Northumbria University
NEWCASTLE

Towards GAN- based Evasion Attacks and Countermeasures

Author
Shan Ali
shan.ali@northumbria.ac.uk

Supervisor
Dr. Nauman Aslam
nauman.aslam@northumbria.ac.uk

Student ID: W18041462

January 2021

KF7029 MSc computer science and digital technologies project

Northumbria University Faculty of Engineering and Environment

Abstract

Information technology is evolving rapidly and giving rise to security concerns. Botnet attacks have matured and became a serious threat to organizations. In the last decade, machine learning has produced plausible results in several different real-world problems. Cybercriminals take advantage of machine learning techniques by generating evasion attacks to compromise the system security. Adversarial evasion attacks are the most dangerous threat to machine learning-based Intrusion Detection Systems (IDSs). There is an indispensable need for state-of-the-art IDS for the detection of evasion attacks. To address this problem, a machine learning-based approach was proposed to mitigate the effect of adversarial evasion attacks on the IDSs. In this work, the evasion rate was reduced from 99.58% to 5.66%. Generative Adversarial Networks (GANs) were used for mitigating the effects of adversarial evasion attacks on the machine learning-based IDS.

Declaration

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is .....

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on the eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other Department or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED:

DATE:

Acknowledgments

I would like to thank, Dr. Nauman Aslam for supervising this research and for his assistance throughout the project. I also want to extend my gratitude to my seniors, Mr. Hasnain Rafique and Mr. Rizwan Hamid. This project would not be possible without the generosity and care of Mr. Hasnain Rafique, he motivated me to work on this idea.

To my parents, whose continuous support for my education has opened up the world to me, thank you very much. Their encouragement and motivation pushed me to achieve things that I couldn't have otherwise. While writing this section, I am thinking and extending my gratitude to my close relatives, who are no more in this world, but their efforts of pushing me for higher education are immeasurable. My heartfelt thanks to my family, especially my brother Nauman Ali. I also want to thank my best friend Waqar for motivating me to study from abroad.

Contents

1. Introduction

1.1 Background

Daily usage of the internet and proliferation of information technology has given rise to the exigence of system security. Malicious attacks accompanied by the latest technologies endeavor to compromise system security. Companies are investing extensively to optimize the detection accuracy of these attacks (da Costa, et al., 2019). Moreover, due to the continuous evolution of malicious attacks, researchers are exploring new techniques to design robust and efficient intrusion detection systems (IDS) (KarsligEl, et al., 2017) (Bamakan, et al., 2016) (Wang, et al., 2017). The core functionality of IDS is to identify similarities between a range of malware and alert user about the significance of threats that is impossible with a traditional firewall (Farnaaz & Jabbar, 2016). Technology has evolved and giving advantages to attackers to use new technologies to compromise system security by generating evasion attacks. Thus, there is an indispensable need for novel IDSs for the detection of malware. Within the last two decades, the use of IDS technology has gained significant importance and proactively analyzing indicative patterns from the flow of traffic. The use of machine learning has emerged in the security-sensitive application including botnet detection, spam classification, and development of IDS (Biggio, et al., 2010) (Biggio, et al., 2013). To design sophisticated IDS, machine learning has emerged as a prime interest in the research community. Machine learning-based IDS is used to leverage the pattern recognition of malign traffic in the network (Khraisat, et al., 2019). However, recent studies also proved that the presence of adversary in machine learning (Goodfellow, et al., 2014).

Objectives of adversarial machine learning attacks depend on the nature of an algorithm written to manipulate the decision of trained IDS. An adversary may be achieved by adjusting the features perturbation of the network flow, forcing IDS to identify malicious traffic as benign making false negatives (Dalvi, et al., 2004). Security-sensitive applications are intrinsic in nature and data distribution is not stationary. Therefore, the attacker takes this weakness into account and tries to compromise the system by launching a black-box attack on the deployed IDS (Biggio, et al., 2013). In the last few years, deep learning has proved in providing remarkable

solutions for complex problems including computer vision, speech recognition, and transformation of health care systems. Attackers usually obfuscate malign data in order to fool the IDS by treating it as benign data. A technique to generate realistic synthetic botnet data is proposed using Generative Adversarial Networks (GAN), called GAN-bots, to test the decision making power of trained classifier about possible evasions. GANs have produced plausible results by mimicking complex probability distributions (Engelmann & Lessmann, 2020).

This work has attempted to develop an IDS for botnet detection with minimal chances of evasion attack. The ISCX-Bot-2014 dataset has employed to develop different machine learning-based models and these models will be evaluated on the basis of their performances. This could be a step towards the mitigation of evasion attacks on IDS by using synthetic data generated through GAN. The botnet is a widely recognized term used in cybersecurity. Particularly, a botnet is a network of compromised devices with a botnet program running on them and connected to the internet. These compromised devices are remotely controlled by botmaster and can cause severe attacks including distributed denial of service (DDoS), disseminating illegal activities, and click fraud (Yu, et al., 2013) (Yu, et al., 2011).

Considering professional, ethical, and legal issues the dataset has open access and is maintained by the University of Brunswick (UNB). There is no involvement of professional, legal, or ethical risks in the use of the dataset. Further information can be found in Section 2.2.

1.2 Research Aim

This research work aims to develop techniques to mitigate the effects of GAN-based evasion attacks on the machine learning-based intrusion detection system (classifier) using synthetic data, such as GAN-bots.

1.3 Approach

The research approach was distributed into 5 different phases (Figure 1.1) to attain our aim of this work. Following are the phases that were considered to strengthen the detection ability of the machine learning model. These phases will ensure a countermeasure against future GAN-based evasion attacks on the trained classifier.

1. Data pre-processing

ISCX-Bot-2014 dataset required additional processing to make it useful for data analysis. Exploration of the dataset and extraction of useful features were examined. Unnecessary data columns were dropped and labeling of data involved in this phase.

2. Training and evaluation of classifiers

The pre-processed dataset was split into 70:30 train-test sets. Support Vector Machine (SVM), k-Nearest Neighbors (KNN), Logistic Regression (LR), Naïve Bayes (NB), and Decision Tree (DT) machine learning models were trained and tested. The performances of all trained models were compared on the basis of evaluation matrices, including accuracy score, recall score, and precision score. These evaluation scores of all models were compared to select the best model concerning performance. The selected classifier will be served as an IDS for further testing and evaluation.

3. Generation of Synthetic data using GAN

To make the classifier aware of the evasion attack, synthetic data was used which is also a crucial part of this research work. To achieve this GAN was employed, see section 2.9. GAN-bots were generated and saved in a "csv" file that will serve as synthetic data for further use.

4. Evasion attack

These GAN-bots, 5000 in total were augmented with randomly selected 5000 benign samples from the dataset and converted into a single unified set. This unified dataset was used as a black box attack on the trained classifier. Evasion rate recorded and evaded samples(those samples that were misclassified as benign by the model) saved in a separate file.

5. Retraining of model, regeneration of synthetic data, and evasion attack

This phase consists of three steps. Firstly, evaded samples from step 4 were augmented with the ISCX-Bot-2014 dataset to retrain the selected optimum classifier. Secondly, new 5000 GAN-bots were generated by repeating step 3. Lastly, to relaunch an attack on the retrained model, these newly generated GAN-bots were combined with the training set to test the strength of our model against evasion attack.

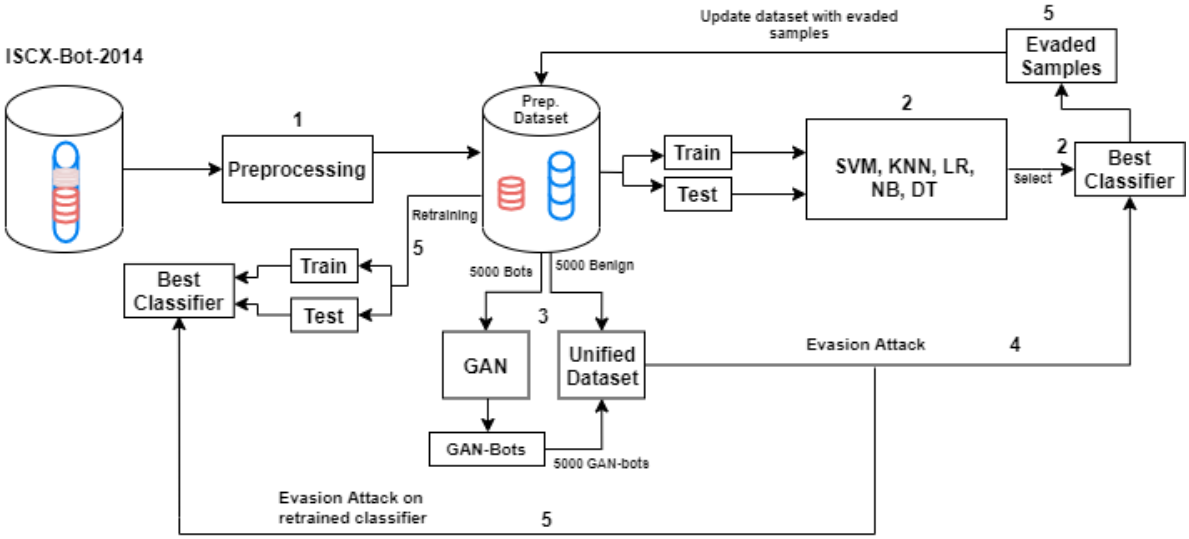

Figure: 1.1: Functional diagram of the research approach where each number represents the number of the approach taken.

1.4 Report Structure

This work is divided into five sections. The summary of five different sections is given below.

Section 1: Introduction

The basic background, as well as the research's aim, is summarised. The scope and objectives of the project with the approaches that were utilized to achieve the results are laid out. Professional, ethical, social, and legal issues were outlined.

Section 2: Context

A brief literature review and relevance are identified in which this research work lies. The technologies and concepts used to acquire the objectives of the project are given including a cursory background of the architecture of all five classifiers and GAN.

Section 3: Methodologies and Design

A summary of data pre-processing, analysis, and exploration are discussed. Additionally, the overview of approaches that were taken for the development of multiple machine learning models and the GAN model were discussed.

Section 4: Results and Analysis

Results for all five classifiers were analyzed and compared with the critical evaluation on the basis of evaluation matrices. Synthetic data generated using GAN is also provided.

Section 5: Conclusion

A concluding assessment and evaluation of the work accomplished are given. Also discussion on how the original problem was solved with the limitations and future work that is relevant to the project outlined in this section.

2. Context

2.1 Intrusion detection systems

Research on computer security, especially on intrusion detection exists since the birth of computers. An IDS is a tool to classify violations of security, attacks, and intrusions proactively at the network-level and host-level. Intrusion detection methods are divided into two main techniques: Signature-based Intrusion Detection Systems (SIDS) and Anomaly-based Detection Intrusion Detection Systems (AIDS).

2.1.1 Signature-based Intrusion Detection Systems (SIDS)

Signature-based detection is a technique that relies on finding patterns that were used to detect previous intrusions (Khraisat, et al., 2019). This technique is also known as Knowledge-based Detection. SIDS triggers an alert to the user when the pattern of a new attack has similarities with the intrusions that are already stored in our database. In addition, SIDS inspects the host's log in order to identify malicious instructions and commands those are marked or labeled as malware. The conceptual architecture of SIDS is demonstrated in Figure 2.1. The main idea of SIDS is to compare the current

activity with the existing set of stored signatures, if it matches then mark it as an intrusion else normal.

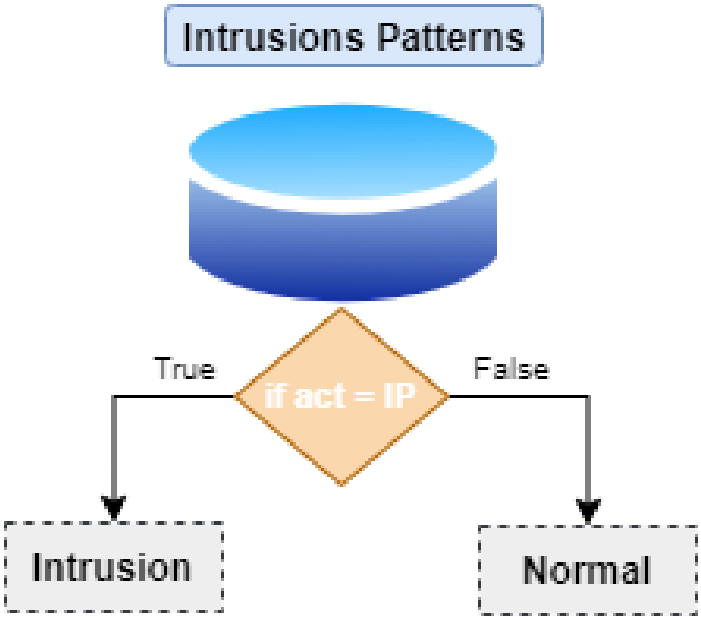

Figure 2.1: Working architecture of SIDS (act = activity, IP = intrusion pattern)

SIDS generally has been proved in providing promising results in identifying new intrusions by comparing them with known signatures (Kreibich & Crowcroft, 2004). However, SIDS is not that efficient in identifying evasion attacks or zero-day attacks because of the lack of previous knowledge for such attacks. New variations of malware attacks are rising rapidly and undermining the adequacy of traditional methods of intrusion detection. As a result, researchers and industry prefer anomaly-based detection over signature-based detection.

2.1.2 Anomaly-based Intrusion Detection Systems (AIDS)

Anomaly-based also labeled as behavior-based detection in literature were developed due to the continuous variations of attacks, and primarily for the detection of unknown malware attacks. Machine learning-based models were used to record the normal behavior of the system (i.e. how does a normal system interacts). AIDS interprets any deviations in the behavior of the suspected model and normal model as an anomaly and marks it as an intrusion (Khraisat, et al., 2019). The normal behavior of the user is different than the malicious attack, AIDS differs the normal and abnormal behavior of a user and mark abnormal behavior as an intrusion. In this research work, five machine learning-based intrusion detectors were used for the detection of botnet data(Section 2.10). Five anomaly-based intrusion detectors were developed by using

supervised machine models. The development of these models depends on training and testing. Firstly, training of the model involves learning the normal and abnormal behavior of the user from the dataset. Secondly, the accuracy of the model examined using the test set. (Butun, et al., 2013).

AIDS is categorized into three groups, Machine learning-based, Statistics-based, and Knowledge-based (Butun, et al., 2013) (Lin, et al., 2015). Knowledge-based and statistics-based intrusion detection techniques were not considered for the research but a brief introduction is given. The statistics-based technique involves building the statistical model of user behavior by examining every data record in the collection. On the other hand, the Knowledge-based approach creates a base of knowledge that reflects the legitimate flow of traffic. Any actions different from the normal flow of traffic will be classified as an intrusion. Machine learning methods (Section 2.5), have been largely explored to develop a robust and efficient IDS.

2.2 ISCX-Bot-2014 dataset

An acceptable level of realistic traffic is required for the performance evaluation of any detection techniques. According to network traffic censorship, it is recommended to target traffic that is available in the form of datasets. However, data quality also plays a crucial role in the evaluation of IDS. Due to the absence of such datasets for the evaluation of intrusion detectors, researchers have to face several challenges.

In the literature, the most prestigious approaches used for the evaluation of anomaly-based intrusions were based partially or fully on synthetic datasets (Tavallaee, et al., 2010). (Aviv & Haeberlen, 2011) shed light on the two main reasons why network traces are not publicly available. Firstly, network traces can hold confidential and sensitive information about the affected organization. Sometimes these traces are not even shared within the same company. Secondly, network traces also hold information about the vulnerabilities of the network that could cause potential damage to other companies. To address the problem, (Beigi, et al., 2014) constructed a dataset by preciously examining the features of botnet traces.

The ISCX-Bot-2014 dataset is a combination of publicly available datasets, CTU-13 (Garcia, 2013), ISCX-2012-IDS (Shiravi, et al., 2012), and ISOT dataset (Zhao, et al., 2013). ISCX-Bot-2014 specifically focuses on botnets. This dataset is the

implementation of the 16 modern types of botnets, blended with the instances of normal and botnet traffic. Dataset is publicly available on the website of the University of New Brunswick UNB's and seems promising in terms of realism, representativeness, and generality. Realism can be defined as the similarity of generated data with the actual traffic. The reflection of the real environment or the ability to reflect real traffic that botnet experience during traces determines representativeness. And generality defines the richness of botnet behavior during evasion attacks.

Original ISCX-Bot-2014 dataset is available in the form of .pcap (WireShark, a tool for packet analysis of a network) files where the authors already divided the dataset into a training set and test set. However, because they capture botnet attacks using WireShark, the training set and test set were large and need to be converted into .csv (MS Excel) file, more details in Section 3.1. Ethical and legal issues could arise when it comes to the sensitivity of the dataset, but the dataset is publicly available and has open access for everyone to so, there is no need for considering these issues.

2.3 Machine Learning

Machine learning models have been extensively used for the exploration and development of robust IDSs. Machine learning is evolved from knowledge and information extraction from large and complex data. Machine learning algorithms search for patterns in the data to make data-driven predictions (Hamid, et al., 2016). In literature, three main problems of machine learning were identified: Unsupervised Learning, Reinforcement Learning, and Supervised Learning. This research's scope is limited to Supervised Learning and unsupervised learning.

Supervised machine learning algorithms are used when labeled data fed to a model for training purposes (Bell, 2020). The goal of supervised learning models is to learn the relationships and mapping within the input data. After exploration of patterns in between the set of inputs, the algorithm trains itself on that patterns for making further predictions on the new same set of inputs. Supervised learning solves two types of problems "Classification" and "Regression". Classification problem occurs when the learning algorithm has to make predictions on discrete data, such as the algorithm has to classify a set of input features whether they belong to a benign class

or botnet class. On the other hand, the Regression problem occurred when the algorithm has to predict continuous values. This research work lies in the classification problem and the methods used to solve this problem, can be found in Section 2.10.

Unsupervised learning is a type of machine learning in which unlabelled data are given to the learning algorithm to automatically discover and learn hidden patterns in a dataset. Unsupervised learning algorithms may be used to discover and group similar data, which is also known as clustering. Clustering is the problem of unsupervised learning which deals in finding patterns and structures in a collection of input data. Besides, density estimation is also an unsupervised learning problem. The goal of density estimation is to determine how the input data is dispensed in the space. In recent years, unsupervised learning has gained huge attention because of the popularity of generative modeling, which is a type of unsupervised learning. In this research, a famous framework of generative modeling is used, which is Generative Adversarial Networks (Section 2.9)

2.4 Deep learning

Deep learning is a stimulating and highly recognized approach of machine learning these days. Deep learning enables systems to learn from experiences and form a hierarchy of concepts. These concepts allow systems to learn complicated patterns by combining multiple non-linear transformations to form simple building blocks (Goodfellow, et al., 2016). In the past few years, the tremendous advancements in computers' hardware had allowed the research community to use deep learning on bigger data, e.g., images. As a result, deep learning has gained the huge attention of the research community by producing plausible results in natural language processing, self-driving vehicles, and computer vision. Deep learning works by using Artificial Neural Networks (ANNs). ANN models the generalization and learning capabilities of real-world biological neurons. ANN consists of three layers, an input layer, a hidden layer, and an output layer. Incrementation of hidden layers form a Deep Neural Networks (DNNs), it is known as "deep" because of the presence of multiple layers. The architecture of both ANNs and DNNs is the same except for the number of hidden layers. Every node in each layer is has a connection to all nodes of the upcoming layer

with some weighted connections (weights) that represent the strength between them (Wermter, et al., 2014). A classical neural network with three layers and a process of activation state is shown in Figure 2.2.

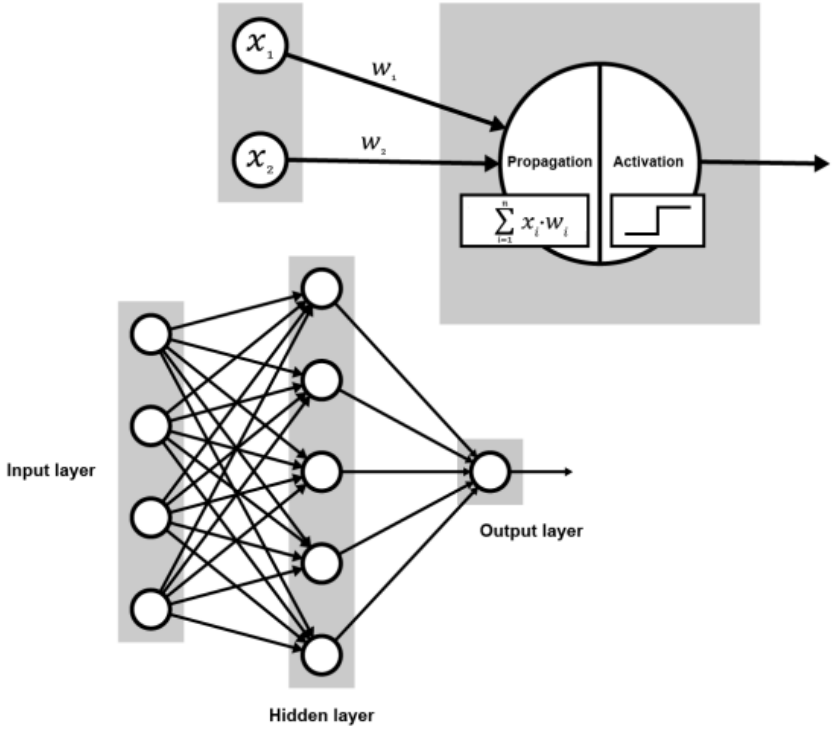

Figure 2.2: A multilayer neural network with single-hidden layer

The architecture of a neural network (NN) depends on the complexity of data. For a single hidden layer neural network, the input layer firstly receives data x_i through its nodes, then feeds the data to a hidden layer through some adjustable initial weights. Then a propagation function receives that data and transforms it by calculating the weighted sum of weights w_i and x_i. The activation function receives transformed data and converts it into an output vector for the following layers, activation step ensures the non-linearity of output. This function also helps in computing the average gradient of the model by adjusting weights according to the input vector that assists in backpropagation (LeCun, et al., 2015). The neural network takes four major steps to calculate backpropagation: 1) randomly initialize weights, 2) feed-forward, propagate from left to right, until NN not getting the predicted result, 3) compare predicted result with expected results means calculate loss, 4) propagate right to left, update weights.

Several activation functions exist, usage of each function depends on the data. The simplest one is a binary function, it is suitable for binary problems where the

output converges either 0 or 1, as shown in Eq. 1. Binary function is impractical as it cannot classify output other than 0 and 1.

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (1)$$

The sigmoid function produces an S-shaped curve with values ranging from 0 and 1. The sigmoid function ensures the non-linearity which allows the output to be continuous. However, output values also stay in the positive range (0,1) which makes it less preferable over tanh function. Tanh function addresses the problem faced by the sigmoid function, it differentiates the output between -1 and 1. However, sigmoid and tanh functions are prone to the machine learning problem "vanishing gradient" (Goodfellow, et al., 2016). The rectified linear unit (ReLU), simply outputs the input as 1 if positive and 0 if negative, ReLu function is one of the most used activation function these days, shown in eq. 2. This research is limited to the use of sigmoid and ReLU activation functions.

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \quad (2)$$

2.5 Machine Learning in Cyber Security

In recent years, machine learning technologies emerged and used exponentially to find interesting and complex patterns in the data (Dua & Du, 2016). The scope of this research is within the creation of efficient and robust IDS to mitigate the evasion attacks, so literature related to a particular type of research was explored. Several unsupervised and supervised machine learning were explored for knowledge extraction from datasets (Xiao, et al., 2018) (Kshetri & Voas, 2017). In addition, Supervised learning models such as Random Forest and Deep Neural Network (DNN) were explored for detection of botnets (Santana, et al., 2018). The goal of using machine learning-based IDS is to empower the system and learn from data-patterns to predict whether the behavior of activity is abnormal or normal. As mentioned above machine learning-based methods have been vastly used for the detection of AIDS (Section 2.1.2) and SIDS (Section 2.1.1), so it shows the importance of machine learning in revolutionizing the field of cybersecurity.

2.6 Botnet Detection

Technology is evolving at a rapid pace, bringing benefits to almost every field of human life. However, cybercriminals continuously trying to penetrate the security of new systems for the sake of personal benefits using malware attacks. As mention earlier botnet is a network of infected computers controller by a botmaster to launch system attacks. The botnet is classified into three main categories, Peer-2-Peer (P2P)-based, Internet Relay Chat (IRC)-based, and Hypertext Transfer Protocol (HTTP)-based (Acarali, et al., 2016). P2P-based botnets operate on decentralized architecture in which every node can perform functionalities as a server and client. In contrast, both HTTP-based and IRC-based botnets use centralized architecture in which to access the command and control (C&C) server bots require to perform data communication (Acarali, et al., 2016).

This research only focuses on the detection of HTTP-based botnets. HTTP is a communication protocol, due to the vast usage of internet-based applications, HTTP-based botnets have many advantages over other botnets (Alkasassbeh & Almseidin, 2018). For instance, the firewall allows HTTP traffic for communication between the client and server, the attacker takes advantage of this point and tries to blend HTTP-based bots to blend with the normal traffic (Gu, 2008). So, from the attackers' perspective, HTTP has more advantages over IRC (Farina, et al., 2016). Moreover, the attack behavior of botnets depends on the nature of commands they receive from the botmaster to conduct a variety of attacks, including DDoS, SQL injection, and man-in-the-browser.

2.7 Security of Machine Learning

Security of AI-based systems has emerged as a crucial concern in the past few years. Researchers and companies are continuously developing new systems and evaluating traditional systems to defend AI-based systems against novel attacks. Researchers used deep-reinforcement learning (a type of machine learning where the model learns from its experience like humans) to keep AI-based models up-to-date against new attacks (Kurakin, et al., 2018). However, due to the adversarial nature of deep learning models, the use of the deep-reinforcement technique could lead the model to train on malicious inputs. This type of adversarial attack on a machine learning model is known

as model extraction attacks or Black box attacks (Papernot, et al., 2017). Adversarial attacks are categorized as White-box attack and Black-box attack. The white-box attack is named a white box because of having all the necessary knowledge about the model, including training data, hyper-parameters, and model's architecture which makes it easier to penetrate the trained classifier. White box attacks are easy to deploy because of having all the information about the model, which also gives rise to the adversarial sample transferability (Liu, et al., 2016), which is also a hot issue these days. On the other hand, in the black-box attack, the criminal has no knowledge about the model and tries to learn the response of the model. White box attacks can be used to effectively exploit the nature of black-box attacks due to the property of transferability. The purpose of this research work is to make the classifiers adept at proactively knowing the adversarial examples so that the effect of black-box attacks could be mitigated.

Transferability of adversarial examples (also known as an adversarial attack) can be defined as if an adversarial example is successful in misleading a model 'A', it would be most likely to mislead model 'B' as shown in Figure 2.3. The main objective of adversarial example attacks is to force the model to misclassify unseen data. Authors (Papernot, et al., 2016) have proposed the same technique in which they argue that black-box attacks are generally valid to target even those models that are different from each other.

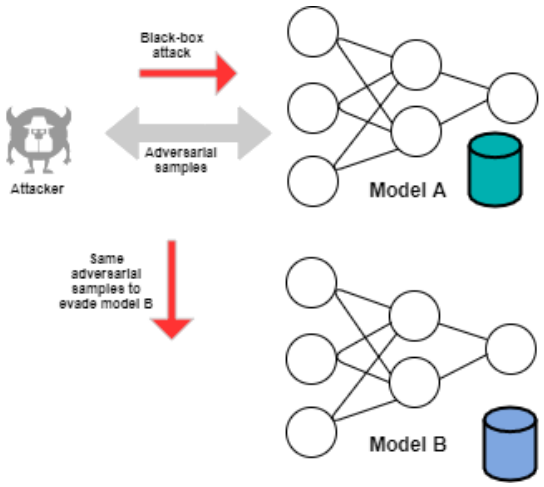

Figure 2.3: Example of Adversarial Sample Transferability. Model 'A' and model 'B' both are trained on different datasets, the attacker first attacked model 'A' and

extracted the adversarial examples, and then used the same set of adversarial examples to evade model 'B'.

2.8 Adversarial Evasion Attack

Machine learning has revolutionized healthcare systems, computer security, computer vision, and financial systems. However, the security of machine learning is at stake because of its vulnerability against adversarial attacks (Usama, et al., 2019) (Goodfellow, et al., 2014). Security of these systems is incumbent because of the sensitive nature of data or results produced by these systems. The adversarial attack can be generated by perturbated legitimate samples in order to fool the trained classifier to misclassify its prediction. These attacks could be used to bypass or evade the IDSs, financial fraud detection systems, or autonomous navigation systems of self-driving cars (Papernot, et al., 2016).

Adversarial attacks can be categorized mainly under adversarial example attacks (section 2.7). Evasion attacks term specifically refers to attacking intrusion detection systems to misclassify unseen data. The ultimate goal of attackers is to fool the IDS to misclassify malign as benign. The research community tried to solve this issue by proposing several defenses. One good defense against adversarial examples attack is to introduce randomness to a model (Wang, et al., 2017). In addition, adversarial training (Huang, et al., 2011) is also a promising defense against these types of attacks. Adversarial training is a brute force solution that demands the generation of a large number of adversarial examples to train the classifier, by injecting them into original data, in order to force the model not to get fooled by these kinds of examples. The purpose of this research work is to make the classifiers adept at proactively knowing the adversarial examples so that the effect of black-box attacks could be mitigated by using adversarial training.

2.9 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a technique of generative modeling using deep learning methods. Generative modeling is an unsupervised learning problem, more details can be found in section 2.3. Generative models can be used to generate a new set of plausible samples that follows the data distribution of the original dataset. GANs have emerged as the most popular learning approach to generate realistic data

across a range of fields, including computer vision, medical applications, and music generation (Jabbar, et al., 2020). GANs are the major advancement in generative modeling, uses deep learning models to solve a problem known as data augmentation. Data augmentation is a technique that reduces overfitting and regularization errors. Data augmentation is used to escalate the amount of data by introducing slight changes in the samples of the original dataset or by generating synthetic data. Several types of GANs have been introduced by the researcher, all of them used different techniques and architectures to solve a specific problem. GANs consists of two neural networks, a generator model, and a discriminator model. Moreover, GANs are based on a zero-sum game in which both models are adversarial to each other and attempt to overcome each other (Goodfellow, et al., 2016). GANs models framed supervised learning problems in a clever way to training generative modeling, which is an unsupervised learning problem. The discriminator model is based on supervised learning that classifies generated data (examples generated by generator) between real and fake, e.g., classification (Brownlee, 2019). In this work, Vanilla-GAN or GAN is used which consists of two deep learning models, a generator and a discriminator.

The generator model tries to generate new realistic examples by taking a fixed-sized random vector. The vector is a set of variables, randomly drawn from multi-dimensional vector space 'z', also referred to as latent space, to feed the generator. After training, the data point of this latent space will be following the distribution of the original dataset. Variables of latent space play important role in the training of generative modeling but these variables are not observable directly (Engelmann & Lessmann, 2020). Latent space (noise vector) is a compressed form of input data distribution that holds the necessary information of observed data. Noise vector provided to the generator model to generate new and different examples. The generator model can be represented as $G: z \rightarrow X$ where z is the noise vector from latent space and X is the normal real distribution of data. Figure 2.4 shows the basic architecture of the generator model.

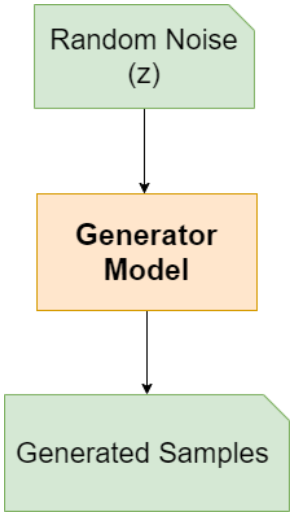

Figure 2.4: Block diagram of Generator Model

The discriminator D:X →[0,1] model is a classifier that takes an input and predicts whether the input is real or fake, as shown in Figure 2.5. Real samples referred to the input from the training set and generated samples were referred to as output by the generator. After training the model with real and generate data, the discriminator model learns the distribution of important features that differentiate real data from fake.

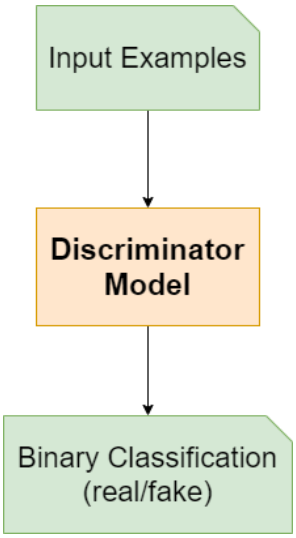

Figure 2.5: Block diagram of the Discriminator Model

The combined model, shown in Figure 2.6, is the combination of a generator and a discriminator where both models learn together by calculating the loss function. The discriminator model trains like a simple deep neural network classifier that distinguishes between real and fake data. Moreover, the discriminator model

separately looks at real data and generated input, and calculates the probability of input whether D (X) = 1 or 0. The output of the discriminator model should be 0 if the input is fake and 1 if it is real. Through this process, the discriminator calculates the loss between real and generated data and updates both models (Zhang, et al., 2019). The GAN model wants the generator to generate samples that are close to D (X) = 1 to fool the discriminator. The combined loss of GAN, shown in equation 3, backpropagates to the generator model where the model updates its hyperparameters and retries to generate samples that are close to real data.

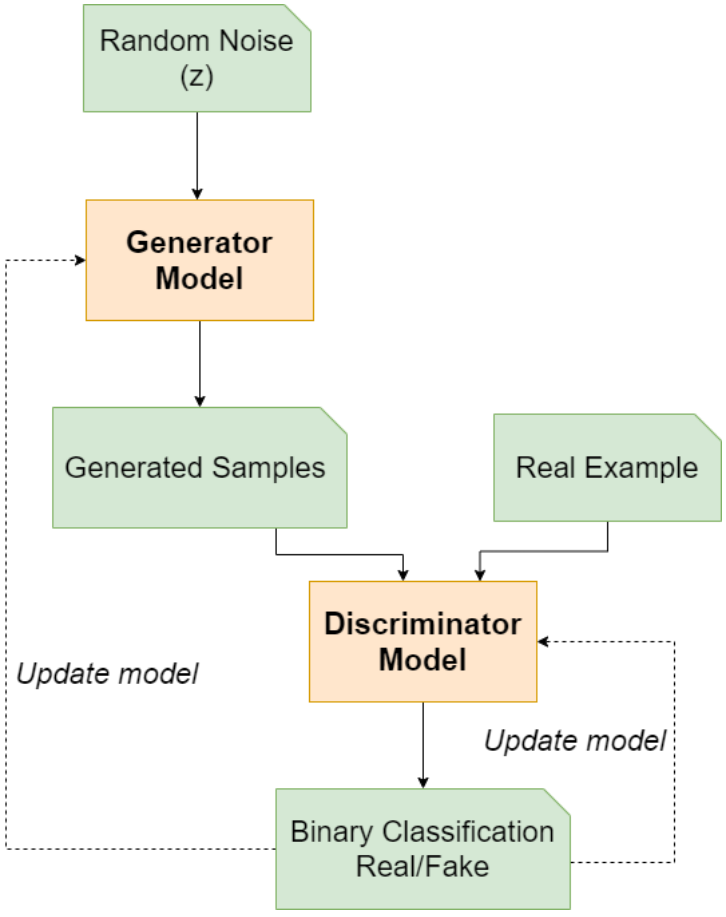

Figure 2.6: Example of GAN's Model Architecture

These two models trained together, the generator takes random noise from latent space, generates samples within the domain of real data and the discriminator model discriminates between real and generated samples and guides the generator model to generate realistic examples. In this way, these two models start playing a zero-sum game where G tries to minimize V, and D wants to maximize V (Gupta, et al., 2018). In

a zero-sum game, when the discriminator successfully distinguishes between real and fake data, it is rewarded and the generator model is penalized with updates of the model. Alternatively, when the generator model succeeds in fooling the discriminative model it receives a reward, while the discriminative model is penalized with updates in the model.

$$\min_G \max_D V(\mathrm{D},\mathrm{G}) = \mathbb{E}_{x\sim Pdata^{(x)}}[\log \mathrm{D}(x)] + \mathbb{E}_{z\sim Pz^{(z)}}[\log(1 - \mathrm{D}(\mathrm{G}(\mathrm{z})))] \quad (3)$$

Since the G cannot control the loss of discriminator model on real data but it can maximize the loss of D on generated data G(z), the loss function is given in equation 4.

$$\mathrm{J}^{G}(G) = \mathbb{E}_{z\sim Pz^{(z)}}[\log(\mathrm{D}(\mathrm{G}(\mathrm{z})))] \quad (4)$$

At a limit, the generator model starts creating plausible examples by following the likelihood of real data after every epoch and the discriminator model cannot differentiate between real and generated examples.

2.10 Supervised Learning Methods

This section includes a brief overview of machine learning-based classification models that were used in this work for the selection of the best classifier among these five models. Details of these five models are as follows.

2.10.1 Support Vector Machines

Support vector machines (SVMs) are a type of supervised learning problem that can be used as classification and regression problems. Support vector machines are also known as the Large Margin Classifier (Barlow, 1989). Support vector machines support a versatile range of kernel functions and also allow the use of custom kernels. These machines are also memory efficient. The objective of support vector machines is to find a hyperplane(a decision boundary classifies data samples) that can distinctly separate the input data point. For the separation of two classes, such as benign or

botnet, the support vector machine may choose several possible hyperplanes. The goal of this model to choose a hyperplane with large margins between data points, such as maximum distance in between benign class and botnet class. Intuitively, the algorithm of this classifier uses support vectors to maximize the margins between data samples (Schölkopf, et al., 2000). Support vectors, shown in figure 2.7, are those points close to hyperplanes these points determine the position and orientation of hyperplanes. Therefore, the lower the margin between data samples the higher the generalization error, and vice versa.

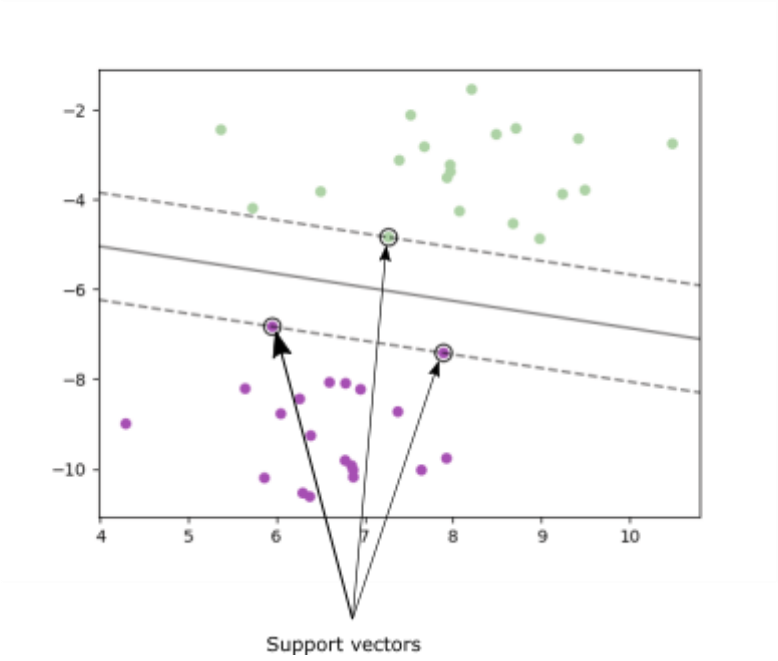

Figure 2.7: Representation of Hyperplane and Support Vectors

In this work, C-Support Vector Classification (SVC) learning algorithm is used, with a linear kernel, that is based on libsvm (an SVM library). SVC provides multiple options of kernels selection, including sigmoid, poly, and linear. A mathematical formulation of the SVC linear kernel is shown in equation 5, which is also a loss function of the SVC kernel.

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1} \quad max(0, y_i(w^T \phi(x_i) + b)) \tag{5}$$

2.10.2 k-Nearest Neighbors

k-nearest neighbors (KNNs), also known as nearest neighbors, is a simple algorithm that can be used in supervised and unsupervised learning. In supervised learning, it can perform tasks such as classification and regression. Spectral clustering, an unsupervised learning algorithm was created by getting inspired by the nearest neighbor clustering algorithm. Nearest neighbors work on the principle to find the distance between a predefined number of samples (K) that are close to a new data point (Goldberger, et al., 2004). k-neighbors is the most commonly used classification technique. In addition, this model uses Euclidean distance to measure the distance between neighbors of a data point. The selection of K which is a user-defined constant, affects the stability of the model, a low number of K increases the number of errors, and vice versa. However, selecting a large number of K is computationally expensive but suppresses noise. Classification problem computed by calculating the distance between all samples, selecting samples that are closest to the K, and votes for the most frequent labels, shown in Figure 2.8. Due to the simplicity of this algorithm, K-nearest neighbor does not train by using gradient descent like other fancy classifiers but by minimizing the mean absolute loss by using the median. In simple words, this algorithm memorizing the data by creating a local copy and search by majority votes when the test set is given to it for prediction.

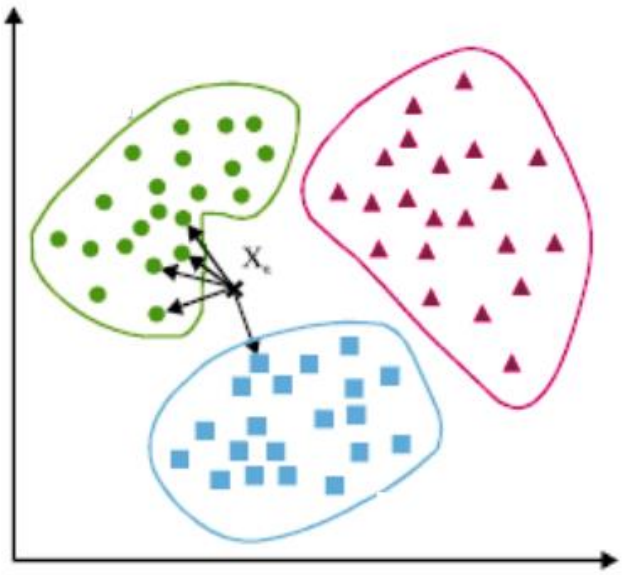

Figure 2.8: KNN classifier calculating the nearest neighbor for a new data point X.

2.10.3 Logistic Regression

Logistic Regression (LR) is a parametric classification model used to solve binary classification problems. The model is based on the Logistic function, also known as a Sigmoid function, that calculates the probability of likelihood by taking real numbers (n) and mapping them between 0 and 1, as shown in equation 6.

$$\frac{1}{1+\exp(-n)} \tag{6}$$

In order to predict value (y), logistic regression combines the input values with coefficients by using equation 7.

$$y = \frac{\exp\left(b_0+b_1x_1+b_2x_2+\cdots+b_nx_n\right)}{1+\exp(b_0+b_1x_1+b_2x_2+\cdots+b_nx_n)} \tag{7}$$

In the above equation, $b_0$is the intercept and $b_n$is the coefficient of $x_n$input (James, et al., 2013). Logistic regression learns the value of coefficients by using maximum likelihood among features in the dataset. Given that this model predicts the probabilities of output class within the range of 0 if the sample is benign and 1 if the input sample is a bot, by using a threshold given in equation 8.

$$y = \begin{cases} 0, & if\ y < 0.5 \\ 1, & if\ y \geq 0.5 \end{cases} \tag{8}$$

2.10.4 Naïve Bayes

Naïve Bayes (NB) is a supervised learning method, based on Bayes' theorem. Naïve Bayes methods use Naïve assumptions of conditional independence between predictors. In simple words, Naïve Bayes assumes that the occurrence of a certain feature individually contributes to classify a certain class without depending on each other (Zhang, 2004). Despite that, every feature in a dataset contributes to the probability of a specific class. The naïve Bayes method calculates the probability using the below equation 9.

$$P(s|X) = \frac{P(X|s)P(s)}{P(X)} \quad (9)$$

In the above equation, *P(s|X)* represents the probability of the targeted class given a set of attributes. *P(s)* is a probability of targeted class *(s).* *P(X)* can be defined as the probability of predictors *X. P(X|s)* is the likelihood of predictors *X* given by the targeted class *s*.

Naïve Bayes models are easy to implement and computationally cheap. These models do not demand large input data. Naïve Bayes models are mostly used in text classification, spam filtering, weather prediction, and medical diagnosis.

2.10.5 Decision Tree

Decision trees (DTs) are a non-parametric type of machine learning models. The objective of the decision trees' algorithm is to predict the target value by learning decision rules from the features of input data. As the name suggests, decision trees construct a tree that starts from the root. Decision trees form a series of decisions that represent branches, as going down from the node these decisions eventually lead to the resulting classification which is also known as terminal. The working diagram of decision trees is shown in figure 2.9.

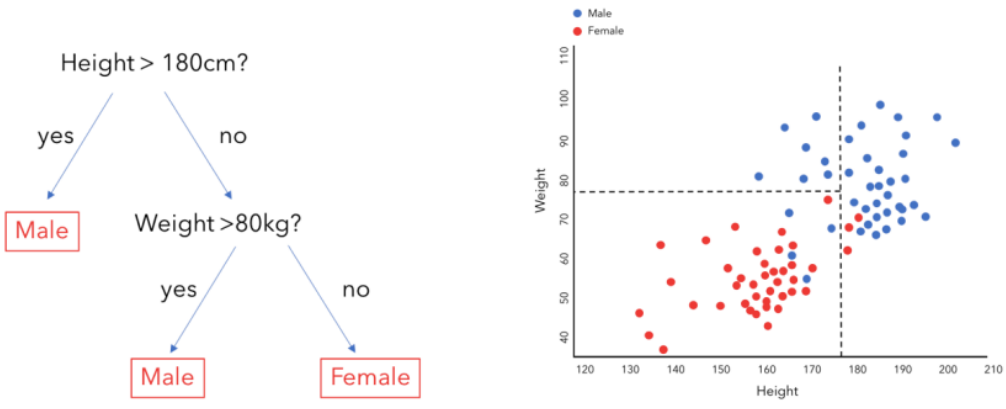

Figure 2.9: Working diagram of Decision Tree and representation in 2D plotted data

In the tree algorithm, several predictor variables (binary decisions) are used for the partition of data. This algorithm then classifies the input data into their classes. These terminals, roots, and decision nodes must be placed in order. Gini index and information gain are used in order to measure how well the algorithm placed these attributes of the tree (Breiman, et al., 1984). The sorting of attributes results in the reduction or gain in entropy. Information gain estimates the change in the entropy of individual attributes by using equation 10.

$$\mathrm{E(T)} = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{10}$$

After obtaining the entropy of the individual attribute using the above equation, the next step is to obtain the entropy of all target attributes (Breiman, et al., 1984)(equation 11).

$$\mathrm{E(T,X)} = \sum_{c \in X}^{c} P(c)E(c) \tag{11}$$

Then to obtain information gain Equation 12 subtracts the entropy of individual attributes from target attributes.

$$Gain(\mathrm{T,X}) = \mathrm{E(T)} - \mathrm{E(T,X)} \tag{12}$$

Information gain decides whether to take further splits. Information gain splits the attributes into different partitions i.e. attribute with the largest information gain will serve as a decision node. Besides, attributes with zero information are chosen as a leaf node(terminals), and all attributes having information greater than zero will further split (Kaushik, et al., 2018).

Gini Index is another method of splitting the decision trees. Gini index used Classification and Regression Tree (CART) to split trees. To achieve this task, Gini index calculates the sum of squared probabilities from each class and subtracts them from 1(Eq. 13). This step should be performed for all the attributes in the dataset.

$$Gini(\mathrm{N}) = \sum_{i=1}^{n} p^2 i \tag{13}$$

In the above equation, 'N' represents a dataset. After splitting the dataset into two parts, N1 and N2 with corresponding sizes M1 and M2. To calculate the Gini index equation 13 is used.

$$Gini_{split}(\mathrm{N}) = \frac{M1}{M} Gini(N1) + \frac{M2}{M} Gini(N2) \tag{14}$$

Decision trees are simple and require little data preparation. Decision trees also easy to interrupt because a user can visualize the trees. It is a powerful supervised learning model that can handle multi-output problems.

2.11 Technology Stack

2.11.1 Python and Jupyter Notebook

Python is a general-purpose programming language prevalent in data science and statistical analysis. Python is famous for its elegant syntax, objected-orientation support, and dynamic typing (De Smedt & Daelemans, 2012). It is also popular among the machine learning community, with a large number of machine learning libraries and packages written using python. Google, Netflix, Spotify, and many other large companies use python on daily basis to solve their routine problems (Perkel, 2018). Jupyter notebook is a local server-based application that runs on internet browsers. It allows users to visualize, create, share live coed, and explanatory text. Write code and visualize output straight away in the same environment.

2.11.2 Scikit-learn

Scikit-learn is a well-known open-source library, written in Python. Scikit-learn is also known as sklearn, is popular among data scientists. It offers a large number of utilities for machine learning problems, data-preparation, visualization, selection, and evaluation of different models (Pedregosa, et al., 2011). It offers a huge range of Application Programming Interfaces (API), including machine learning problems,

supervised learning, unsupervised learning, and deep learning. Sklearn reduces the coding time, helps the developer to focus on the logical aspects of the program than writing the code for everything from scratch. In this work, preprocessing of data and implementation of all supervised learning models, including SVM, KNN, Logistic Regression, Naïve Bayes, and Decision Trees, are achieved by using this library.

2.11.3 Keras

Keras is a high-level deep learning framework, written in Python (Chollet & others, 2015). It speeds up the experimentation process. Keras runs on top of other deep learning frameworks, e.g. Tensorflow, Theano, and PyTorch. Keras provides strong support for using Graphics Processing Units (GPUs). It can create any architecture of deep learning with just a few lines of code, including GANs, Autoencoders, Convolutional Neural Networks (CNNs), DNN.

2.11.4 Pandas

Panda is a famous and highly appreciated library in the data science community. It runs on top of the Python programming language. Pandas offer powerful functions for data manipulation. It can also read and write files from and to the operating system in different formats. It also offers a large number of functions, including reshaping of datasets, handling missing values, merging and joining, data filtration, insertion of data, deletion of data, creating copies of data, and splitting data into different groups.

3. Methodologies and Design

This chapter sheds some light on the design and methodologies that were used to accomplish the aim of the project, section 1.2. The implementation of the research approaches, section 1.3, is given below.

3.1 Data preparation

The ISCX-Bot-2014 dataset was published and maintained by the Canadian Institute of Cybersecurity (CIC). Dataset was available in the form of large .pcap files. CIC also provided a utility to extract the features from the dataset called CICFlowmeter-v4. All features were extracted by using this utility. Table 3.1 shows the extended features used in this work. The features are categorized into different groups based on flow, size, flags, time, and number of packets. All categorized features were dropped because there is no not using an IP address or port numbers. However, (Yin, et al., 2018)authors have used these categorical features in their work. To enhance the robustness of our IDS flags are also used in this research. ISCX-Bot-2014 was already divided into the training set and test set by the provider. This dataset has information about several types of botnet attacks, including VIRUT, Zeus bot, Neris, Menti, and Murlo. In this research VIRUT (an HTTP-based botnet) botnet is used and the reason for choosing VIRUT is that HTTP attacks are commons these days. VIRUT's samples, 42716, were extracted from the ISCX-Botnet-2014 dataset and merged with 127745 benign samples. So, the resulted dataset consisted of 61 features and 170461 entries.

For preparing the dataset for data analysis, several steps of treatment were involved some of them are as follows:

- WireShark application is used for the extraction of features and data from the .pcap file and converting the data into a .csv file.
- ISCX-Botnet-2014 also needs labeling of each entry on the basis of information provided over the website. Every botnet is associated with an IP address. VIRUT's IP address was 147.32.84.160, features were extracted related to this IP.
- VIRUT's features were labeled as 1 and normal or benign entries labeled as 0.

- High and low skewed values should be removed, to achieve this goal all Not a Number (NaN) and Infinites values were removed from the dataset to suppress outliers.
- Any column with zero standard deviation was removed. Features that are selected for this work in shown in Table 3.1.
- Categorical features were not considered and dropped, such as IP address and port number.
- All features were scaled in the range of [0,1] because ReLU is used as an activation function in GAN and the range of ReLU activation function is also in between [0,1].

After data preparation, the resulting dataset contained 170461 entries with 61 features.

3.2 Dataset Splits

To maintain the effectiveness and robustness of a supervised learning model, the data science community had introduced a technique that not only assists to check the validity of data but also helps to control overfitting. The technique is to split the dataset into two parts, training set, and test set. Traditionally, a classifier takes training data and split it for validation, this technique resulted in measuring the model's training loss and hyperparameter tuning. This validation method is not a good indicator to check the performance of the classifier, because after some time the classifier starts being biased towards the training data which results in overfitting. Train-test split solves bias problem and results in a good evaluation matrix. However, the splitting of the dataset depends on several factors, including the size of a dataset, which classifier is being used, and the hyperparameter of a selected classifier. In this work, 70% of the dataset was used as a training set and 30% as a test set.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('./datasets/ISCX_Botnet.csv')
# selection of all columns except label class with label = 0
X = dataset.iloc[:, :-1].values
# selection of only label class with label = 1
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
```

Figure 3.1: Code used to split the dataset into a training set and test set

3.3 Support Vector Machines Approach

Support Vector Machines are famous supervised learning methods that build to solve classification and regression problems. The objective of support vector machines is to find a hyperplane that can distinctly separate the input data point. More details on support vector machines can be found in section 2.10.1.

Implementation of the SVC learning algorithm is straightforward and easy. Support vector machines offer a wide variety of hyperparameter tuning while initializing the classifier. Basically, the implementation of this classifier is based on libsvm. The parameter *kernel* offers a different type of kernels (default is rbf), in this work linear kernel is used and the objective of the linear kernel is that it returns the best fit of input data. The regularisation parameter *C*, is a penalty term used to control the strength of the penalty. The regularisation parameter must stay positive (default value is 1) and the strength of regularisation is inversely proportional to *C*. The value of *random_state* hyperparameter was set to 0 (default is none). The *random_state* ensures the random shuffling of probability estimation. Code used to run SVM is available in appendix 7.3.

3.4 k-Nearest Neighbors Approach

The model is based on the Sigmoid function that predicts the output class within the range of 0 and 1. This classifier is a parametric classification model used to solve binary classification problems. Section 2.10.2 provides more details of this approach.

This approach offers a limited number of hyperparameter tuning. The first parameter was the selection of *n_neighbors* which represents the number of nearest neighbors of

each data point. The value of *n_neighbors* was set to 6 (default 5). Brute force (a data structure used to compute the distance between pairs of samples) does not depend on the value of *k* but if a user using Ball Tree (a type of algorithm for this model) than Ball Tree's query time may increase and slow down the overall performance of this model (Pedregosa, et al., 2011). The second parameter that is used in this work is *metric* which is a distance metric of this model. To select the Euclidean distance as a distance value, it is a must to select *metric* as 'minkowski' and with p=2. The code used to run this model can be found in appendix 7.4.

3.5 Logistic Regression Approach

The model is based on the Sigmoid function that predicts the output class within the range of 0 and 1. This classifier is a parametric classification model used to solve binary classification problems. Section 2.10.3 provides more details of this approach.

Hyperparameter tuning of the logistic regression approach consists of setting a large number of parameters. The *penalty* hyperparameter demands which type of optimization solver that the user wants to use with the default value 'l2'. In this work, a Stochastic Gradient decent (sag) optimization solver is used which is faster for large datasets with both large features and samples. The value of *random_state* hyperparameter was set to 0 (default is none). The *random_state* ensures the random shuffling of probability estimation. Code used to run logistic regression model is available in appendix 7.5. The *max_iter* is the number of iteration that a model will take to converge by using solvers, the value of this hyperparameter was set to 400 (default is 100).

3.6 Naïve Bayes Approach

Naïve Bayes is a machine learning approach that is based on Bayes' theorem. Implementation of the Naïve Bayes approach can be achieved by using several algorithms, including Nominal Naïve Bayes, Complement Naïve Bayes, Bernoulli Naïve Bayes, and Categorical Naïve Bayes. More information is available in section 2.10.4. This research is limited to using the Gaussian Naïve Bayes algorithm which is famous

among all other algorithms. Gaussian Naïve Bayes approach offers only two hyperparameters to tune but no parameter was tunned to run this algorithm. Default values of all parameters were used in this work and code can be found in appendix 7.6.

3.7 Decision Tree Approach

Decision trees are a non-parametric type of machine learning models. The decision tree model measures the quality of split by using information gain or Gini index, more information can be found in section 2.10.5.

Implementation of this classifier involves tuning of several hyperparameters, some are not essential to use due to the nature of this work. The *criterion* hyperparameter takes two types of input, *gini* or *entropy*. The value was set as *entropy* because information gain is used to measure the quality and positions of splits. The parameter *random_state* controls the randomness of estimation by selecting the maximum number of features at each split. The code used to run this model can be found in appendix 7.7.

3.8 GAN Approach

Generative adversarial networks work by playing a zero-sum game where the generator model and discriminator model are adversarial to each other. The architecture and training process of GAN is in section 2.9.

There are many challenges involve in the training of this model because a large number of choices are involved, including but not limited to: the architecture of the two models (the generator and discriminator), optimizer, selection of learning rate, number of epochs, activation functions, and selection of dropout. Therefore, these choices largely affect the performance of GAN. In addition, the training of GAN can also face a stable cycle where both models stop improving and repeating the same tricks (Zhang, et al., 2019). The discriminator may overpower the generator model which results in both models neither learn nor improve. Another major problem that encounters most of the time while training is mode collapse. Mode collapse occurred when the generator network starts learning only a small subset of input data and ignoring other variations of data.

In this work, to overcome these difficulties several hyperparameters were tuned to achieve realistic GAN-bots. The hyperparameter *noise_dim* was set to 32, which is used to generate random noise that is given to the generator to produce random data. As mentioned above, the generator model is an unsupervised learning model that starts learning the distribution of data from random noise. The *learning_rate* refers to the size of steps that the optimizer will take after each iteration until it converged. The larger the size of *learning_rate*, the model's optimization time decreases and vice versa. The *learning_rate* was set 5e-4(0.0005) and the optimizer algorithm selected for this work is *Adam*. Adam is a famous stochastic optimization method based on adaptive estimation. Adam is a memory-efficient and straightforward approach and well suited for a large dataset (Kingma & Ba, 2014). While the training of GAN, the generator and discriminator models are evaluated by the *loss* function which is a difference between the input data and estimated value. To measure the *loss*, the *binary_crossentropy* approach is used. The parameter *epoch* is a traversal of the whole dataset. Traversal of the dataset depends on the number of epoch selected for traversal, 301 epochs were chosen. Traversal of the whole dataset requires more memory and time-consuming. Therefore, researchers introduced the *batch_size* (value was set to 32) which is the traversal of the dataset in mini-batches that enhance the training time.

The generator model is a 5 layered network with 3 hidden layers. The summary of the generator model is shown in figure 3.2. All layers of the generator model were dense with ReLU as an activation function. On the other hand, the discriminator model is 8 layered network, shown in figure 3.3. Two dropout layers, which is a regularization approach to reduce overfitting and reduce generalization error, were used in the discriminator model.

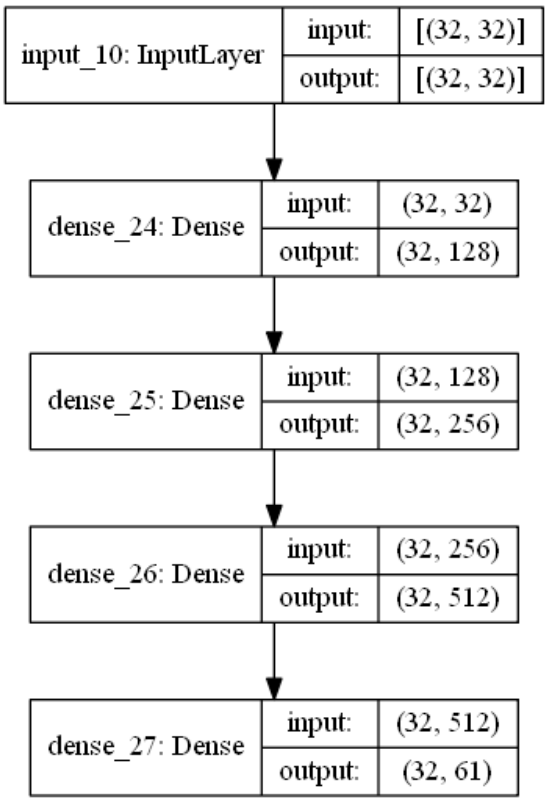

Figure 3.2: Architecture of the Generator Model

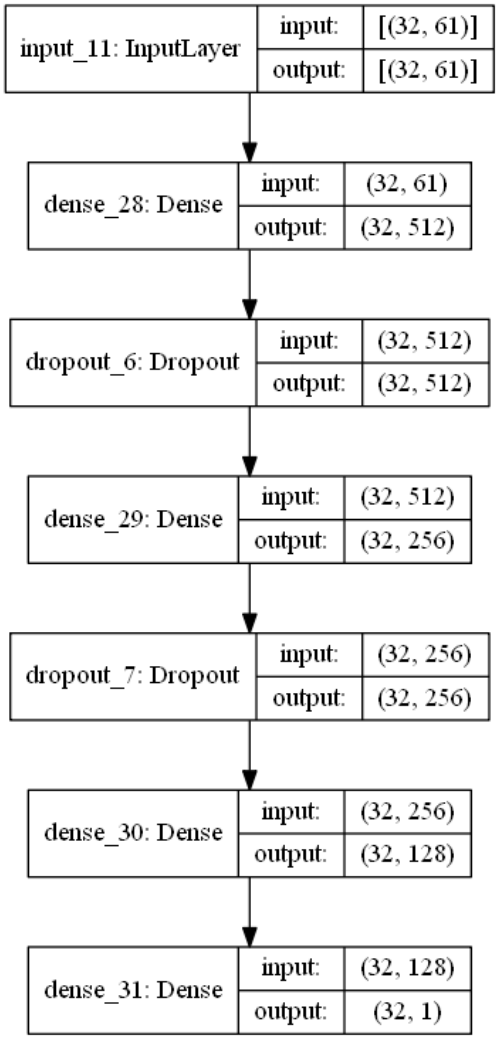

Figure 3.3: Architecture of the Discriminator Model

3.9 Evaluation Metrics

In order to select the best model among five trained models, three key matrixes were identified: Accuracy, Precision, and Recall. Accuracy metric alone may not evaluate the sustainability of the model because of the accuracy paradox. Accuracy paradox refers to the situation where most of the time, the model is predicting a specific class is correct because equal weights are assigned to both false positive (FP) and false

negative (FN). Therefore, two relative class metrics were considered: precision and recall.

Accuracy: The accuracy of a classifier is defined as the number of accurate predictions (i.e. True Positive (TP) and True Negative(TN)) out of the total number of predictions

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Precision: Precision is the exactness of the model. Precision measures how many predictions were correct out of all positive predictions.

$$\text{Precision} = TP / (TP + FP)$$

Recall: Recall is the ability of a model to find all similar samples. Recall can be defined as the comprehensiveness of the model.

$$\text{Recall} = TP / (TP + FN)$$

Evasion Rate: The evasion rate (ER) measures the number of malicious samples that were evaded while the attack on a machine learning model.

$$\text{ER} = \text{malicious evaded samples} / \text{total malicious samples}$$

4. Results

4.1 Traning and evaluation of classifiers

Five different machine learning classifiers were trained, including SVM, Decision Tree, Naïve Bayes, Logistic Regression, and k-Nearest Neighbors. In this work, the best classifier will be served as the Intrusion Detector so it is crucial to chose a classifier with high accuracy, precision, and recall rate. Therefore, a separate test set was used to calculate their performances on unseen traffic(normal and botnet traffic). These are the results of phase number 2 of the research approach(Section 1.3).

Table 4.1 shows that the DT model is outperforming the SVM, KNN, LR, and NB. In this work, DT will be served as a machine learning-based IDS.

Model	**Accuracy**	**Precision**	**Recall**
SVM	95.89%	95.69%	87.47%
KNN	99.60%	99.19%	99.25%
LR	95.15%	95.00%	85.20%
NB	58.42%	37.38%	97.03%
DT	**99.91%**	**99.83%**	**99.82%**

Table 4.1: Accuracy, Precision, and Recall of SVM, KNN, LR, NB, and DT on ISCX-Bot-2014 dataset.

To gain more insight, figure 4.1compares performances of all classifiers by using a bar chart. It can be seen that KNN and DT both showed good results as compared to other classifiers but overall scores of DT are approximately more than 99.80% in all evaluation matrixes.

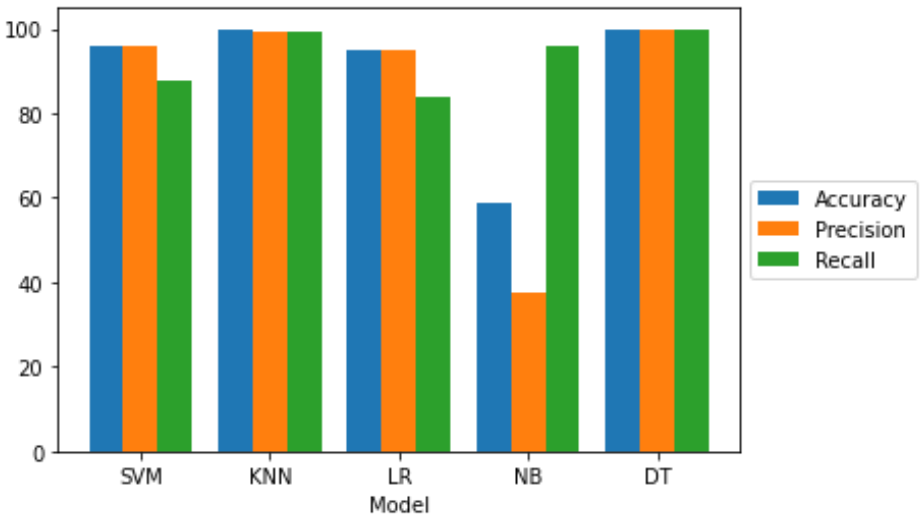

Figure 4.1: Comparison of all classifiers on the basis of Accuracy, Precision, and Recall

4.2 GAN performance

Generative adversarial networks are difficult to train due to tunning of a large number of hyperparameters and the selection of architecture. GAN was used in this work for the generation of synthetic data (GAN-bots). GAN consists of two networks the generator and discriminator. The training evaluation of GAN depends on the accuracy and loss of both models.

Figure 4.2 shows the training history of GAN. The figure consists of two subplots. In the top subplot, d_real (blue) represents the loss of discriminator for real data after each epoch. Whereas d_fake (orange) is the loss of discriminator for fake data. The generator loss is represented by a green line. It can be seen that in the early run the losses were eccentric and start stabilizing between epoch 15 and 90. It is expected that the GAN had generated plausible GAN-bots in between of these epochs. The stable loss of discriminator for real and fake data should sit around 0.5 to 0.7. The loss of the generator perhaps varies from 0.5 to 2.1.

The bottom subplot shows the discriminator's classification accuracy for real (blue) and fake (orange) data. These accuracies of stable GAN are expected to fluctuate between 50% to 80%. It can be seen that the classification accuracy of the discriminator for real and fake data was stable in between 15 to 90 epochs. However,

the classification accuracy of a discriminator for identifying fake data remains high after 90 epochs. Therefore, this implies that the discriminator is easily identifying fake samples because the consistency of the generator is poor in generating new samples. This type of failure refers to a mode collapse. Mode collapse can be defined as a scenario in which the generator learns mapping of a large number of inputs (z) to the same output. However, GAN generated 5000 plausible samples (GAN-bots) between 15 to 90 epochs which is used as an adversarial evasion attack on the trained classifier.

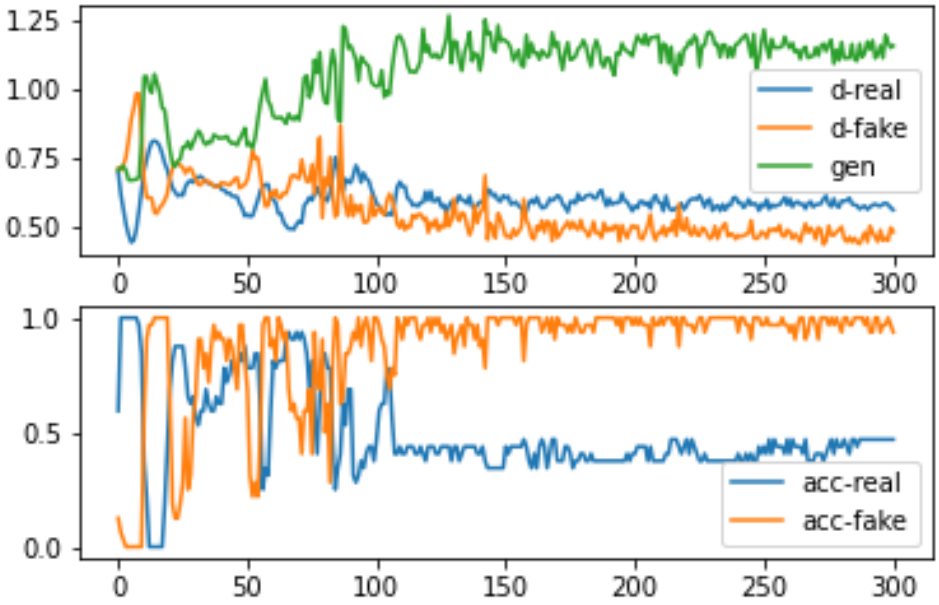

Figure 4.2: Training history of GAN represented using line plots.

4.3 Evaluation of evasion attack and countermeasures

GAN was employed to generate 5000 GAN-bots for the evasion attack on the trained DT classifier. Before the evasion attack, 5000 benign samples were randomly selected from the original dataset and augmented with the GAN-bots making it a new dataset of 10000 samples. The evasion attack can be launched by using only GAN-bots but benign samples were selected to analyze the behavior of the classifier on both types of input samples. The augmented dataset was used to penetrate the DT classifier and the results can be seen in table 4.1.

Accuracy	Precision	Recall
50.20%	95.45%	0.42%

Table 4.1: Evaluation scores of Decision tree classifier after evasion attack

To understand these results a confusion matrix was obtained. Figure 4.3 shows the number of samples that were identified by the DT classifier. The upper left section indicates the TN matrix which shows that out of the 5000 bening samples the classifier successfully classified 4999 of them as benign. The DT classifier only misclassified 1 sample as malign which can be seen in the upper right section (FP) of figure 4.3. In every classification problem of machine learning, TN (lower right matrix) is the most hazardous matrix among others. It can be seen that the DT only classified 21 samples as malicious samples out of 5000 which is a TP rate of the classifier. The DT identified 4979 GAN-bots samples as benign.

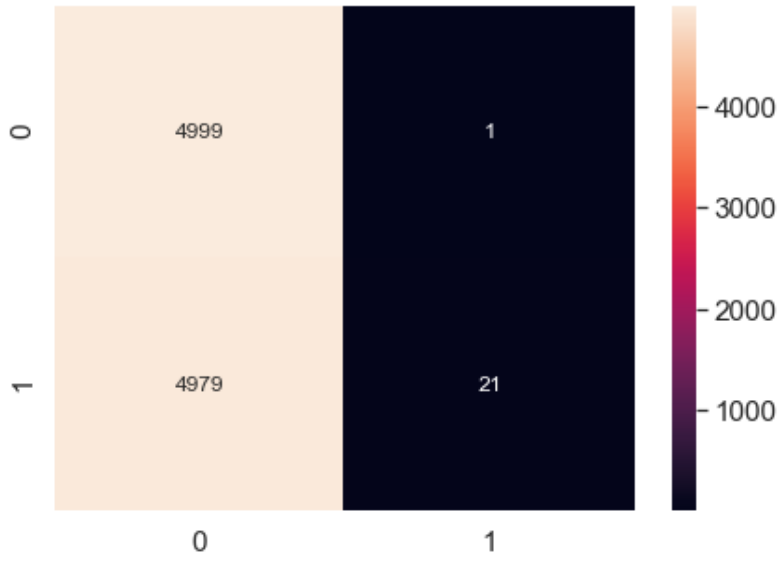

Figure 4.3: Confusion matrix

During the evasion attack on the trained machine learning-based IDS, 99.58% of GAN-bots were evaded. Therefore, **99.58% is the evasion rate recorded while an attack** on the classifier making it clear that the best classifier is prone to adversarial evasion attacks. Evasion rate was calculated using eq. 15.

$$Evasion\ rate = \frac{evaded\ samples}{total\ number\ of\ malicious\ samples} \tag{15}$$

This type of evasion attack identifies the security concerns related to ML-based IDS. These types of attacks giving advantages to attackers to compromise system security. The integral part of this research work to mitigate the effect of GAN-based adversarial attacks on ML-based IDS.

To address this problem, evaded samples were collected and augmented with the original ISCX-Bot-2014 dataset. This newly augmented dataset was used to retrain the DT classifier and strengthen a classifier against GAN-based evasion attacks. After augmentation, the dimension of the ISCX-Bot-2014 dataset evolved to 175440 entries and 61 features. For retraining purposes, the dataset was again split into a 70:30 ratio for training and testing. After training, the resulting DT model with an accuracy of 99.91%, precision rate of 99.83%, and recall rate of 99.85%.

To test the strength of retrained classifier over the adversarial attack, GAN was again employed for the generation of new 5000 GAN-bots. A new dataset of 10000 was formulated by combining newly generated GAN-bots and randomly selected 5000 benign samples. The DT classifier was penetrated and results can be seen in table 4.2.

Accuracy	**Precision**	**Recall**
97.16%	99.97%	94.34%

Table 4.2: Accuracy, recall, and precision of Decision tree classifier

It can be seen in figure 4.4 that the TN and FP rate of the decision tree model remained the same as above but there is an enormous change in the FN and TP metrics. The FN rate was reduced from 4979 to 283, meaning the classifier identified 283 GAN-bots as benign out of 5000. Besides, the TP rate escalated from 21 to 4717 which represents that the classifier correctly classified 4717 GAN-bots as bots. **So, the evasion rate was minimized from 99.58% to 5.66%.** Our methodology has produced state-of-the-art results in mitigating the effects of adversarial attacks on machine learning-based Intrusion Detection Systems.

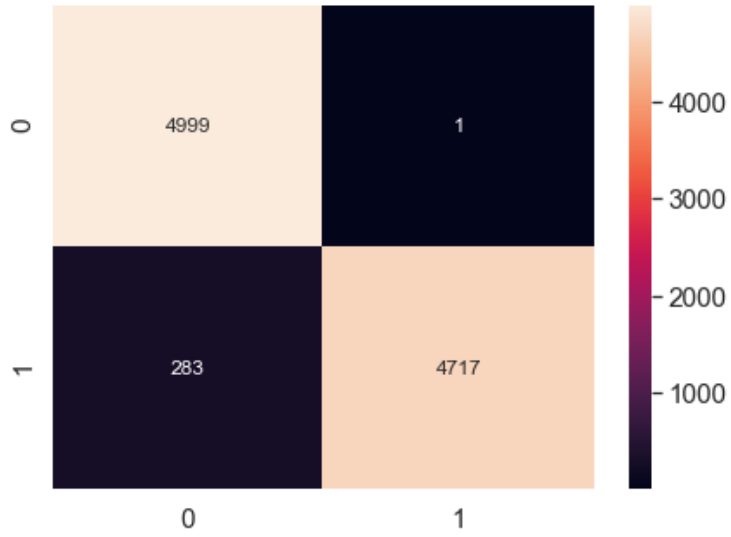

Figure 4.4: Confusion matrix acquire after evasion attack

5. Conclusion

5.1 Critical evaluation

The prime objective of this research work is to strengthen the detection ability of machine learning-based IDS against adversarial evasion attacks. Due to the magnificent change in technology, cybercriminals analyze new methods to compromise the security of systems. Botnet attacks have evolved and using sophisticated methods e.g., evasion attacks for compromising systems. To address this problem, the dataset ISCX-Bot-2014 was used which holds the information of botnet attacks. This dataset was used to train five different machine learning-based IDS for the selection of best among them on the basis of evaluation matrices. The GAN was employed for the generation of GAN-bots that were similar to the original bots in our dataset for evasion attack. GAN-bots were used to attack the machine learning model and 99.58% of GAN-bots were classified as benign samples by the model which indeed a critical issue considering the security of models.

However, this work proposed a technique to tackle the problem of evasion attacks. The samples that were evaded during the attack was used to retrain the model. GAN was considered again for the generation of new GAN-bots to relaunch the evasion attack on the newly trained classifier. A newly trained classifier was penetrated with GAN-bots and the evasion rate was improved from 99.58% to 5.66%. Our proposed approach ensures that the machine learning-based IDS aware of adversarial evasion attacks.

5.2 Future work

The scope of the research was limited to the mitigation of adversarial attacks and their countermeasures. Future work would be the generation of limited categories of GAN-bots that could help the classifier in detecting evasion attacks. It could be achieved by using active learning. In addition, Deep Neural Networks could also be adopted for performance analysis.

6. References

Acarali, D., Rajarajan, M., Komninos, N. & Herwono, I., 2016. Survey of approaches and features for the identification of HTTP-based botnet traffic. *Journal of Network and Computer Applications,* Volume 76, p. 1–15.

Alauthman, M. et al., 2020. An efficient reinforcement learning-based Botnet detection approach. *Journal of Network and Computer Applications,* Volume 150, p. 102479.

Alkasassbeh, M. & Almseidin, M., 2018. Machine learning methods for network intrusion detection. *arXiv preprint arXiv:1809.02610.*

Aviv, A. J. & Haeberlen, A., 2011. Challenges in experimenting with botnet detection systems.

Bamakan, S. M. H., Wang, H., Yingjie, T. & Shi, Y., 2016. An effective intrusion detection framework based on MCLP/SVM optimized by time-varying chaos particle swarm optimization. *Neurocomputing,* Volume 199, p. 90–102.

Barlow, H. B., 1989. Unsupervised learning. *Neural computation,* Volume 1, p. 295–311.

Beigi, E. B., Jazi, H. H., Stakhanova, N. & Ghorbani, A. A., 2014. *Towards effective feature selection in machine learning-based botnet detection approaches.* s.l., s.n., p. 247–255.

Bell, J., 2020. *Machine learning: hands-on for developers and technical professionals.* s.l.:John Wiley & Sons.

Biggio, B. et al., 2013. *Evasion attacks against machine learning at test time.* s.l., s.n., p. 387–402.

Biggio, B., Fumera, G. & Roli, F., 2010. Multiple classifier systems for robust classifier design in adversarial environments. *International Journal of Machine Learning and Cybernetics,* Volume 1, p. 27–41.

Biggio, B., Fumera, G. & Roli, F., 2013. Security evaluation of pattern classifiers under attack. *IEEE transactions on knowledge and data engineering,* Volume 26, p. 984–996.

Breiman, L., Friedman, J., Stone, C. J. & Olshen, R. A., 1984. *Classification and regression trees.* s.l.:CRC press.

Brownlee, J., 2019. A gentle introduction to generative adversarial networks (GANs). *Retrieved June,* Volume 17, p. 2019.

Buczak, A. L. & Guven, E., 2015. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials,* Volume 18, p. 1153–1176.

Butun, I., Morgera, S. D. & Sankar, R., 2013. A survey of intrusion detection systems in wireless sensor networks. *IEEE communications surveys & tutorials,* Volume 16, p. 266–282.

Chollet, F. & others, 2015. *Keras.* s.l.:s.n.

da Costa, K. A. P. et al., 2019. Internet of Things: A survey on machine learning-based intrusion detection approaches. *Computer Networks,* Volume 151, p. 147–157.

Dalvi, N., Domingos, P., Sanghai, S. & Verma, D., 2004. *Adversarial classification.* s.l., s.n., p. 99–108.

De Smedt, T. & Daelemans, W., 2012. Pattern for python. *The Journal of Machine Learning Research,* Volume 13, p. 2063–2067.

Dua, S. & Du, X., 2016. *Data mining and machine learning in cybersecurity.* s.l.:CRC press.

Elhag, S. et al., 2015. On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems. *Expert Systems with Applications,* Volume 42, p. 193–202.

Engelmann, J. & Lessmann, S., 2020. Conditional Wasserstein GAN-based Oversampling of Tabular Data for Imbalanced Learning. *arXiv preprint arXiv:2008.09202.*

Farina, P., Cambiaso, E., Papaleo, G. & Aiello, M., 2016. Are mobile botnets a possible threat? The case of SlowBot Net. *Computers & Security,* Volume 58, p. 268–283.

Farnaaz, N. & Jabbar, M. A., 2016. Random forest modeling for network intrusion detection system. *Procedia Computer Science,* Volume 89, p. 213–217.

Garcia, S., 2013. *Malware capture facility project, cvut university.* s.l.:s.n.

Goldberger, J., Hinton, G. E., Roweis, S. & Salakhutdinov, R. R., 2004. Neighbourhood components analysis. *Advances in neural information processing systems,* Volume 17, p. 513–520.

Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y., 2016. *Deep learning.* s.l.:MIT press Cambridge.

Goodfellow, I. J., Shlens, J. & Szegedy, C., 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572.*

Gu, G., 2008. *Detecting botnet commandandcontrolchannelsinnetworktraffic.* s.l., s.n., p. 269–286.

Gupta, A. et al., 2018. *Social gan: Socially acceptable trajectories with generative adversarial networks.* s.l., s.n., p. 2255–2264.

Hamid, Y., Sugumaran, M. & Balasaraswathi, V. R., 2016. Ids using machine learning-current state of art and future directions. *Current Journal of Applied Science and Technology,* p. 1–22.

Huang, L. et al., 2011. Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence.

Jabbar, A., Li, X. & Omar, B., 2020. A Survey on Generative Adversarial Networks: Variants, Applications, and Training. *arXiv preprint arXiv:2006.05132.*

James, G., Witten, D., Hastie, T. & Tibshirani, R., 2013. *An introduction to statistical learning.* s.l.:Springer.

KarsligEl, M. E. et al., 2017. *Network intrusion detection using machine learning anomaly detection algorithms.* s.l., s.n., p. 1–4.

Kaushik, S. et al., 2018. *Evaluating frequent-set mining approaches in machine-learning problems with several attributes: a case study in healthcare.* s.l., s.n., p. 244–258.

Khraisat, A., Gondal, I. & Vamplew, P., 2018. *An anomaly intrusion detection system using C5 decision tree classifier.* s.l., s.n., p. 149–155.

Khraisat, A., Gondal, I., Vamplew, P. & Kamruzzaman, J., 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity,* Volume 2, p. 20.

Kingma, D. P. & Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Kreibich, C. & Crowcroft, J., 2004. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review,* Volume 34, p. 51–56.

Kshetri, N. & Voas, J., 2017. Hacking power grids: A current problem. *Computer,* Volume 50, p. 91–95.

Kurakin, A. et al., 2018. Adversarial attacks and defences competition. In: *The NIPS'17 Competition: Building Intelligent Systems.* s.l.:Springer, p. 195–231.

LeCun, Y., Bengio, Y. & Hinton, G., 2015. Deep learning. *nature,* Volume 521, p. 436–444.

Lin, W.-C., Ke, S.-W. & Tsai, C.-F., 2015. CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. *Knowledge-based systems,* Volume 78, p. 13–21.

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

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

Papernot, N. et al., 2016. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697,* Volume 1, p. 3.

Papernot, N. et al., 2017. *Practical black-box attacks against machine learning.* s.l., s.n., p. 506–519.

Papernot, N. et al., 2016. *The limitations of deep learning in adversarial settings.* s.l., s.n., p. 372–387.

Pedregosa, F. et al., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* Volume 12, p. 2825–2830.

Perkel, J. M., 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature,* Volume 563, p. 145–147.

Santana, D., Suthaharan, S. & Mohanty, S., 2018. What we learn from learning-Understanding capabilities and limitations of machine learning in botnet attacks. *arXiv preprint arXiv:1805.01333.*

Schölkopf, B., Smola, A. J., Williamson, R. C. & Bartlett, P. L., 2000. New support vector algorithms. *Neural computation,* Volume 12, p. 1207–1245.

Shiravi, A., Shiravi, H., Tavallaee, M. & Ghorbani, A. A., 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security,* Volume 31, p. 357–374.

Tavallaee, M., Stakhanova, N. & Ghorbani, A. A., 2010. Toward credible evaluation of anomaly-based intrusion-detection methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* Volume 40, p. 516–524.

Usama, M. et al., 2019. *Generative Adversarial Networks for Launching and Thwarting Adversarial Attacks on Network Intrusion Detection Systems.* s.l., s.n., p. 78–83.

Wang, H., Gu, J. & Wang, S., 2017. An effective intrusion detection framework based on SVM with feature augmentation. *Knowledge-Based Systems,* Volume 136, p. 130–139.

Wang, Q. et al., 2017. *Adversary resistant deep neural networks with an application to malware detection.* s.l., s.n., p. 1145–1153.

Wermter, S. et al., 2014. *Artificial Neural Networks and Machine Learning–ICANN 2014: 24th International Conference on Artificial Neural Networks, Hamburg, Germany, September 15-19, 2014, Proceedings.* s.l.:Springer.

Xiao, L. et al., 2018. IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?. *IEEE Signal Processing Magazine,* Volume 35, p. 41–49.

Yin, C. et al., 2018. *An enhancing framework for botnet detection using generative adversarial networks.* s.l., s.n., p. 228–234.

Yu, S., Tian, Y., Guo, S. & Wu, D. O., 2013. Can we beat DDoS attacks in clouds?. *IEEE Transactions on Parallel and Distributed Systems,* Volume 25, p. 2245–2254.

Yu, S. et al., 2011. Discriminating DDoS attacks from flash crowds using flow correlation coefficient. *IEEE transactions on parallel and distributed systems,* Volume 23, p. 1073–1080.

Zhang, H., 2004. *IThe Optimality of Naive Bayes,".* s.l., s.n., p. 1–6.

Zhang, H., Sindagi, V. & Patel, V. M., 2019. Image de-raining using a conditional generative adversarial network. *IEEE transactions on circuits and systems for video technology.*

Zhao, D. et al., 2013. Botnet detection based on traffic behavior analysis and flow intervals. *computers & security,* Volume 39, p. 2–16.

7. Appendix

7.1 Data snapshots

Flow Dur	Total Fw	Total Bw	Total Len	Total Len	Fwd Pack	Fwd Pack	Fwd Pack	Fwd Pack	Bwd Pack	Bwd Pack	Bwd Pack	Bwd Pack	Flow Byte	Flow Pac	Flow IAT	Flow IAT	Flow IAT	Fwd IAT T	Fwd IAT I	Fwd IAT I	Fwd IAT I	Bwd IAT T	Bwd IAT I	Bwd IAT S
0.51504	6.72E-05	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-07	0.04293	4.20E-07	0.0009	0.51504	0.04293	4.20E-07	0.0009	0	0	0
0.40083	7.84E-05	0	0	0	0	0	0	0	0	0	0	0	0	1.50E-07	0.02864	7.30E-07	6.00E-08	0.40083	0.02864	7.30E-07	6.00E-08	0	0	0
0.99997	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50007	1.67E-06	0.50001	0.99997	0.50007	1.67E-06	0.50001	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.99998	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50003	0.99998	0.50008	1.67E-06	0.50003	0	0	0
0.86521	2.24E-05	0	0	0	0	0	0	0	0	0	0	0	0	2.00E-08	0.21634	2.89E-06	9.00E-08	0.86521	0.21634	2.89E-06	9.00E-08	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.50003	5.60E-06	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.50012	1.67E-06	0.50012	0.50003	0.50012	1.67E-06	0.50012	0	0	0
0.50003	5.60E-06	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.50012	1.67E-06	0.50012	0.50003	0.50012	1.67E-06	0.50012	0	0	0
0.49996	5.60E-06	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.50005	1.67E-06	0.50005	0.49996	0.50005	1.67E-06	0.50005	0	0	0
0.99997	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50007	1.67E-06	0.50002	0.99997	0.50007	1.67E-06	0.50002	0	0	0
0.71029	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.35521	1.67E-06	0.2103	0.71029	0.35521	1.67E-06	0.2103	0	0	0
0.99998	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50004	0.99998	0.50008	1.67E-06	0.50004	0	0	0
0.99998	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99998	0.50008	1.67E-06	0.50005	0	0	0
0.99998	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99998	0.50008	1.67E-06	0.50005	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50006	0.99999	0.50008	1.67E-06	0.50006	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50004	0.99999	0.50008	1.67E-06	0.50004	0	0	0
1	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50009	1.67E-06	0.50005	1	0.50009	1.67E-06	0.50005	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.50003	5.60E-06	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.50012	1.67E-06	0.50012	0.50003	0.50012	1.67E-06	0.50012	0	0	0
0.99997	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50007	1.67E-06	0.50005	0.99997	0.50007	1.67E-06	0.50005	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.99999	1.68E-05	0	0	0	0	0	0	0	0	0	0	0	0	1.00E-08	0.33339	1.67E-06	0.16705	0.99999	0.33339	1.67E-06	0.16705	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50003	0.99999	0.50008	1.67E-06	0.50003	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50005	0.99999	0.50008	1.67E-06	0.50005	0	0	0
0.99999	1.12E-05	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50008	1.67E-06	0.50007	0.99999	0.50008	1.67E-06	0.50007	0	0	0

Bwd IAT I	Bwd IAT I	Fwd PSH	Fwd Hea	Bwd Hea	Fwd Pack	Bwd Pack	Packet Le	Packet Le	Packet Le	Packet Le	Packet Le	FIN Flag	SYN Flag	RST Flag	PSH Flag	ACK Flag	Down/Up	Average I	Fwd Segr	Bwd Segr	Bwd Byte	Bwd Pack	Bwd Bulk	Subflow
0	0	0	0	0	1.10E-07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.60E-07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	2.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1.00E-08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Subflow	Subflow	FWD Init	Bwd Init	Fwd Act D	Fwd Seg	Idle Mea	Idle Std	Idle Max	Idle Min	Label
0	0	0	0	0	0	3.00E-08	2.00E-08	5.00E-08	1.00E-08	0
0	0	0	0	0	0	0.96552	2.00E-08	0.96552	0.96552	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.96552	6.00E-08	0.96552	0.96552	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	4.00E-08	0	4.00E-08	4.00E-08	0
0	0	0	0	0	0	4.00E-08	0	4.00E-08	4.00E-08	0
0	0	0	0	0	0	4.00E-08	0	4.00E-08	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	2.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	4.00E-08	0	4.00E-08	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.64368	0.78835	0.96552	3.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0
0	0	0	0	0	0	0.48276	0.96552	0.96552	4.00E-08	0

7.2 Imported libraries

The header.py file holds all import files so keep the code clean. Every new file has to import header.py to use these utilities.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import dataset_loader
import seaborn as sns
import os
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras import Model
from tensorflow.keras.optimizers import Adam
import math
```

7.3 Dataset loader

```
import header
from header import *


class DatasetLoader:
    def __init__(self):
        return None
    def __new__(self, path):
        # Laod scv from the path
        dataset = pd.read_csv(path)
        print(dataset['Label'].value_counts())
        # selection of all columns except label class with label = 0
        X = dataset.iloc[:, :-1].values
        # selection of only label class with label = 1
        y = dataset.iloc[:, -1].values
        # spliting of data into X_train, X_test, y_train and y_test
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
        return (X_train, X_test, y_train, y_test)
```

7.4 Code for SVM

Runs the decision tree model defined in section SVM model described in section 3.3 and prints accuracy and confusion matrix.

```
import header
from header import *

class SVM:
    def __init__(self, X_train, X_test, y_train, y_test):
        self.model = False
        self.evaluate_acc = False
        self.evaluate_cm = False
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.classifier()
    def classifier(self):
        print('Running model')
        # XGBClassifier initialisation
        self.model = SVC(kernel = 'linear', random_state = 0)
        # Training of model
        self.model.fit(self.X_train, self.y_train)
        # Evaluation the model
        self.evaluate_acc = self.model_evaluate(self.model)
        self.evaluate_cm = self.model_evaluate(self.model)
    def model_evaluate(self, model):
        # Make predictions using test set
        y_pred = model.predict(self.X_test)
        # Calculating accuracy
        accuracy = accuracy_score(self.y_test, y_pred)
        # Building confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        return (accuracy, cm)

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = dataset_loader.DatasetLoader('./assets/ISCX_Botnet.csv')
    # Run model and print accuracy score and cm
    svm = SVM(X_train, X_test, y_train, y_test)
    print('Model accuracy score: {}'.format(svm.evaluate_acc[0]*100.00))
    print(svm.evaluate_cm[1])
```

7.5 Code for knn

Represents the k-Nearest Neighbors model defined in section 3.4.

```
import header
from header import *

class KNN:
    def __init__(self, X_train, X_test, y_train, y_test):
        self.model = False
        self.evaluate_acc = False
        self.evaluate_cm = False
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.classifier()
    def classifier(self):
        print('Running model')
        # XGBClassifier initialisation
        self.model = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
        # Training of model
        self.model.fit(self.X_train, self.y_train)
        # Evaluation the model
        self.evaluate_acc = self.model_evaluate(self.model)
        self.evaluate_cm = self.model_evaluate(self.model)
    def model_evaluate(self, model):
        # Make predictions using test set
        y_pred = model.predict(self.X_test)
        # Calculating accuracy
        accuracy = accuracy_score(self.y_test, y_pred)
        # Building confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        return (accuracy, cm)

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = dataset_loader.DatasetLoader('./assets/ISCX_Botnet(Prep).csv')
    # Run model and print accuracy score and cm
    knn = KNN(X_train, X_test, y_train, y_test)
    print('Model accuracy score: {}'.format(knn.evaluate_acc[0]*100.00))
    print(knn.evaluate_cm[1])
```

7.6 Code for logistic regression

This file responsible for running the Logistic Regression model in section 3.5 and prints the accuracy and confusion matrix.

```
import header
from header import *

class LogisticRegression:
    def __init__(self, X_train, X_test, y_train, y_test):
        self.model = False
        self.evaluate_acc = False
        self.evaluate_cm = False
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.classifier()
    def classifier(self):
        print('Running model')
        # XGBClassifier initialisation
        self.model = LogisticRegression(random_state = 0, solver='lbfgs', max_iter=400)
        # Training of model
        self.model.fit(self.X_train, self.y_train)
        # Evaluation the model
        self.evaluate_acc = self.model_evaluate(self.model)
        self.evaluate_cm = self.model_evaluate(self.model)
    def model_evaluate(self, model):
        # Make predictions using test set
        y_pred = model.predict(self.X_test)
        # Calculating accuracy
        accuracy = accuracy_score(self.y_test, y_pred)
        # Building confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        return (accuracy, cm)

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = dataset_loader.DatasetLoader('./assets/ISCX_Botnet(Prep).csv')
    # Run model and print accuracy score and cm
    lr = LogisticRegression(X_train, X_test, y_train, y_test)
    print('Model accuracy score: {}'.format(lr.evaluate_acc[0]*100.00))
    print(lr.evaluate_cm[1])
```

7.7 Code for Naïve Bayes

This file runs the Naïve Bayes model defined in section 3.6.

```
import header
from header import *

class NaiveBayes:
    def __init__(self, X_train, X_test, y_train, y_test):
        self.model = False
        self.evaluate_acc = False
        self.evaluate_cm = False
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.classifier()
    def classifier(self):
        print('Running model')
        # XGBClassifier initialisation
        self.model = GaussianNB()
        # Training of model
        self.model.fit(self.X_train, self.y_train)
        # Evaluation the model
        self.evaluate_acc = self.model_evaluate(self.model)
        self.evaluate_cm = self.model_evaluate(self.model)
    def model_evaluate(self, model):
        # Make predictions using test set
        y_pred = model.predict(self.X_test)
        # Calculating accuracy
        accuracy = accuracy_score(self.y_test, y_pred)
        # Building confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        return (accuracy, cm)

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = dataset_loader.DatasetLoader('./assets/ISCX_Botnet(Prep).csv')
    # Run model and print accuracy score and cm
    nb = NaiveBayes(X_train, X_test, y_train, y_test)
    print('Model accuracy score: {}'.format(nb.evaluate_acc[0]*100.00))
    print(nb.evaluate_cm[1])
```

7.8 Code for decision tree

Runs the Decision Tree model defined in section 3.7.

```
import header
from header import *

class DecisionTree:
    def __init__(self, X_train, X_test, y_train, y_test):
        self.model = False
        self.evaluate_acc = False
        self.evaluate_cm = False
        self.X_train = X_train
        self.y_train = y_train
        self.X_test = X_test
        self.y_test = y_test
        self.classifier()
    def classifier(self):
        print('Running model')
        # XGBClassifier initialisation
        self.model = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
        # Training of model
        self.model.fit(self.X_train, self.y_train)
        # Evaluation the model
        self.evaluate_acc = self.model_evaluate(self.model)
        self.evaluate_cm = self.model_evaluate(self.model)
    def model_evaluate(self, model):
        # Make predictions using test set
        y_pred = model.predict(self.X_test)
        # Calculating accuracy
        accuracy = accuracy_score(self.y_test, y_pred)
        # Building confusion matrix
        cm = confusion_matrix(y_test, y_pred)
        return (accuracy, cm)

if __name__ == '__main__':
    X_train, X_test, y_train, y_test = dataset_loader.DatasetLoader('./assets/ISCX_Botnet(Prep).csv')
    # Run model and print accuracy score and cm
    dt = DecisionTree(X_train, X_test, y_train, y_test)
    print('Model accuracy score: {}'.format(dt.evaluate_acc[0]*100.00))
    print(dt.evaluate_cm[1])
```

7.9 Code for GAN

```
import header
from header import *

class GAN():

    def __init__(self, gan_args):
        [self.batch_size, lr, self.noise_dim,
         self.data_dim, layers_dim] = gan_args

        self.generator = Generator(self.batch_size).\
            build_model(input_shape=(self.noise_dim,), dim=layers_dim, data_dim=self.data_dim)

        self.discriminator = Discriminator(self.batch_size).\
            build_model(input_shape=(self.data_dim,), dim=layers_dim)

        optimizer = Adam(lr, 0.5)

        # Build and compile the discriminator
        self.discriminator.compile(loss='binary_crossentropy',
                                   optimizer=optimizer,
                                   metrics=['accuracy'])

        # The generator takes noise as input and generates imgs
        z = Input(shape=(self.noise_dim,))
        record = self.generator(z)

        # For the combined model we will only train the generator
        self.discriminator.trainable = False

        # The discriminator takes generated images as input and determines validity
        validity = self.discriminator(record)

        # The combined model  (stacked generator and discriminator)
        # Trains the generator to fool the discriminator
        self.combined = Model(z, validity)
        self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)
```

```
    def get_data_batch(self, train, batch_size, seed=0):

        start_i = (batch_size * seed) % len(train)
        stop_i = start_i + batch_size
        shuffle_seed = (batch_size * seed) // len(train)
        np.random.seed(shuffle_seed)
        # wasteful to shuffle every time
        train_ix = np.random.choice(list(train.index), replace=False, size=len(train))
        # duplicate to cover ranges past the end of the set
        train_ix = list(train_ix) + list(train_ix)
        x = train.loc[train_ix[start_i: stop_i]].values
        return np.reshape(x, (batch_size, -1))

    def train(self, data, train_arguments):
        [cache_prefix, epochs, sample_interval] = train_arguments

        data_cols = data.columns

        # Adversarial ground truths
        valid = np.ones((self.batch_size, 1))
        fake = np.zeros((self.batch_size, 1))

        for epoch in range(epochs):
            # ---------------------
            #  Train Discriminator
            # ---------------------
            batch_data = self.get_data_batch(data, self.batch_size)
            noise = tf.random.normal((self.batch_size, self.noise_dim))

            # Generate a batch of new images
            gen_data = self.generator.predict(noise)

            # Train the discriminator
            d_loss_real = self.discriminator.train_on_batch(batch_data, valid)
            d_loss_fake = self.discriminator.train_on_batch(gen_data, fake)
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

            # ---------------------
```

```
    # ---------------------
    #  Train Generator
    # ---------------------
    noise = tf.random.normal((self.batch_size, self.noise_dim))
    # Train the generator (to have the discriminator label samples as valid)
    g_loss = self.combined.train_on_batch(noise, valid)

    # Plot the progress
    print("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100 * d_loss[1], g_loss))

    # If at save interval => save generated events
    if epoch % sample_interval == 0:
        #Test here data generation step
        # save model checkpoints
        model_checkpoint_base_name = 'model/' + cache_prefix + '_{}_model_weights_step_{}.h5'
        self.generator.save_weights(model_checkpoint_base_name.format('generator', epoch))
        self.discriminator.save_weights(model_checkpoint_base_name.format('discriminator', epoch))

        #Here is generating the data
        z = tf.random.normal((432, self.noise_dim))
        gen_data = self.generator(z)

plt.subplot(2, 1, 1)
plt.plot(d1_hist, label='d-real')
plt.plot(d2_hist, label='d-fake')
plt.plot(g_hist, label='gen')
plt.legend()
# plot discriminator accuracy
plt.subplot(2, 1, 2)
plt.plot(a1_hist, label='acc-real')
plt.plot(a2_hist, label='acc-fake')
plt.legend()
# save plot to file
plt.savefig('plot_line_plot_loss.png')
plt.close()
```

```
    def save(self, path, name):
        assert os.path.isdir(path) == True, \
            "Please provide a valid path. Path must be a directory."
        model_path = os.path.join(path, name)
        self.generator.save_weights(model_path)  # Load the generator
        return

    def load(self, path):
        assert os.path.isdir(path) == True, \
            "Please provide a valid path. Path must be a directory."
        self.generator = Generator(self.batch_size)
        self.generator = self.generator.load_weights(path)
        return self.generator

class Generator():
    def __init__(self, batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, dim, data_dim):
        input= Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(dim, activation='relu')(input)
        x = Dense(dim * 2, activation='relu')(x)
        x = Dense(dim * 4, activation='relu')(x)
        x = Dense(data_dim)(x)
        return Model(inputs=input, outputs=x)

class Discriminator():
    def __init__(self,batch_size):
        self.batch_size=batch_size

    def build_model(self, input_shape, dim):
        input = Input(shape=input_shape, batch_size=self.batch_size)
        x = Dense(dim * 4, activation='relu')(input)
        x = Dropout(0.1)(x)
        x = Dense(dim * 2, activation='relu')(x)
        x = Dropout(0.1)(x)
        x = Dense(dim, activation='relu')(x)
        x = Dense(1, activation='sigmoid')(x)
        return Model(inputs=input, outputs=x)
```