just monitors the victim's network, by listening to the network traffic flow, or by scanning for ports that are open. This helps the attacker get information about which ports are highly used and which are idle. The main purpose of passive attacks is to collect information about the target system. They do not intend to harm the normal functioning of the systems because they want to go unnoticed while stealing information. This might also include sensitive information like passwords. Attackers use different techniques like eavesdropping and sniffing network traffic to expose a user's sensitive information. Many times, passive attacks set up the path for active attacks. Active attacks are those in which the attacker is actively interfering in the flow of data on a network. They are intended for harming the machine by introducing a virus or malware that disrupts the working of the system, consumes system memory and bandwidth by self-replication, or leaks important information to the attacker. Denial of

Service (DoS) attack is an example of an active attack. In this attack, the attacker occupies the bandwidth of the target system and keeps it busy so that it cannot service requests by other machines. This creates a huge problem for businesses because they lose out on customers that leads to a loss in revenue. For example, 48 percent of data center operators say that customer retention and gathering new customers is a significant problem after an attack (Whalen, 2018, Consequences). The same report also states that companies have lost between $10,100 and $100,000 in 2017. A variation of DoS is the Distributed Denial of Service (DDoS) attack, where multiple attackers target the same system from different IP addresses and different locations. With the growth of Internet of Things (IoT), the threat of DDoS attacks is even more widespread. In his report, Whalen (2018) mentions, "According to the data from NETSCOUT Arbor's Active Threat Level Analysis System, which covers approximately one-third of global internet traffic, there were 7.5 million DDoS attacks in 2017 alone." The same article mentions that 57% of enterprises and 45% of data center operators that were surveyed saw their internet bandwidth saturated due to DDoS attacks. Various techniques were developed to prevent networks from these attacks. The most common of them is Intrusion Detection Systems (IDS).

2.2 Intrusion Detection Systems

A wide range of security technologies such as information encryption, access control, and firewalls are used to protect network-based systems, but there are still many undetected intrusions. The most popular way to detect intrusions is by examining the network traffic and distinguishing anomalous traffic from normal traffic using an IDS. An IDS can be compared with a burglar alarm. For example, your home is protected by a lock system which is unlocked with a key. But if a burglar breaks into your house by forcing open the lock, the burglar alarm will detect that the lock has been broken and alerts the user about the intrusion. Similarly, the IDS monitors all traffic flow to the user, and if it identifies any anomalous activity, it will alert the user. Therefore, an IDS is a security system that monitors computer systems and network traffic and analyzes the traffic for possible hostile attacks (Scheidell, 2009). According to the simple model proposed by

Scheidell (2009), a network administrator sets rules for every user, which guide the traffic flow in the network. If there is any traffic outside this set of rules, it alerts the network administrator. Using this model as the base, many variants of IDS were developed. The most common types of intrusion detection systems are network-based systems and host-based systems. Network-based intrusion detection systems are deployed at points on the network itself so that they can monitor the incoming and outgoing traffic to detect intrusions. Host-based intrusion detection systems run on computers or devices in the network, which get access to the internet. There are two major methodologies of intrusion detection, namely misuse detection and anomaly detection. Misuse detection identifies intrusions based on the patterns of known intrusions. Such systems monitor the network packets and compare them against the database of known attacks. While misuse detection can be very efficient in detecting the attacks whose patterns are known to the system, it is not possible to discover all the possible attacks that could occur (Ryan, Lin, & Miikkulainen, 1998); It usually fails in detecting zero-day attacks. Anomalies are deviations from normal behavior and anomaly detection detects intrusions based on the observed abnormal activity. Detection of anomaly patterns is computationally expensive because of the overhead of maintaining and updating several system profiles since behavior patterns and system usage vary from system to system and from user to user.

2.2.1 Machine Learning based Intrusion Detection Systems

Until now, many soft computing approaches have been implemented for intrusion detection. These approaches can be classified as statistics-based, pattern-based, rule-based, state-based, and heuristic-based (Liao, Lin, Lin, & Tung, 2013). Statistics-based approaches focus on predefined mean and standard deviation as well as probabilities to identify intrusions. Pattern-based detections focus on detecting known attacks through string matching. State-based methods use finite state machine derived from network behaviors to identify attacks. All these methods require us to have knowledge about the data we wish to identify. But when it comes to zero-day attacks, this is the exact information we are missing, making these methods ineffective for the cause

we are trying to achieve. The heuristic-based approach is inspired by artificial intelligence techniques, which attempt to learn the differences between the classes of classification. Many data mining approaches have been proposed for intrusion detection. Sangkatsanee, Wattanapongsakorn, and Charnsripinyo (2011) implemented different machine learning techniques for intrusion detection like decision tree, RIPPER rule, back-propagation neural network, radial basis function neural network, Bayesian Network, and naïve Bayesian model. The experimental results of this paper show that decision trees provide higher detection rates as compared to other algorithms, and hence the authors developed a new real-time IDS based on decision trees. This new real-time IDS succeeded to efficiently detect intrusions with improved speed and lesser CPU consumption and memory. All the above techniques were evaluated on two datasets KDDCup 1999 and NSL-KDD.

In spite of this widespread use of machine-learning based algorithms for intrusion detection, there are some disadvantages that come along with these.

2.2.2 Problems with ML based IDS

Sommer and Paxson (2010) discuss why the domain of cybersecurity does not align with the requirements of machine learning. Two of the problems that they discuss are very relevant here: the high cost of errors and the lack of training data.

In machine learning vocabulary, misclassified instances are known as False Positives (FP) and False Negatives (FN). False positives are those traffic flows that the system identifies as an attack even though it is a valid activity. For alarms raised by false positives, the network operator has to investigate it only to find out it was a false alarm. This wastes investigation time and creates a false scare. This makes it difficult to trust the system to identify attacks if it also alerts for valid traffic flows. On the other hand, false negatives are indeed anomalous traffic flows, but the IDS fails to raise an alarm and allows

that traffic to pass. This is very dangerous as the system is being attacked, making the IDS inefficient. This shows us that misclassification is more highly penalized in the cyber security domain than others. Just a high accuracy number is not enough; but we need to aim for lower misclassification rates.

Another problem discussed in this domain is about the dearth of attack data for training these algorithms. ML algorithms are developed to find similarities, and anything which is not similar is an outlier. This approach does not work well with the cybersecurity data because not all traffic flow can be described in set rules. There is no fixed structure to the attacks that the machine learning algorithms can learn. Even if they do, attackers can easily change their methods and bypass the IDS. The existing datasets do not have enough variety in attack examples for the machine learning algorithms to learn them all. Out of all traffic data available, only a small percentage consists of attack data. This is known as data imbalance and makes it difficult to train machine learning algorithms. This small amount of data is considered as random noise or as outliers by the algorithms and they don't learn to classify it.

2.3 Data Imbalance

The problem of an unbalanced dataset in machine learning and data science is a scenario where class distribution is not uniform among the classes. There are usually two classes: the majority class and the minority class. In real-world applications, there are multiple datasets that are highly skewed (Hido et al., 2009). For example, the number of fraudulent actions is much smaller than legitimate transactions in credit card datasets. The number of patients is much smaller than the number of healthy people in normal medical usage data. Again, the number of attack data is smaller than normal data in network usage data. The result of this imbalance in intrusion detection can be seen in the work by Sabhnani and Serpen (2003). They implemented various classifiers including a multilayer perceptron neural network, an incremental radial basis function neural network, a gaussian classifier, a K-means classifier, a nearest neighbor classifier, a C4.5 decision tree, a fuzzy ARTMAP classifier, a leader cluster, and a hypersphere algorithm. Each classifier was

trained using the KDD training dataset and tested using the KDD testing dataset. There were a total of 1,074,991 records out of which 75.61% data is normal traffic, 23% is DDoS attack, 1.3% is Probe attack, 0.093% is R2L and only 0.004% is U2R attack. This depicts the imbalance in the available dataset. When this was used to train the above-mentioned algorithms, the probability of detection varied for each of the attacks. DDoS had approximately 97.2% probability and probe has 88.7% whereas U2R and R2L had detection probabilities 13.2% and 5.6% respectively. This shows us that if there are more training examples of a particular attack in the dataset, the probability of the algorithm learning to identify it is higher. Because of this, many attempts have been made to increase the minority class dataset. These approaches to the problem of an unbalanced dataset can be broadly classified into algorithmic level approaches and data level approaches.

2.3.1 Techniques to handle Data Imbalance

At the algorithm level, a modification is made to the algorithm to accommodate the problem of an unbalanced dataset. Sukhanov, Merentitis, Debes, Hahn, and Zoubir (2015) proposed an ensemble approach to solving the unbalanced dataset problem at varying levels by using bootstrap-based SVM aggregation methods. Bagging is used to reduce the variance, then sampling is performed only on the majority class data, and minority samples are combined with the bagged samples to form balanced sets, that are then used to train different SVMs. This method allows to discover more minority dataset but does not improve the overall accuracy of the machine learning algorithm. On the other hand for the data level approach, the dataset is re-balanced either by under-sampling, oversampling, or random sampling before making it available to the classifier as input. Japkowicz et al. (2000) examined the effect of imbalance in a dataset using data level approaches. She proposed two methods, under-sampling and resampling, and a recognition-based induction scheme. The researcher applied the concept of artificial ID data to measure and construct concept complexity. For the resampling method, two variants were explored: random and focused resampling. Random resampling consisted of resampling the minority class at random until it has as many samples as the majority class

while focused resampling consisted of resampling only those minority examples that occurred on the boundary between the minority and majority classes. But random under-sampling can result in loss of important information, and random over-sampling can lead to the problem of overfitting. SMOTE algorithm is one of the most popular oversampling methods. Chawla, Bowyer, Hall, and Kegelmeyer (2002) proposed SMOTE (Synthetic Minority Over-sampling Technique). This approach generates synthetic samples for the minority dataset. Each minority class data is taken, and a synthetic example is introduced along the line segments joining the minority class sample to its nearest neighbors, helping to augment the dataset. Douzas and Bacao (2017) presents Self Organizing Map Oversampling (SOMO), a two- dimensional representation of the input space and based on it, applies the SMOTE procedure to generate intra-cluster and inter-cluster artificial data that preserve the underlying structure. However, SMOTE has the disadvantage of generating noisy samples because of the unclear separation between the majority class and the minority class clusters (Chen & He, 2009). To overcome this, Sáez, Luengo, Stefanowski, and Herrera (2015) extends the SMOTE algorithm with an Iterative Partitioning Filter (SMOTE-IPF model). A new attribute was added to the original SMOTE model, an iterative ensemble-base noise filter that removes the noisy and borderline examples in imbalanced datasets. Also, S. Hu, Liang, Ma, and He (2009) presented the MSMOTE technique, in which the samples of the minority classes are divided into 3 types: security samples, border samples, and latent noise, and they are dealt with differently. This helps to identify more minority samples than the simple SMOTE algorithm and can help generate samples similar to the border samples as well. But even this algorithm had limitations because it could not consider the distribution of various features. SMOTE works by introducing noise to the existing distribution, thereby changing the actual distribution of the feature space.

Douzas and Bacao (2018) suggests that the data generated by SMOTE based techniques is localized to some of the minority class samples, and we need techniques which can generate data based on the data distribution. They conducted experiments with 71 datasets with varying imbalance ratios and concluded that Generative Adversarial Networks (GANs) work better in generating minority class data.

2.4 Generative Adversarial Networks

Generative Adversarial Networks (GANs) introduced a new way of data creation. Since their development, they have been used in various applications. GANs are a relatively newer class of neural networks developed by Goodfellow et al. (2014) that learn from the given data and use that knowledge to generate something similar. A standard Generative Adversarial Network consists of two neural networks, a generator, and a discriminator, that are trying to win against each other. The generator is similar to a con artist, who creates fake currency. The generator is trained to create new data from a random noisy space, but this new data should be similar to some pre-defined input. The discriminator acts like a detective who can identify fake currency from the real one. The discriminator is trained to identify the difference between the pre-defined data and the generated new data. Both the generator and the discriminator are trained simultaneously such that the generator gets better at generating fake data whereas the detective learns to identify the differences to the smallest details. The network is considered to be trained when the generator's fake data is so genuine that the detective is confused about its credibility. This data can then be considered as training data for machine learning algorithms.

GANs are used widely in image processing for generating new paintings. Zhu, Park, Isola, and Efros (2017) used this technique for image to image translation. They could change a photograph to look like a painting or generate a painting by Monet in Van Gogh's style. Ledig et al. (2017) used GANs to create a high resolution picture from a lower resolution picture. Kuefler, Morton, Wheeler, and Kochenderfer (2017) demonstrated the use of GAN for imitating the behavior of drivers on highways, which can be used in the development of self-driving cars. In cybersecurity, Kim, Bu, and Cho (2018) used the concept of generative networks for malware detection. They use a combination of autoencoders and GANs for better learning. First, the autoencoder reconstructs the given input, and in doing so learns important features about the data. Using this knowledge, they train a GAN to generate various malware examples. Using these generated examples, they can train a neural net to identify malware attacks that have

not been seen yet. W. Hu and Tan (2017) developed GANs for generating new malware. They considered existing malware detection models as black-box models, and they aimed to generate malware that could attack these black-box models. They report almost zero detection rate, meaning that the attack samples generated could not be detected by the black-box models and hence validating their quality. Yin, Zhu, Liu, Fei, and Zhang (2018) used GANs for detecting botnet attacks. They developed a neural network architecture to identify if the data is a normal activity or botnet attack. They then expanded the dataset by adding the newly generated samples, and the discriminator was used as a classifier to identify normal samples, attack samples, and fake samples. By this method, they improved the accuracy of the earlier architecture from 69.3% to 70-71%.

These above approaches show that GANs have a potential in the network security area as well. This aims to use this technique to generate synthetic network attacks. Malwares have a structured pattern within the codes in which they are hidden that the GANs can learn and recreate. However, for network attack data, there is more variability. Lin, Shi, and Xue (2018) created a model called IDSGAN where they used Wasserstein GANs to generate fake network data from NSL-KDD dataset. This thesis follows the same workflow as the research presented in the given paper and builds upon it. Lin et al. (2018) generate data for DoS and U2R&R2L and compare the performance of various machine learning models like support vector machines, naïve-Bayes classifier, perceptron, logistic regression, decision tree, random forest, and k-nearest neighbor. All of these models are considered as black box models, meaning that the IDS can be made up of any of these models. Doing this generalizes the results irrespective of the model used. The metric used by them is called detection rate that shows how good the model is in identifying attacks out of all the attacks. They suggest that the data generated is only as good as the black box model used. If the detection rate of the model reduces after addition of GAN samples, it means we have successfully created attack data. In this project, we take this idea and add to it by saying if these attack samples are then used to retrain the model, the detection rate of the model should in fact increase and give us a better intrusion detection model.

2.5 Summary

This chapter provided a review of the literature in support of the research question mentioned earlier. This section highlights the problem of unbalanced datasets and why they are a problem for training machine learning algorithms. This section also tells us about the applications of Generative Adversarial Networks (GANs) and how they have been used to combat imbalance in other datasets. But they have not yet been used to address the data imbalance for network data, hence this thesis generates attack data using this technique.

CHAPTER 3. METHODOLOGY

This chapter provides the detailed methods and algorithms used in this thesis. First, information about the dataset used is given, then the workflow is explained, later the models used are explained, and lastly, the metrics of evaluation are highlighted.

3.1 Dataset

The most common dataset used for intrusion detection is the KDD dataset, but according to the work of Sommer and Paxson (2010), they are no longer applicable. This is because it is a very old dataset, synthetically generated, and so widely studied that any results from this dataset is not considered important. Hence, the dataset used in this study is the UGR'16 dataset that is developed for the study of cyclostationarity based network intrusion detection systems. This dataset is chosen because it does not consist of just the simulated attack flows like previous cyber attack datasets, but also real traffic flow. The data collection process ranges from March 2016 to August 2016, giving a wide range of traffic with which to work. They have filtered this traffic through existing IDS and identified attacks on the network in this time duration. Additionally, they have inserted attacks into the network at pre-decided times during the last 6 weeks of the data collection process. This helps with the labeling of the traffic flows. Since the attacks are simulated for only 6 weeks out of 6 months, the imbalance ratio is significant. This imbalance makes this database a suitable candidate for this study. The dataset is provided as CSV files with the following fields:

- Timestamp of end of flow
- Duration of the flow
- Source IP address
- Destination IP address
- Source port

- Destination port
- Protocol
- Flags
- Forwarding Status
- Type of Service
- Number of packets exchanged in the flow
- Number of bytes exchanged
- label of the flow - background flow / type of attack

Out of all the attacks conducted for the collection of this dataset, this study focuses only on Denial of Service (DoS) attacks due to the widespread of DoS and its variants (Whalen, 2018, Threat Landscape).

3.1.1 Data Extraction and Cleaning

The attacks were induced during 6 weeks of data collection, from July, week 5 to August, week 5 of 2016, out of which DoS attacks were implemented in the first 3 weeks. Out of these 3 weeks of DoS data, data only from the 1st week is considered for the further process. This was done solely due to the sheer size of the data. The complete data of July, week 5 is of 52G, which consists of 530 million entries of normal traffic flow and 7.5 million attack data. This 7.5 million includes approximately 3 million DoS, 2 million ssh-scan, and 60,000 botnet attacks along with some filtered traffic due to blacklisted IPs and spams (approximately 2 million) . This huge dataset was then cleaned by using a regular expression, which filtered out inconsistencies and missing data. It also filtered out other attack types, leaving us with only DoS traffic and normal traffic.

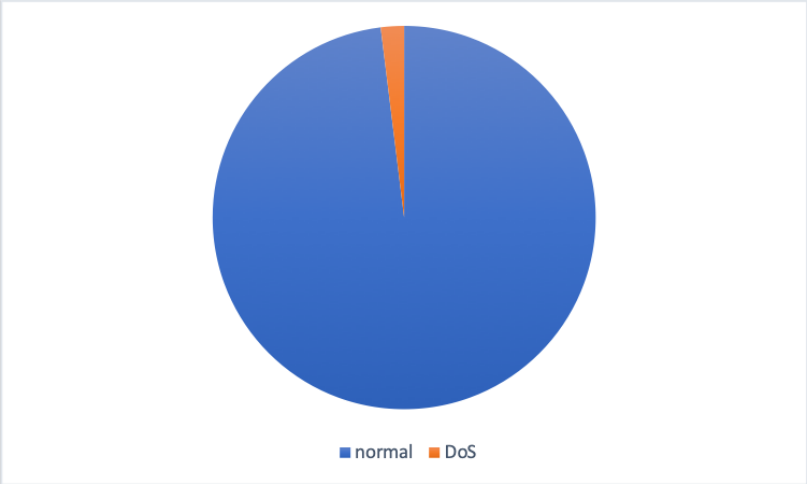

Figure 3.1. Data composition

This file was too large with which to be able to work, so the data was scaled down to a manageable size. This was done by randomly sampling a tenth of the attack data and 50 times the normal data. The final dataset used in all the methods ahead has a 1:50 imbalance ratio as seen in Figure 3.1. This was done to mimic the imbalance in the original data.

3.1.2 Data Transformation

The attacks were executed in batches for creating the database. For a certain number of days, a particular attack was induced in the network at a certain time. To make sure that the black box machine learning models are not learning this schedule based pattern to identify attacks, the time stamp information was not completely used. Timestamp was reported in the format “2016-07-27 13:43:21.” The year, month and day information was filtered out, and only the hour, month, and seconds information was preserved for the next tasks.

Next, the IP addresses which were stored as strings in the CSV file were converted to IPs using the IP address library in Python. For further processing, they were converted to their integer representations by using type casting. While collecting the traffic flows for creating the dataset, the authors anonymized the IP addresses by changing the last octet. Still, the IPs are within a fixed range and for a fixed number of machines, so it is possible that the black box machine learning models learn to identify these IPs in a particular pattern. To see the effect of IPs, the analysis has been done in two ways, once considering the IPs and once without the IPs. The performance in both cases is measured.

The flags are stored as a string of 6 characters with every character signifying the work of that traffic flow. For example, A means awaiting ACK, S means initial SYN. Each of these 6 characters was split into 6 columns, and each of those was converted to their ASCII values. For classification in machine learning, generally labels 0 and 1 are used with 1 being the desired class of classification. Hence, the attack samples were labeled as 1 and normal samples as 0. All the other columns were kept as they are.

As a final step towards preparing the data for the models, a MinMaxScaler was used to normalize the data to a smaller range (usually 0 to 1). As every column in the data has a different range of data, computations get too space and time consuming for the machine learning models. To reduce this burden, the data is represented in a standard scale by changing the values from the original scale to 0 to 1 scale. This is done by the following:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

This scaled data is then used for classification by neural network and support vector machine. Decision tree is a scale-invariant algorithm, so unscaled data was provided for it.

3.2 Experiment Design

A black box model is an algorithm, whose functioning is not known to the outer system. In this case, the IDS can consist of any algorithm, thus being a black box to the user. This study aims to generate data irrespective of the model used. After the data is clean and prepared, the following steps are followed for every black box model:

1. Split the cleaned dataset into training and testing sets.
2. Train the black box model on the training set.
3. Note the metrics on the testing set.
4. Filter out only the attack information of the dataset and use it to train the WGAN.
5. Collect the data generated by the WGAN that is identified as an attack by the black box model.
6. Retrain the black box model by adding the generated data with the previous training data.
7. Note the metrics on the same test data as before and compare.

3.2.1 Performance Metrics

This study compares the performance of the black box model before and after adding the new samples. To compare the performance, 2 metrics are used: False Negatives (FN) and Detection Rate (DR). To measure both of them, a classification matrix is used. This is a tabular representation of the actual number of samples in the training set and the number of samples classified by the model in the 2 classes as seen in Table 3.1.

In the confusion matrix, TP and TN stand for True Positives and True Negatives respectively that are the correctly classified samples. FN and FP stand for False Negatives and False Positives, respectively and are misclassified instances.

Table 3.1. Confusion Matrix

	Predicted Background	Predicted Attack
Actual Background	TN	FP
Actual Attack	FN	TP

False negatives are the attacks that are classified as normal traffic by an IDS. These traffic flows are not blocked and they continue down their path in the network, causing the harm they were intending. These flows can be fatal to the network and are best avoided. Hence, we measure the number of false negatives before and after adding the generated samples.

The second metric used is known as Detection Rate, which is a measure of how good the model is at detecting the class of interest, in this case, the attacks. The detection rate is calculated as a ratio of the number of attacks detected by the total number of attacks present(Lin et al., 2018). Lower detection rates indicate that the model is failing at identifying attacks.

With these definitions in mind, a good model is one with the lower number of false negatives and a higher detection rate.

3.3 Models used

The final dataset was then trained on 3 different models that take as input the unbalanced dataset and return the classified results. The metrics mentioned above are measured. The attack part of this dataset is then given to the WGAN to generate more attack examples, which are then combined with the original dataset. This combined dataset is then retrained on the 3 machine learning models, and the metrics are measured again. The architectures of all these models are explained in the next section.

3.3.1 Machine Learning models

Any machine learning algorithm is completed in 5 steps:

- Normalization
- Train-test split
- Fitting the model on the training data
- Use the fitted model for prediction on testing data
- See the classification report

These steps are followed in this study as well. Min-Max normalization has been used for all the algorithms except decision trees. The training testing split ratio used is 80:20. A classification matrix is printed for every algorithm and detection rate is calculated.

3.3.1.1 Decision Tree

A decision tree classifier uses rules to classify the samples. Based on the samples given, it identifies all the values a particular attribute can take and learns the best identification of class from the attribute value. The algorithm learns which is the best attribute on which to split the data by calculating 2 values known as entropy and information gain. Entropy is a measure of the randomness in the subset of the data, and information gain is a measure of the degree to which the split helps to classify the samples of the dataset. Hence the best attribute on which to split will be the one with the low entropy and the high information gain.

Figure 3.2 is an example of how a decision tree algorithm works. The algorithm found X to be the best in splitting the data into the 2 classes, based on the condition. Depending on which branch the next sample follows, the next node will be the next criteria of the decision, so on and on. Continuing this process, a decision tree forms rules like "IF $X \leq 10$, then next action," etc. Then for the testing set, each sample just follows the formed rules and its class is decided.

In this study, the depth of the tree was restricted to 5 to make sure only the most relevant features are used for classification. Tables 3.2 and 3.3 show the confusion matrices for the original data, once when trained with the IP attributes, and once without. Figure 3.3 highlights the rules that the classifier has learned after training on the original

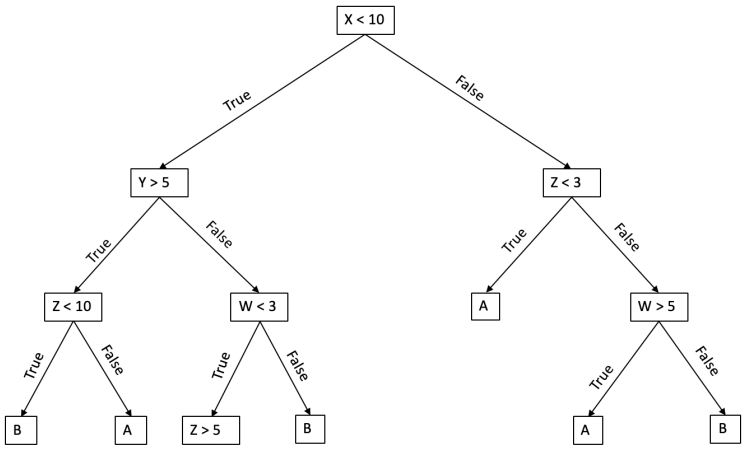

Figure 3.2. Decision Tree

data. The root node shows that source IP is the most important feature for this classification. But in real world, one would not know all the IPs that can attack a system. Hence, the classifier should not learn to identify attacks based on the IPs. This led us to train the classifier without the IP information as well. Figure 3.4 shows us the new rules of classification.

Table 3.2. Decision Tree Confusion Matrix for Original Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3134372	1
Actual Attack	0	63080

Table 3.3. Decision Tree Confusion Matrix for Original Data (without IP)

	Predicted Background	Predicted Attack
Actual Background	3126710	7663
Actual Attack	20	63060

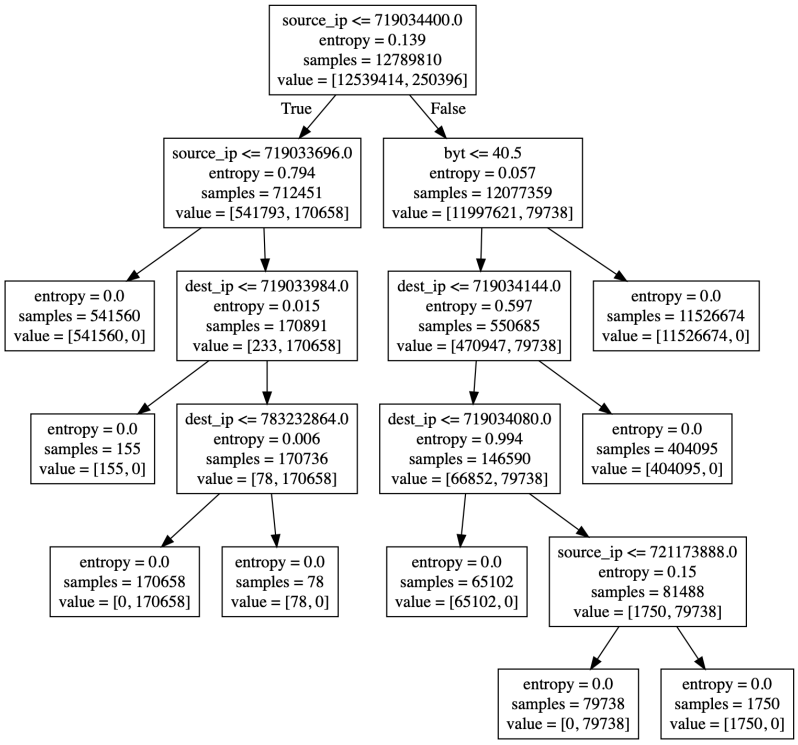

Figure 3.3. Decision Tree rules for original data (with IP)

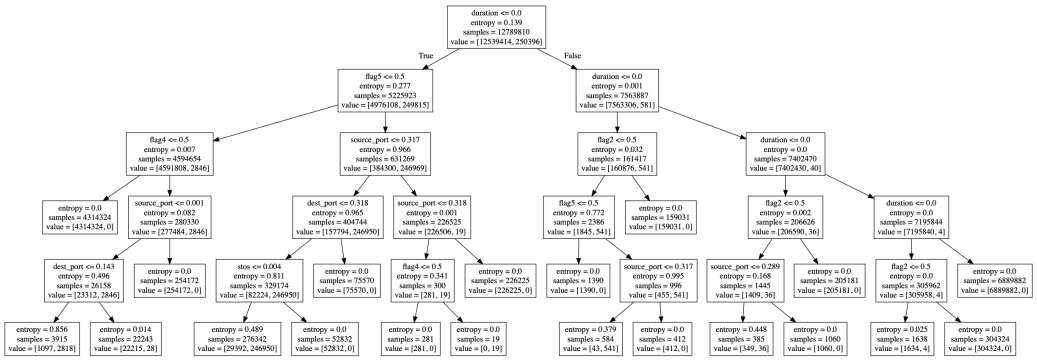

Figure 3.4. Decision Tree rules for original data (without IP)

3.3.1.2 Support Vector Machine

A Support Vector Machine is a model that separates the classes by finding a boundary that can separate the two classes geometrically. In the perfect case, all samples on one side of said boundary are of one class and all samples on the other side, are from a different class. When the distribution of samples in the dataset is such they can be separated by a linear equation, a linear kernel can be used for classification. For example, if all samples of one class satisfy the constraint $ax + by + c > 0$ and all samples of the other class satisfy the constraint $ax + by + c < 0$, then the plane $ax + by + c$ is a valid candidate for being a separating hyperplane. The final choice is made by calculating that plane that has the maximum margin between the plane and the data points.

However, a linear hyperplane can only be used when the data set is linearly separable. When the distribution of data is such that there cannot be a straight line separating the two classes, a linear kernel cannot give correct results, since there will always be misclassified samples. In these cases, we can sometimes use non-linear kernels. They transform data from lower dimensions to higher dimensions while separating the samples in a way that their representation in the higher dimension can have a hyperplane that will cut through the dataset properly. One such non-linear kernel is a Gaussian kernel, also known as the Radial Basis Function (RBF). This study uses the RBF kernel for classification. By design, a hard margin SVM algorithm (C=1.0) keeps finding the best plane to separate the classes iteratively. If the samples of the dataset are very close to each other and it's impossible to classify them perfectly, the algorithm does not complete. To avoid this, a soft margin classifier is implemented (C=0.8) to allow some room for misclassifications but still being able to perform well as a classifier. Also, a hard limit is set on the number of iterations (10,000) after which the classification at that stage is reported. Another important point to note is that SVM does not perform well for datasets with overlapping samples of both classes because that makes it difficult to find even one way to separate the two distributions.

Tables 3.4 and 3.5 show the confusion matrices when SVM is trained on original data, once with the IP attributes, and once without. High misclassification is seen, especially when IP information is not considered for classification.

Table 3.4. SVM Confusion Matrix for Original Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3133195	1178
Actual Attack	813	62267

Table 3.5. SVM Confusion Matrix for Original Data (without IP)

	Predicted Background	Predicted Attack
Actual Background	3095491	38882
Actual Attack	7526	55554

3.3.1.3 Neural Network

A Neural Network (NN) is a model inspired by the human brain where information is communicated using units called neurons. In a neural network, the neurons are arranged in layers with a minimum of 2 layers for input and output, and there can be 1 or more hidden layers. The model learns by passing information through the network in batches, comparing its output with the expected output and trying to reduce the offset between the two values by a process called backpropagation. For training this model, multiple hyper-parameters have to be explored; this study varied the number of layers, the number of nodes within the layers, the type of activation function, the learning rate, and the number of epochs to build a high accuracy classifier.

Various architectures and hyperparameters were tested before finalizing on this architecture. For all the analysis here, a 3 layer neural network was used: 19 nodes in the input layer (17 for non-IP data), 64 nodes in the hidden layer, and 1 for the output layer. ReLU activation function was used for the input and hidden layer while a sigmoid was used for the output layer. These activation functions were chosen because the output of the model was preferred to be a number between 0 and 1 since the labels in the dataset were 0 and 1. This architecture was chosen because it was the simplest one with good results; architectures with more number of layers gave similar results. The model was trained for 20 epochs. Tables 3.6 and 3.7 show the confusion matrices when this neural network was trained on the original data, once considering the IP attributes and once without them. Once again, better results are seen with the IP information considered for classification.

Table 3.6. Neural Network Confusion Matrix for Original Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3133958	415
Actual Attack	70	63010

Table 3.7. Neural Network Confusion Matrix for Original Data (without IP)

	Predicted Background	Predicted Attack
Actual Background	3132218	2155
Actual Attack	610	62470

3.3.2 Generative Adversarial Network models

Neural networks have been in use for a variety of applications. They have been in use for understanding context in natural language, in identifying images, and in detecting and responding to surrounding information, and we have seen these applications in our daily activities. The Generative Adversarial Networks (GANs) were invented with the aim of using neural networks to learn this information and construct something new (Goodfellow et al., 2014). For examples, construct new images which are similar to existing images, like new dog images, or creating new music. This technique has opened up a whole new avenue for researchers to explore.

A standard GAN consists of two neural networks, a generator and a discriminator, that are trying to win against each other. The generator is similar to a con artist who creates fake currency. The generator is trained to create new data from a random noisy space, but this new data should be similar to some pre-defined input. The discriminator acts like a detective who can identify fake currency from the real one. The discriminator is trained to identify the difference between the pre-defined data and the generated new data.

Both the generator and the discriminator are trained simultaneously such that the generator gets better at generating fake data whereas the detective learns to identify the differences to the smallest details. The network is considered to be trained when the generator's fake data is so genuine that the detective is confused about its credibility. This

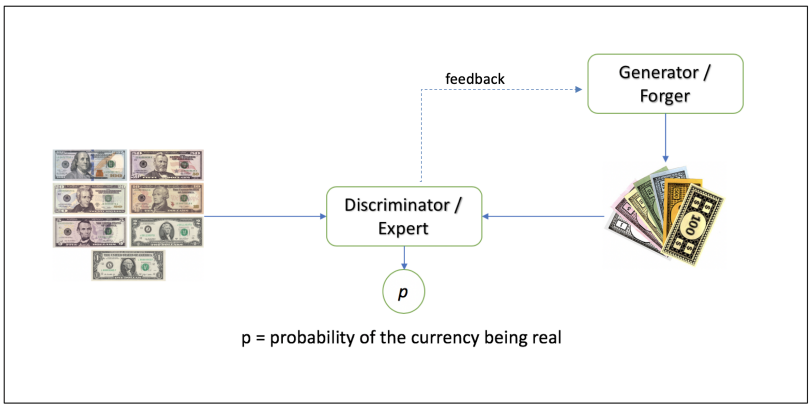

Figure 3.5. Pictorial Representation of GAN working

is done by considering the generator and discriminator as players of a min-max game with value function as follows:

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z}(z)[log(1-D(G(z)))]$$

In the above equation, G stands for the Generator network, D for Discriminator network, x is actual data of which we wish to generate more, and z is the noise vector. To achieve this value function, the binary cross entropy loss (BCE loss) of a neural network is used. It is represented as -

$$\sum l_i, l_i = -w_i(y_i log(v_i) + (1-y_i) log(1-v_i))$$

In this, w_i is the weight matrix of the neural network, y represents the labels of the data, and v are the inputs. Both the generator and the discriminator contribute to this loss function. If we give label 1 for real data and label 0 for generated data to the discriminator, it learns in a way that it wants to generate data with label 1. While training, the discriminator loss comes to a constant of 0.5, which means that it is no longer able to identify the data as real or fake. The generated data at this stage could be considered as good data.

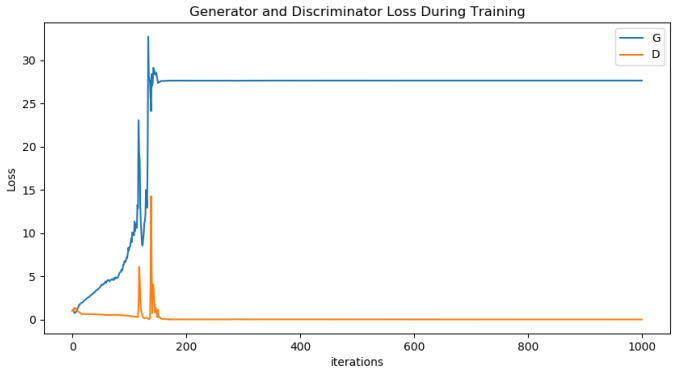

Figure 3.6. GAN generator and discriminator loss plot

However, training GANs is extremely unstable. It is very common to see one model overpowering the other. In either case, the whole model learns nothing, and it is very difficult to diagnose this situation. As can be seen from Figure 3.6, the discriminator is easily identifying everything as fake, completely overpowering the generator. Hence, the generator learns nothing, and its loss value remains constant or increases instead of converging. A solution to this is suggested in the form of Wasserstein GAN (Arjovsky, Chintala, & Bottou, 2017).

3.3.3 Wasserstein Generative Adversarial Network

The basic GANs try to learn the probability distribution of data from the given set of data points. This is based on the distance metric Kullback-Leibler (KL) divergence.

Consider the real distribution (P_r) is what we want to learn and parametric distribution (p_θ) is like a man in disguise, trying to adjust its parameters so it looks like the real distribution. KL divergence attempts to reduce the closeness between the real and the parametric distributions. In other words, it means the closer the generated data is to the real data, the lower would be the KL divergence.

$$KL(\mathbb{P}_r, \mathbb{P}_g) = \int log(\frac{P_r(x)}{P_g(x)})P_r(x)d\mu(x)$$

The problem with KL divergence arises due to the log terms in the equation. If the distribution of generated data is far off from real data, the log term of the real samples become ∞, making log 0, which is not defined. To avoid this problem, an advised solution is to add noise to the data. However, adding noise for multiple features is computationally heavy and pointless. Also, adding noise during training produces noisy samples. So, GANs leverage another property to deal with this issue, i.e, there is a low-level representation for every high-level representation.

Hence Arjovsky et al. (2017) suggest to learn the parameters such that generated data can belong to the distribution of data in higher dimensions. In other words, learn the mapping in order to bring the distributions close in original space. This is based on the property of continuity distribution that if θ_1 and θ_2 are close, the distributions of θ_1 and θ_2 should not vary drastically. To measure the distance in the 2 distributions, another metric called Earth Movers distance is used.

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

In the above equation, also known as Wasserstein Distance, $\gamma(x,y)$ indicates how much mass must be transported from x to y in order to transform the distributions $\mathbb{P}_r$ into $\mathbb{P}_g$. The discriminator in the GAN network is replaced with a critic who aims to minimize this Wasserstein distance. In training the models of WGANs, the critic needs to be trained more than the generator. This is done because it is important for the critic to be good in its job before generating data. Also, for WGANs, the weights of the network are clipped. We constrain the weights to be in the range -0.01 to 0.01. The general architecture is adapted from Lin et al. (2018), various generator and critic architectures were tried before finalizing this. This architecture was chosen because it trained the critic to output distance as 0 (seen in Figure 3.7) as this means that the critic is successfully trained. Other architectures had critic outputs oscillating around 0.

The hyperparameters used for this training are:

- Generator Architecture - Linear(100,128,128,19)
- Critic Architecture - Linear(19,128,128,1)

- Actiavtion Function - ReLU
- Optimization Function - RMSProp
- Epochs - 500000 for generator
- Learning rate - 0.00005

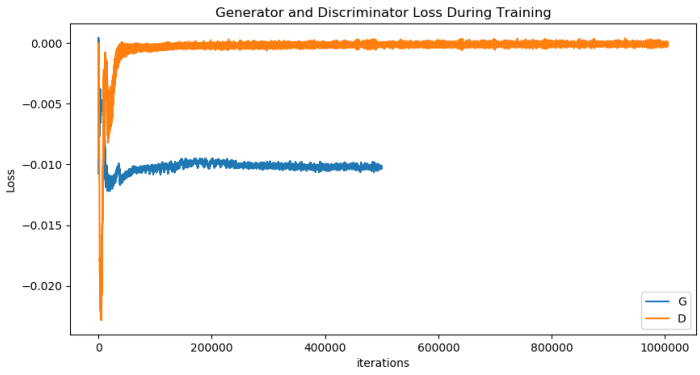

Figure 3.7. WGAN critic and generator loss plot

3.4 Hypotheses

By using the Wasserstein GAN, this study aims to generate more attack data that helps to balance the dataset for better attack identification. GANs generate data that are in the same distribution as the input data. Hence, we use a two sample test to check whether the generated samples come from the same distribution as the original samples.

Lopez-Paz and Oquab (2016) suggest the use of a binary classifier for conducting a two sample test for GANs. They consider the original dataset space to be P and the generated dataset to be Q. Based on this, the null and alternative hypotheses are as follows:

H_0: The generated samples come from the same distribution as the original attack samples.

$$P = Q$$

H_α: The generated samples do not come from the same distribution as the original attack samples.

$$P \neq Q$$

To test this hypothesis, a neural network is used as a classifier with 20 hidden layer neurons (with ReLU activation) and 1 output layer neuron (with sigmoid activation). This network is trained for 100 epochs. A sample set of 10,000 is randomly chosen from the original attack data and the generated attack data, respectively. The original attack samples are labeled 0, and the generated ones are labeled as 1. The 2 sample sets are combined, shuffled, and then split into training and testing sets. The classifier is then trained on the training set, and accuracy is measured for the testing set.

If the samples come from the same distribution, the classifier should ideally not be able to classify them, thereby resulting in a low test accuracy. If the testing accuracy is represented by t, according to the work done by the authors (Lopez-Paz & Oquab, 2016), the null and alternative hypotheses can be rewritten as the following:

$$H_0 : t = 0.5$$

$$H_\alpha : t = 0.5 + \delta$$

3.5 Summary

This chapter described the experiment design followed in this study. It also explained the models used in this study, the metrics measured, and the hypothesis that will be tested.

CHAPTER 4. EXPERIMENT RESULTS AND CONCLUSION

This chapter provides the results of the experiments. The first section details the result when 19 features were used for the dataset while the second section details the results for 17 features (without IPs). Table 4.1 summarizes these results and shows the percentage difference in the metrics, number of false negatives, and detection rates with values measured on the original dataset and on the balanced dataset.

Table 4.1. Change in the model metrics by adding the generated samples.

Model	With IP		Without IP	
Black-box Model	△FN (%)	△DR (%)	△FN (%)	△DR (%)
Decision Tree	0	0	-45	0.013
SVM	-19.31	0.252	145.26	-19.68
Neural Network	-72.86	0.081	-98.03	0.96

4.1 Results with IP

First, only the year, month and day information were removed, and all models were trained on the remaining 19 features. Figure 4.1 shows the difference in the number of False Negatives, and Figure 4.2 shows the difference in Detection Rate.

Table 4.2. Decision Tree Confusion Matrix for Combined Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3133911	462
Actual Attack	0	63080

Table 4.3. SVM Confusion Matrix for Combined Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3133123	1250
Actual Attack	656	62424

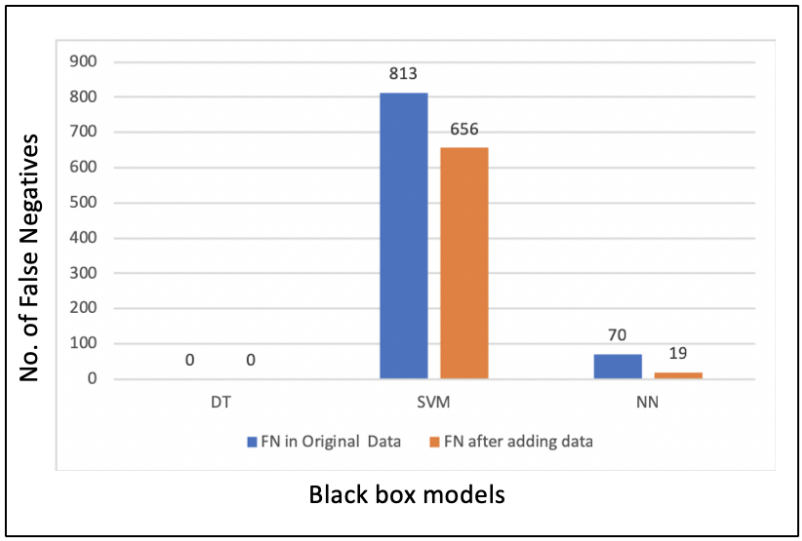

Figure 4.1. Difference in the number of False Negatives (with IP)

Table 4.4. Neural Network Confusion Matrix for Combined Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3133379	994
Actual Attack	19	63061

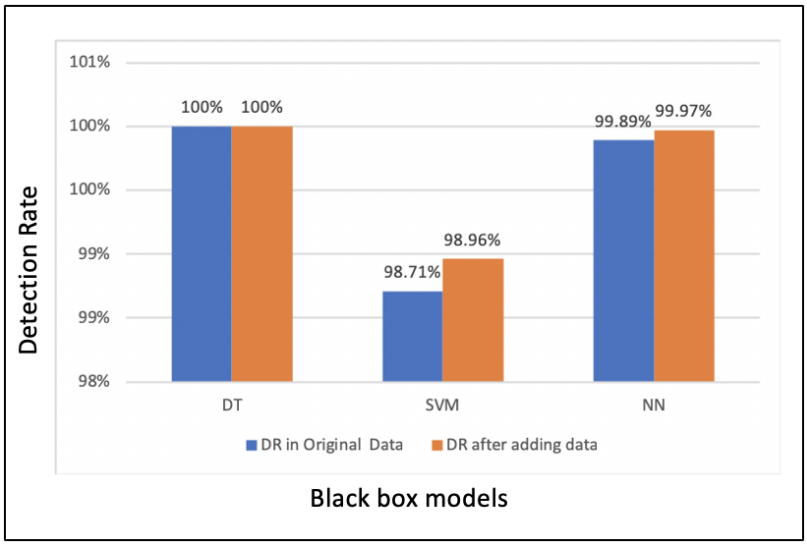

Figure 4.2. Difference in the detection rate of models (with IP)

4.1.1 Result Analysis

For Support Vector Machines (SVMs), there is a reduction of 19.3% in the number of false negatives, while there is a reduction of 72.86% for the neural network. In both these models, there was a notable increase in the detection rate of the attacks (0.252% and 0.081%, respectively), but the percentage of increase is very small. This small increase could be because of a pretty high detection rate($\geq$ 90%) with the original data itself. In spite of reduction in the number of false negatives, a large increase is also seen in the number of false positives. This is not good as this misclassification increases time spent by experts in analysing the traffic, but it is not as harmful as false negatives.

Decision trees have been the most successful in detecting the attacks with absolutely no misclassifications in the original data itself. The test with generated data was still conducted to see if there is an increase in the number of false negatives due to potential over-fitting. But even after the addition of new samples, the model still had a 100% attack detection rate.

4.2 Results without IP

The same set of experiments was also done for the dataset by removing the IP information. This resulted in a different set of observations. Figure 4.3 shows the difference in the number of False Negatives, and Figure 4.4 shows the difference in Detection Rate.

Table 4.5. Decision Tree Confusion Matrix for Combined Data (without IP)

	Predicted Background	Predicted Attack
Actual Background	3124166	10207
Actual Attack	11	63069

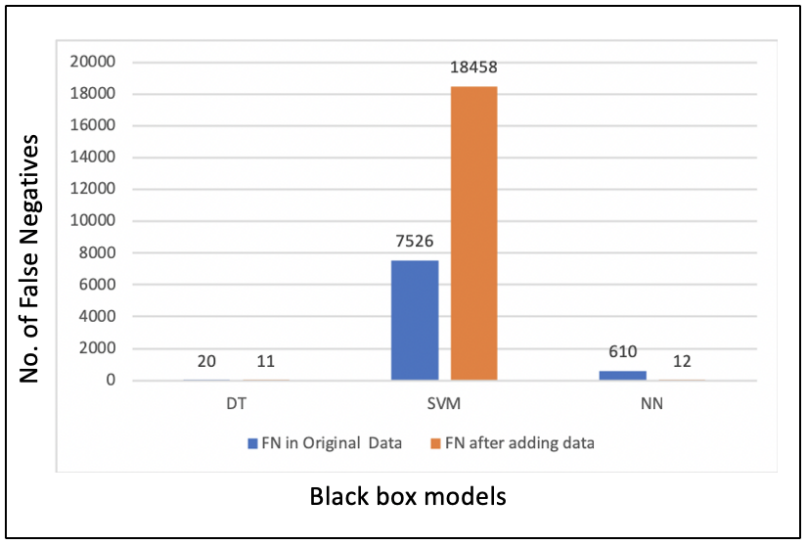

Figure 4.3. Difference in the number of False Negatives (without IP)

Table 4.6. SVM Confusion Matrix for Combined Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	2511312	623061
Actual Attack	18458	44622

Table 4.7. Neural Network Confusion Matrix for Combined Data (with IP)

	Predicted Background	Predicted Attack
Actual Background	3073080	61293
Actual Attack	12	63068

4.2.1 Result Analysis

The first thing to observe here is that it is difficult for Neural Network and Support Vector Machine to identify attacks in this scenario. This means that they were learning IP-based information to identify attacks, which could mislead the learning. The decision tree also suffered, but the impact was not as much. Its detection rate with the unbalanced data is still almost 100%.

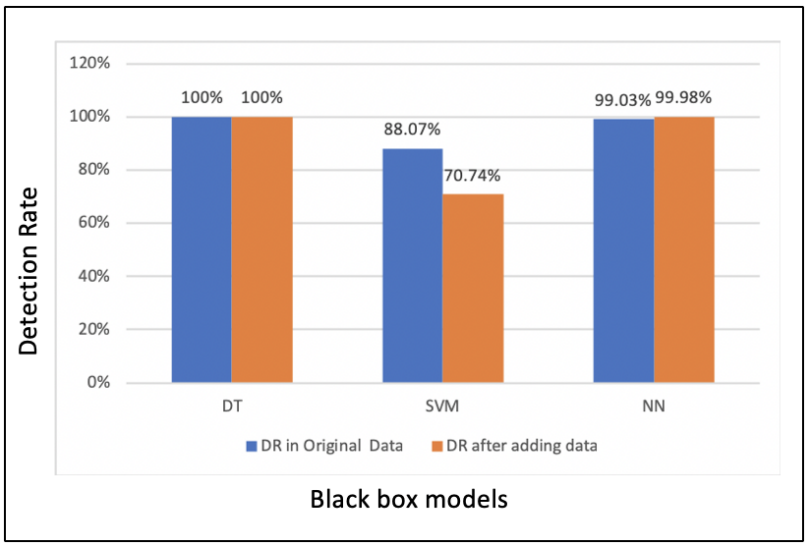

Figure 4.4. Difference in the detection rate of models (without IP)

For Decision Tree and Neural Network, the earlier trend still follows. There is a reduction in the number of False Negatives as well as a small increase in the Detection Rate. However, the Support Vector Machine's performance worsened greatly. This could have happened because the new data samples could have overlapped with the normal samples, making it difficult for the algorithm to find a proper separating boundary.

4.3 Evaluation of the results

The above numbers tell us that there certainly is a reduction in the number of False Negatives. To check if the generated data is similar to the attack data in the original dataset, a two sample test using a binary classifier was used.

$$H_0 : t = 0.5$$

$$H_\alpha : t = 0.5 + \delta$$

$$\hat{t} = 0.9165$$

As $\hat{t} > 0.5$, we reject the null.

Thus we conclude that the attacks generated by the Wasserstein GANs are **not similar** to the original attacks as expected. To understand these results, a Principal Component Analysis (PCA) plot was generated for the attack datasets. The original attack data was combined with the generated attack data. The PCA algorithm identified the top two eigen vectors to classify these two attack sets. Using these vectors as axes, Figure 4.5 was generated. Here, the green dots represents the attacks created by GANs, and the yellow dots are the original attacks. There are more green points than yellow because 12 million attacks were generated from the original 300 thousand attacks. Figure 4.5 shows

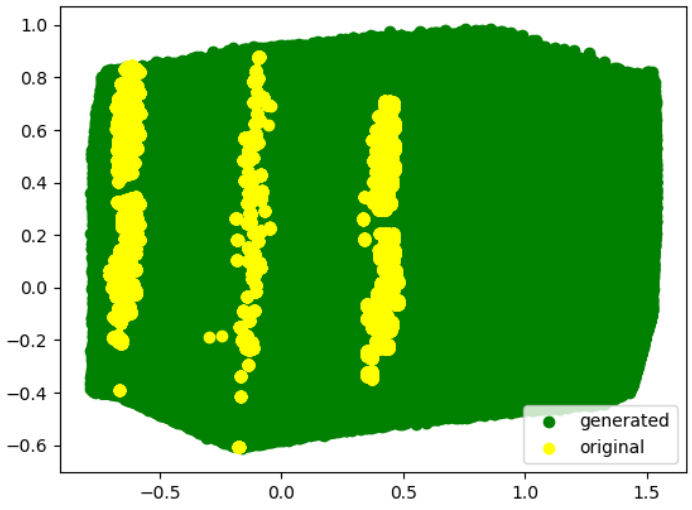

Figure 4.5. PCA plot of original attacks and generated attacks

the distribution of the two attack datasets. The yellow dots are distinguishable from the green ones, implying that they can be classified.

4.4 Summary

In this chapter,the results of all experiments are compared. Figures 4.1, 4.2, 4.3, and 4.4 show the numerical differences in the metric values for all the tests, and Table 4.1 summarizes these changes. Also, this section helps reach the conclusion that our hypothesis is not satisfied and the generated attacks are not similar to the original attacks.

CHAPTER 5. DISCUSSION AND FUTURE PLAN

5.1 Discussion

The earlier chapter concluded that the attacks generated were not from the same distribution as the attacks in the original distribution. Figure 4.5 shows that the attacks generated are in the area surrounding the original attacks, but they are still easily distinguishable. There could be a few reasons why this could be happening.

- The generated data was only checked to be similar to the attack data, without given any information about the normal traffic flow. The plot in Figure 5.1 shows the distribution of attack data and normal data. If this information is provided to the GAN, it might perform differently.

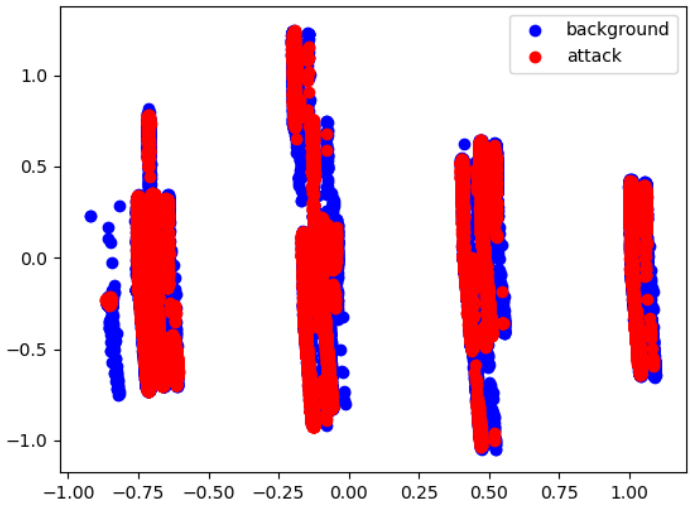

Figure 5.1. PCA plot of data distribution in the original file

- To ease computation, initially the data was sampled in 1:50 ratio of attack to normal traffic. This could have changed the actual distribution of the data, resulting in learning the wrong distribution. Different data structures and more hardware can be used to accommodate the complete dataset as is.

- There is no fixed metric to measure the quality of GAN generated data. Most of the research in this field is done on images, giving us metrics like Inception Score and Fréchet Inception Distance. The problem with these metrics is that they work well with images trained on ImageNet, but beyond that, these methods fail to accurately test other images (Barratt & Sharma, 2018). Thus looking into these methods for non-image data is not reliable. The method used in this study is one of the newer attempts to generalize GAN testing methodologies. Even in the original paper by Lopez-Paz and Oquab (2016), they tested their method against image dataset and it failed. Only when they changed their binary classifier to a ResNet (image identification model) did they get desired results. All these works focus on the quality assessment in human terms, by trying to assess the quality of the generated image as a human critic would judge it. But in the case of attack data, there is no human perception to it. So it does not make sense to use those methods to judge this data. Borji (2019) studied 29 different GAN evaluation metrics, which can be further studied and a better technique can be found for this application.

5.2 Future Work

- Other GAN models such as Conditional GANs can be used to generate attack data with the context of normal data.

- Parallel and scalable ML packages, such as dask, can be used to process the whole dataset and not just samples.

- Different hypothesis testing methodology that is more suited for this dataset can be applied. The comparative study by Barratt and Sharma (2018) can help to choose the appropriate method.
- In the UGR'16 dataset, DoS attack is majorly collected. Because of the comparatively higher volume of data, its original detection rate is high. The same study can be done with other attack types, which have a lower detection rate to begin with.
- The sampled dataset in this study had a ratio of 1:50 for attack data and normal data. Even with this imbalance, the classification accuracy was high. So, generating data to make this ratio 1:1 isn't really necessary. Different numbers of data can be generated and the performance of the IDS can be compared for these different batches. This can help identify the trend in classification accuracy and number of generated samples.

5.3 Summary

This chapter explains how the research conducted in this thesis can be expanded further. The future work section highlights the research fronts in this topic.

REFERENCES

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In *International conference on machine learning* (pp. 214–223).

Barratt, S., & Sharma, R. (2018). A note on the inception score. *arXiv preprint arXiv:1801.01973*.

Borji, A. (2019). Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, *179*, 41–65.

Cashell, B., Jackson, W. D., Jickling, M., & Webel, B. (2004). The economic impact of cyber-attacks. *Congressional Research Service Documents, CRS RL32331 (Washington DC)*.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, S., & He, H. (2009). Sera: selectively recursive approach towards nonstationary imbalanced stream data mining. In *Neural networks, 2009. ijcnn 2009. international joint conference on* (pp. 522–529).

Douzas, G., & Bacao, F. (2017). Self-organizing map oversampling (somo) for imbalanced data set learning. *Expert systems with Applications*, *82*, 40–52.

Douzas, G., & Bacao, F. (2018). Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with applications*, *91*, 464–471.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Hido, S., Kashima, H., & Takahashi, Y. (2009). Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *2*(5-6), 412–426.

Hu, S., Liang, Y., Ma, L., & He, Y. (2009). Msmote: improving classification performance when training data is imbalanced. In *Computer science and engineering, 2009. wcse'09. second international workshop on* (Vol. 2, pp. 13–17).

Hu, W., & Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*.

Japkowicz, N., et al. (2000). Learning from imbalanced data sets: a comparison of various strategies. In *Aaai workshop on learning from imbalanced data sets* (Vol. 68, pp. 10–15).

Kim, J.-Y., Bu, S.-J., & Cho, S.-B. (2018). Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences*, *460*, 83–102.

Kuefler, A., Morton, J., Wheeler, T., & Kochenderfer, M. (2017). Imitating driver behavior with generative adversarial networks. In *Intelligent vehicles symposium (iv), 2017 ieee* (pp. 204–211).

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... others (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Cvpr* (Vol. 2, p. 4).

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16–24.

Lin, Z., Shi, Y., & Xue, Z. (2018). Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077*.

Lopez-Paz, D., & Oquab, M. (2016). Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*.

Ryan, J., Lin, M.-J., & Miikkulainen, R. (1998). Intrusion detection with neural networks. In *Advances in neural information processing systems* (pp. 943–949).

Sabhnani, M., & Serpen, G. (2003). Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context. In *Mlmta* (pp. 209–215).

Sáez, J. A., Luengo, J., Stefanowski, J., & Herrera, F. (2015). Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, *291*, 184–203.

Sangkatsanee, P., Wattanapongsakorn, N., & Charnsripinyo, C. (2011). Practical real-time intrusion detection using machine learning approaches. *Computer Communications*, *34*(18), 2227–2235.

Scheidell, M. (2009, October 13). *Intrusion detection system.* Google Patents. (US Patent 7,603,711)

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Security and privacy (sp), 2010 ieee symposium on* (pp. 305–316).

Sukhanov, S., Merentitis, A., Debes, C., Hahn, J., & Zoubir, A. M. (2015). Bootstrap-based svm aggregation for class imbalance problems. In *Signal processing conference (eusipco), 2015 23rd european* (pp. 165–169).

Uma, M., & Padmavathi, G. (2013). A survey on various cyber attacks and their classification. *IJ Network Security*, *15*(5), 390–396.

Walters, R. (2014). Cyber attacks on us companies in 2014. *The Heritage Foundation*, *4289*, 1–5.

Whalen, K. (2018, Jan). *Frequency and complexity of ddos attacks is rising.* Retrieved from `https://www.netscout.com/news/press-release/complexity-ddos-attacks`

Yin, C., Zhu, Y., Liu, S., Fei, J., & Zhang, H. (2018). An enhancing framework for botnet detection using generative adversarial networks. In *2018 international conference on artificial intelligence and big data (icaibd)* (pp. 228–234).

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.

APPENDIX A. CODES

All the codes are mentioned in this appendix. First the codes for the machine learning models are written, then the code for the WGAN.

A.1 Data Cleaning

This shows the script used to clean the dataset as mentioned in 3.1.1.

```
import argparse
import platform, logging, os
import re
import csv
import ipaddress

REG_EXPR = '^2016-0[78]-\d{2} \d{2}:\d{2}:\d{2},
\d+\.\d+,\d+\.\d+\.\d+\.\d+,\d+\.\d+\.\d+\.\d+,
\d+,\d+,[A-Z]+,[A-Z\.]{6},\d+,\d+,\d+,\d+,.+[sd]$'

def convert(line):
    outrow = []
    row = line.split(',')
    #print("type of row:",type(row),row)
    d, t = row[0].split(' ')
                # split the date: 2016-08-01 into 3 columns
    for val in d.split('-'):
        outrow.append(val)

                # split the time: 07:59:46 into 3 columns
    for val in t.split(':'):
        outrow.append(val)
    outrow.append(row[1])

    # convert IP adresses to integer value
    outrow.append(int(ipaddress.ip_address(row[2])))
    outrow.append(int(ipaddress.ip_address(row[3])))
    outrow.append(row[4])
    outrow.append(row[5])

    # convert: TCP -> 1 ; UDP -> 2 ; SCTP -> 3 & rest -> 4
    if row[6] == 'TCP':
        outrow.append(1)
    elif row[6] == 'UDP':
        outrow.append(2)
    elif row[6] == 'SCTP':
        outrow.append(3)
    else:
        outrow.append(4)

    # convert Flags to ASCII value & split it into different columns
    for ch in row[7]:
        outrow.append(ord(ch))

```

```
    outrow.append(row[8])
    outrow.append(row[9])
    outrow.append(row[10])
    outrow.append(row[11])

    # convert dos -> 0 & rest -> 1
    if row[12] == 'background\n':
        outrow.append(0)
    elif row[12] == 'dos\n':
        outrow.append(1)
    else:
        print ("type -",row[12])
        return 0
    return outrow


def main(infile, outfile):

    if outfile == "":
        outfile = '/dos1.csv'
    if infile == "":
        infile = 'data/original/file1.csv'
    with open(infile) as f:
        with open(outfile, 'w',newline='') as csvfile:
            writer = csv.writer(csvfile)
            r = re.compile(REG_EXPR)

            line = f.readline()
            while line:
                if r.match(line):
                    if (convert(line) == 0):
                        pass
                    else:
                        writer.writerow(convert(line))
                else:
                    pass

                line = f.readline()

if __name__ == "__main__":
```

Listing A.1: Data cleaning script

A.2 Decision Tree

```
import pandas as pd
import numpy as np
np.random.seed(5)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

#read data
mydt = pd.read_csv("original_data.csv")
print("original data-",mydt.shape)
columns=['year','month','day','hour','minute','second','duration',
'source_ip','dest_ip','source_port','dest_port','protocol','flag1',
```

```
'flag2','flag3','flag4','flag5','flag6','fwd','stos','pkt','byt']

#preparation
X = mydt.iloc[:, 3:22].values
y = mydt.iloc[:, 22].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.2,random_state=5)
print("for normal -",sum(y_train==0))
print("for attack -",sum(y_train==1))

#Model
clf_entropy = DecisionTreeClassifier(criterion = "entropy",
random_state = 5,max_depth = 5, min_samples_leaf = 5)

print("Results using Entropy")
clf_entropy.fit(X_train,y_train)
y_pred_entropy = clf_entropy.predict(X_test)
print("Confusion matrix \n",confusion_matrix(y_test,y_pred_entropy))
```

Listing A.2: Decision Tree classifier

A.3 Support Vector Machine

```
import pandas as pd
import numpy as np
np.random.seed(5)
from sklearn.preprocessing import MinMaxScaler
from  sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

#data
mydt = pd.read_csv("original_data.csv")
columns=['year','month','day','hour','minute','second','duration',
'source_ip','dest_ip','source_port','dest_port','protocol',
'flag1','flag2','flag3','flag4','flag5','flag6','fwd',
'stos','pkt','byt']

# convert the columns value of the dataset as floats
float_array = mydt.values.astype(float)

# create a min max processing object
min_max_scaler = MinMaxScaler()
scaled_array = min_max_scaler.fit_transform(float_array)

# convert the scaled array to dataframe
mydt_normalized = pd.DataFrame(scaled_array)

#  Create matrix of features and matrix of target variable
X = mydt_normalized.iloc[:, 3:22].values
y = mydt_normalized.iloc[:, 22].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=5)

#model
```

```
svclassifier=SVC(C=0.8,kernel='rbf',max_iter=10000,random_state=0)
svclassifier.fit(X_train,y_train)
y_pred = svclassifier.predict(X_test)

print ("For SVM - ")
print (confusion_matrix(y_test,y_pred))
```

Listing A.3: Support Vector Machine

A.4 Neural Network

```
print(hash("keras"))
import numpy as np
np.random.seed(2)
import tensorflow as tf
tf.set_random_seed(2)
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
from keras.models import model_from_json

#data
mydt = pd.read_csv("original_data.csv")
columns=['year','month','day','hour','minute','second','duration',
'source_ip','dest_ip','source_port','dest_port','protocol',
'flag1','flag2','flag3','flag4','flag5','flag6','fwd',
'stos','pkt','byt']

# convert the columns value of the dataset as floats
float_array = mydt.values.astype(float)

# create a min max processing object
min_max_scaler = MinMaxScaler()
scaled_array = min_max_scaler.fit_transform(float_array)

# convert the scaled array to dataframe
mydt_normalized = pd.DataFrame(scaled_array)

#  Create matrix of features and matrix of target variable
X = mydt_normalized.iloc[:, 3:22].values
y = mydt_normalized.iloc[:, 22].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state=5)

#Initializing Neural Network
classifier = Sequential()
# Adding the input layer and the first hidden layer
classifier.add(Dense(output_dim = 64, init = 'uniform',
```

```
activation = 'relu', input_dim = 19))
# Adding the second hidden layer
classifier.add(Dense(output_dim = 1, init = 'uniform',
activation = 'sigmoid'))
# Compiling Neural Network
adm = optimizers.Adam(lr=0.001)
old_model = classifier.compile(optimizer = adm,
loss = 'binary_crossentropy', metrics = ['accuracy'])
# Fit the model
history = classifier.fit(X_train, y_train,shuffle=False,
epochs=20, batch_size=10000,validation_data=(X_val,y_val))

#testing
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
cm = confusion_matrix(y_test, y_pred)
print ("confusion_matrix for test data\n",cm)

#save model
model_json = classifier.to_json()
with open("IP_model.json","w") as json_file:
  json_file.write(model_json)
classifier.save_weights("IP_model.h5")
```

Listing A.4: Neural Network

A.5 Wasserstein GAN

```
import torch
import torch.nn
import torch.nn.functional as nn
import torch.autograd as autograd
import torch.optim as optim
import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os
from torch.autograd import Variable
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import torch.utils.data as Data

#data
df = pd.read_csv("original_data.csv")

#hyperparameters
mb_size = 64
z_dim = 100
X_dim = 19
y_dim = 1
h_dim = 128
cnt = 0
lr = 0.00005

```

```
#data preparation
criteria0 =  df['label'] == 1  #just attack data
data0 = df[criteria0]
float_array = data0.values.astype(float)
min_max_scaler = MinMaxScaler()
scaled_array = min_max_scaler.fit_transform(float_array)
dt = pd.DataFrame(scaled_array)
dt.columns=['year','month','day','hour','minute','second',
'duration','source_ip','dest_ip','source_port','dest_port',
'protocol','flag1','flag2','flag3','flag4','flag5','flag6',
'fwd','stos','pkt','byt','label']
dt=dt.drop(dt.index[0:132])
print(dt['label'].value_counts())

train_target = torch.tensor(dt['label'].values.astype(np.float32))
train = torch.tensor(dt.drop(['year','month','day','label'],
axis = 1).values.astype(np.float32))
train_tensor = Data.TensorDataset(train, train_target)
train_loader = torch.utils.data.DataLoader(dataset =
train_tensor, batch_size = mb_size, shuffle = True)

#define models
G = torch.nn.Sequential(
    torch.nn.Linear(z_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, X_dim),
    torch.nn.Sigmoid()
)

D = torch.nn.Sequential(
    torch.nn.Linear(X_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, 1),
)
G.cuda()
D.cuda()

def reset_grad():
    G.zero_grad()
    D.zero_grad()

#initialize optimizers and dataholders
G_solver = optim.RMSprop(G.parameters(), lr=lr)
D_solver = optim.RMSprop(D.parameters(), lr=lr)
G_losses = []
D_losses = []
data_iter = iter(train_loader)

#training
d_iter = 2
for it in range(500000):
    if it<100:
        d_iter = 50
    else:
        d_iter = 2
    for _ in range(d_iter):
```

```
        # Sample data
        z = Variable(torch.randn(mb_size, z_dim)).cuda()
        try:
            X, _ = data_iter.next()
        except StopIteration:
            data_iter = iter(train_loader)
            X, _ = data_iter.next()
        #X = Variable(torch.from_numpy(X))
        X = X.cuda()
        # Dicriminator forward-loss-backward-update
        G_sample = G(z)
        D_real = D(X)
        D_fake = D(G_sample)

        D_loss = -(torch.mean(D_real) - torch.mean(D_fake))

        D_loss.backward()
        D_solver.step()

        # Weight clipping
        for p in D.parameters():
            p.data.clamp_(-0.01, 0.01)

        D_losses.append(D_loss.data.cpu().numpy())

        # Housekeeping - reset gradient
        reset_grad()

    # Generator forward-loss-backward-update
    #X = Variable(torch.from_numpy(X))
    z = Variable(torch.randn(mb_size, z_dim)).cuda()

    G_sample = G(z)
    D_fake = D(G_sample)

    G_loss = -torch.mean(D_fake)

    G_loss.backward()
    G_solver.step()

    # Housekeeping - reset gradient
    reset_grad()

    G_losses.append(G_loss.data.cpu().numpy())

    # Print and plot every now and then
    if it % 1000 == 0:
        print('Iter-{}; D_loss: {}; G_loss: {}'
        .format(it, D_loss.data.cpu().numpy(),
              G_loss.data.cpu().numpy()))

#plot losses
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.savefig('losses.png')
```

```
#save model
torch.save({'modelG_state_dict': G.state_dict(),
            'modelD_state_dict': D.state_dict(),
            'optimizerG_state_dict': G_solver.state_dict(),
            'optimizerD_state_dict': D_solver.state_dict(),
            'G_loss':G_losses,
            'D_loss':D_losses
           }, "wgan_3layer_model.pth")
```

Listing A.5: Wasserstein GAN

A.6 Data generation

```
import numpy as np
np.random.seed(2)
import tensorflow as tf
tf.set_random_seed(2)
import torch
import torch.nn
import torch.nn.functional as nn
import torch.autograd as autograd
import torch.optim as optim
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import os
from torch.autograd import Variable
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
import torch.utils.data as Data
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.models import model_from_json

mb_size = 64
z_dim = 100
X_dim = 19
y_dim = 1
h_dim = 128
print (X_dim,y_dim)

#this loaded model will vary for each black box model.
#Only NN used here.
json_file = open("model.json","r")
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model.h5")

G = torch.nn.Sequential(
    torch.nn.Linear(z_dim, h_dim),
```

```
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, h_dim),
    torch.nn.ReLU(),
    torch.nn.Linear(h_dim, X_dim),
    torch.nn.Sigmoid()
)

checkpoint = torch.load("wgan_3layer_model.pth")
G.load_state_dict(checkpoint['modelG_state_dict'])

G.eval()
G.cuda()

final_Data = pd.DataFrame(columns = ['hour','minute','second',
'duration','source_ip','dest_ip','source_port','dest_port',
'protocol','flag1','flag2','flag3','flag4','flag5','flag6',
'fwd','stos','pkt','byt','label'])
print("empty df",final_Data)
it =0

while(final_Data.shape[0]<12000000):
  it+=1
  z = Variable(torch.randn(3000, z_dim)).cuda()
  with torch.no_grad():
    data = G(z)
  data = data.cpu().numpy()
  fake = pd.DataFrame(data)

  newy = np.ones((fake.shape[0],))
  #this is the model which changes
  new_pred = loaded_model.predict(fake)
  new_pred = (new_pred > 0.5)
  cm = confusion_matrix(newy,new_pred)
  #print ("confusion_matrix for new data with deeper model\n",cm)

  combined_data = np.concatenate((fake,new_pred),axis=1)
  combined_data = pd.DataFrame(combined_data)
  combined_data.columns=['hour','minute','second','duration',
  'source_ip','dest_ip','source_port','dest_port','protocol',
  'flag1','flag2','flag3','flag4','flag5','flag6','fwd',
  'stos','pkt','byt','label']
  criteria1 =  combined_data['label'] == 1  #just attack data
  data1 = combined_data[criteria1]
  #print("only attack data - ",data1.shape)
  final_Data = final_Data.append(data1, ignore_index = True)
  print("final data for iteration ",it, "- ",final_Data.shape)

final_Data.to_csv("generated_data.csv",index=False,header=False)
```

Listing A.6: Generating data from saved GAN model

A.7 Combining datasets

```
newdt = np.genfromtxt("generated_data.csv",delimiter=',')
new_data = newdt[:, 0:19]
newy = newdt[:, 19]
```

```
combined_X = np.concatenate((X_train,new_data),axis=0)
combined_y = np.concatenate((y_train,newy),axis=0)
```

Listing A.7: combining original and generated datasets

A.8 Hypothesis Testing

```

import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt


df1 = pd.read_csv("unbalanced_dos.csv")
df2 = pd.read_csv("original_data.csv")
df3 = pd.read_csv("generated_Data.csv")
columns=['hour','minute','second','duration','source_ip',
'dest_ip','source_port','dest_port','protocol','flag1',
'flag2','flag3','flag4','flag5','flag6','fwd','stos','pkt','byt']
new_columns=['hour','minute','second','duration','source_ip',
'dest_ip','source_port','dest_port','protocol','flag1','flag2',
'flag3','flag4','flag5','flag6','fwd','stos','pkt','byt','label']

df1 = df1.drop(['year','month','day'],axis=1)
df2 = df2.drop(['year','month','day','label'],axis=1)

criteria = df1['label'] == 1
data1 = df1[criteria]
data1 = data1.drop(['label'],axis=1)
df3 = df3.iloc[:, 0:19]
print("df1 - ",df1.shape)
print("df2 - ",df2.shape)
print("df3 - ",df3.shape)

frac = 10000/data1.shape[0]
data_original = data1.sample(frac=frac)
print("df1 sampled - ",data_original.shape)

df_all = data_original.merge(df2.drop_duplicates(),
on=columns, how='left', indicator = True)
print("merged- ",df_all.shape)

data_original = df_all[df_all['_merge'] == 'left_only']
data_original = data_original.drop(['_merge'],axis = 1)
print("final sampled data of original= ",data_original.shape)

float_array = data_original.values.astype(float)
```

```
min_max_scaler = MinMaxScaler()
scaled_array = min_max_scaler.fit_transform(float_array)
data_original = pd.DataFrame(scaled_array)
data_original.columns=columns
data_original['label'] =pd.DataFrame
    (np.zeros((data_original.shape[0],)),dtype=int)

print("label values: ", data_original['label'].value_counts())

frac = data_original.shape[0]/df3.shape[0]
data_fake = df3.sample(frac=frac)
print("final sampled data of generated= ",data_fake.shape)
zer = pd.DataFrame(np.zeros((data_fake.shape[0],)),dtype=int)
data_fake.columns = columns
data_fake['label'] = 1

final_Data = pd.concat([data_original,data_fake],
ignore_index = True,axis=0)
final_Data = final_Data.sample(frac=1.0)

print("label values: ", final_Data['label'].value_counts())

X = final_Data.iloc[:, 0:19].values
y = final_Data.iloc[:, 19].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2,random_state=5)

print("class 0 -",sum(y_train==0))
print("class 1 -",sum(y_train==1))

model = Sequential()
model.add(Dense(output_dim = 20, init = 'uniform',
activation = 'relu', input_dim = 19))
model.add(Dense(output_dim = 1, init = 'uniform',
activation = 'sigmoid'))

adm = optimizers.Adam()
new_model = model.compile(optimizer = adm,
loss = 'binary_crossentropy', metrics = ['accuracy'])
history = model.fit(X_train, y_train ,shuffle=False,
epochs=100, batch_size=1000)

y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
cm = confusion_matrix(y_test, y_pred)
print ("confusion_matrix for test data\n",cm)
```

Listing A.8: Hypothesis testing script

A.9 PCA plots

```
from sklearn.decomposition import PCA as sklearnPCA

pca = sklearnPCA(n_components=2)
transformed = pd.DataFrame(pca.fit_transform(X_train))

```

```
plt.scatter(transformed[y_train==0][0],transformed[y_train==0][1],
label='original', c='red')
plt.scatter(transformed[y_train==1][0], transformed[y_train==1][1],
label='generated', c='blue')

plt.legend()
plt.savefig("pca_significance.png")
```

Listing A.9: Script for plotting PCA plots