Polymorphic Adversarial DDoS attack on IDS using GAN

by

Ravi Chauhan

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

December 2020

THESIS EXAMINATION INFORMATION

Submitted by **Ravi Chauhan**

Master of Science in **Computer Science**

Thesis title: Polymorphic Adversarial DDoS attack on IDS using GAN

An oral defense of this thesis took place on December 8th, 2020 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Faisal Qureshi
Research Supervisor	Shahram Shah Heydari
Examining Committee Member	Stephen Marsh
Thesis Examiner	Ying Zhu

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

IDS are essential components in preventing malicious traffic from penetrating networks. IDS have been rapidly enhancing their detection ability using ML algorithms. As a result, attackers look for new methods to evade the IDS. Polymorphic attacks are favorites among the attackers as they can bypass the IDS. GAN is a method proven in generating various forms of data. It is becoming popular among security researchers as it can produce indistinguishable data from the original data. I proposed a model to generate DDoS attacks using a WGAN. I used several techniques to update the attack feature profile and generate polymorphic data. This data will change the feature profile in every cycle to test if the IDS can detect the new version attack data. Simulation results from the proposed model show that by continuous changing of attack profiles, defensive systems that use incremental learning will still be vulnerable to new attacks.

Keywords: Adversarial Attacks, Generative Adversarial Networks (GAN), Intrusion Detection System, DDoS attacks, Machine Learning

AUTHOR'S DECLARATION

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for scholarly research. I understand that my thesis will be made electronically available to the public.

Ravi Chauhan

STATEMENT OF CONTRIBUTIONS

I hereby certify that I am the sole author of this thesis. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis. Some parts of the work described in Chapter 3, 4, and 5 have been published as follows:

Ravi Chauhan and Shahram Shah Heydari. "Polymorphic Adversarial DDoS attack on IDS using GAN." Proceedings of the 2020 International Symposium on Networks, Computers, and Communications (ISNCC): MLNGSN Workshop, 20-22 October 2020, Montreal, Canada.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my research supervisor Dr. Shahram Shah Heydari, for the continuous expert guidance, motivation and provided me the opportunity to do this research. I am also grateful to him for his patience and continuous support throughout my graduate studies. I am honored and privileged to work and study under his supervision. I am also grateful to my parents for their endless support, belief, prayers that gave me the strength for my future endeavors.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 3

CHAPTER 5

Appendix A

Appendix B

LIST OF FIGURES

APPENDIX

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CICIDS	Canadian Institute of Cybersecurity Intrusion Detection System
CIDDS	Coburg Intrusion Detection Data Sets
DARPA	Defense Advanced Research Projects Agency
DCGAN	Deep Convolutional GAN
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
ECML-PKDD	European Conference on Machine Learning and Principles KDD
FP	False Positives
FN	False Negatives
GAN	Generative Adversarial Networks
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
KDD	Knowledge Discovery in Databases
ML	Machine Learning
RL	Reinforcement Learning
SQL	Structured Query Language
SIEM	Security Information and Event Management
SHAP	SHapley Additive exPlanations
TP	True Positives
TN	True Negatives
WGAN	Wasserstein GAN
XPATH	XML Path Language

Chapter 1. Introduction

1.1 Background

The Internet is being used in many fields, like data transfer, e-learning, and many more, and its growth has impacted all aspects of life. This increasing usage of the Internet causes concerns about network security and needs constant improvements in securing Internet technologies from various attacks. Examples of these attacks include DDoS attacks, Man-in-the-middle attacks, Phishing, Password-based attack, SQL injection, and many more. Network vulnerabilities can cause damage to small or large organizations. According to one survey, 98% of businesses in the UK depend on Information Technology services. Over 43% of small scale and 72% of large-scale organizations suffered from cyber-attacks in the past years [1]. There are many tools available to secure or prevent cyber-security attacks, including but not limited to: Intrusion Detection Systems (IDS) , Intrusion Prevention Systems (IPS), Anti-malware, Network Access Control, Firewalls. Among those, one of the most commonly used and effective tool is the Intrusion Detection System.

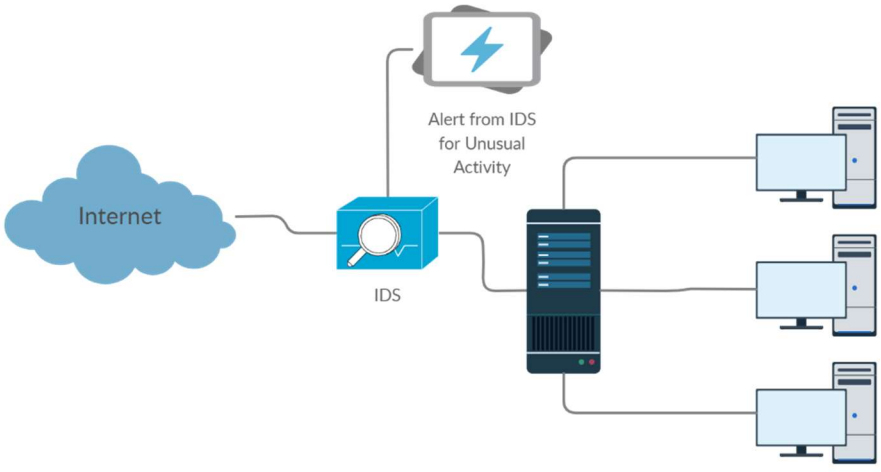

Figure 1: How IDS works?

IDS analyzes the traffic data to distinguish between malicious and normal traffic and generate alerts so that necessary precautions can be carried out to prevent damage [2]. With the advancement in network attacks, the security detections and prevention systems are also improving. Artificial Intelligence (AI) is now commonly used in defensive measures in IDS [3], and attackers have also started to use AI techniques for generating malicious attacks [4], [5].

AI and machine learning algorithms need a large amount of data to train and test the models. Some techniques that can be used to generate large datasets include malware detection on endpoints [6], [7], security orchestration [8], and SIEM [9], [10]. Many network attack datasets [11], [12], [13] are also available on the Internet. Therefore, researchers can develop a model to test and train detection and prevention systems to improve the security of the network. Just as such, malware authors or attackers also use machine learning techniques to generate synthetic/adversarial network attack data to evaluate the security systems [4], [5]. Adversarial samples are data that can cause a machine-learning algorithm to misclassify the attack.

One of the frameworks to generate adversarial data is Generative Adversarial Networks (GAN). It is an architecture that consists of two neural networks: the Generator and the Discriminator. The Generator uses gradient descent or the response from the discriminator and generates adversarial data. The discriminator distinguishes between the original and the adversarial data. The Generator and the discriminator compete in this way, and, in the end, the Generator produces synthetic or adversarial data [14]. GAN has been utilized in research to generate various types of datasets like images [15], sound [16], text [17], and network attack data [18].

1.2 Polymorphic Attacks

Polymorphic is a term that consists of a vital keyword, morph, which means changing the form. In the context of network security, polymorphic attacks refer to the type of attack that mutates attack signatures to evade the detection techniques. The polymorphic attack mutates in such a way that it maintains functionality. The polymorphic engine is employed to change the signature of the attack, as shown in the following image.

The polymorphic property of attacks makes old detection signatures obsolete [19]. Attackers continuously find new ways to manipulate attacks using various technologies to evade the detection systems. These types of attacks are specifically deployed when the detection system uses signature-based pattern matching techniques. The first known polymorphic attack was used to generate malicious URLs for a phishing attack [19]. That evades the signature-based anti-phishing defense systems, and defense systems are unable to blacklist malicious URLs.

1.3 Motivation

The amount of research that produces adversarial attack data using machine learning and AI methods is limited and mostly in specific formats like image, text, and sound. For example, according to surveys, phishing attacks are automated using adversarial AI [20], [21] suggested that attackers are using GAN to generate the voice of a group of people to breach the security access. However, it is essential to understand that results of machine learning classifiers are skeptical in terms of network security.

Furthermore, in network security, some of the tasks involve attackers' efforts that evade the detection systems. We state that it is vital for the network designers to ensure various techniques that an adversary/attacker can adapt to evade the detection system.

Moreover, the DDoS attacks are the most common to generate using simple scripting tools like Slowloris [22], Goldeneye [23], Hulk [24]. Apart from that, these attacks mostly target specific organizations and need fewer resources to produce attacks.

Based on these motivations, I assume that an attacker is able to manipulate the behavior of the network attack in such a way that maintains the intensity of the attack but can be misclassified as a regular network flow. Furthermore, Wasserstein GAN [25] is the favored method that an attacker can use to generate a synthetic attack. It can also be useful in manipulating the attack data classified as the regular network flow. This architecture will be useful for the network security tool designers to prepare for such unknown, polymorphic attacks.

1.4 Related Works

1.4.1 Evolution of Network Datasets

Network security has always been crucial, and it is essential to offer a standard of protection to every network service. Researchers have been collecting and developing network attack datasets using different scenarios and environment settings to evaluate different defensive models. A dataset has real information collected from the devices in the form of packets or logs that could help learn the patterns in network flow and is used as a baseline to determine if there is an anomaly in the network traffic.

A survey in [26] represents various datasets based on the network that has been operational for many years; it also compares and explains the datasets in brief. One

commonly applied dataset is KDD Cup 1999, which was collected in MIT Lincoln Labs by DARPA. It consists of Denial of Service, user-to-root, Remote to Local, and Probing attacks.

ECML-PKDD 2007 dataset was collected for a conference on Machine Learning by the EU. That has attacks such as Cross-Site Scripting, SQL Injection, XPATH Injection, and command execution. This dataset was compiled in the XML format. Another Data set covered by this research paper is HTTP CSIC 2010 [27], prepared by the Spanish Research National Council. That consists of regular and anomalous HTTP requests and contains attack types like SQL Injection, CSS Scripting, Buffer Overflows, and so forth.

As the detection systems evolved with time, various attacks also emerged that could compromise networking systems. The dataset needs to be updated with the latest attack features. Canadian Institute for Cyber Security developed a dataset from real-time simulations. They have published various datasets involving numerous attacks like Android malware, Botnet, DOS, DDoS, and many more. Intrusion Detection, Evaluation dataset known as CICIDS2017 [13] contains benign data and commonly known attacks like Brute Force SSH, DOS, Web Attack, Botnet, and DDoS. To produce a reliable data set, authors have considered critical criteria like complete network configuration, complete traffic, labeled dataset, complete interaction, complete capture, available protocols, attack diversity, feature set, which consists of more than 80 features, and metadata. None of the previous datasets have considered these benchmarks.

1.4.2 Advancements in Network Security

Network security has been a significant research trend for the past few decades. AI and Machine Learning are the main areas of research in network security. Li et al. proposed a

framework to evaluate the risk of network security using Support Vector Machine [28]. Machine Learning and AI algorithms are efficient and flexible to deploy. However, a survey [29] proposed by Guan et al. explains that there are some issues in machine learning itself, like data poisoning while training the model that can affect the overall performance of the framework. They have also suggested a possible solution to avoid data poisoning. There are various machine learning approaches that can detect network anomalies discussed in [30] and the performance comparison of these techniques.

IDS is an essential part of network security, and to improve the functionalities and ability of the IDS, researchers have been working on various techniques. For instance, [31] authors proposed a kernel-based IDS that can detect anomalies like DDoS attacks. It uses the K-means clustering technique to classify between adversarial and standard examples. There are two techniques used by IDSs to detect various attacks, Signature-based detection and anomaly-based detection. The first technique analyzes a network for a specific pattern of predefined attack. The limitation of this technique is that it is unable to detect an unknown attack. The second method classifies the network flow data into standard and anomalous data. This technique uses statistical methods, machine learning in IDS.

ML algorithms like Random Forest, Decision Tree, Support Vector Machines are being used for anomaly detection [32]. However, most of them resulted in lower accuracy. A Study in [32] analyzes challenges on anomaly detection by using Deep Learning techniques based on semi-supervised learning. Results show that using an Auto Encoder technique, and the model gets reasonable detection accuracy with a fair amount of training data. A survey [33] represents using various machine learning classifiers like Support Vector Machine, K-nearest Neighbor, Decision Tree in IDS classification. A review by [34]

represents the role of machine learning techniques like Supervised, Unsupervised, and Hybrid classifiers in IDS and in creating AI-based attacks. Kumar et al. [35] proposed a model that uses Deep Learning to prevent DDoS attacks.

A recent survey [36] points towards the recent developments and shortcomings in IDS. It suggests that the main concern with the latest IDS is that they tend to alarm on fake attack data, which is a result of high false positives in terms of machine learning. Furthermore, it compares the detection rate and accuracy between various machine learning techniques like ANN, SVM, Naïve Bayes, Random Forest, and AdaBoost.

Various studies explain techniques to reduce the false-positive rate. For example, [37] proposed a model that applied a genetic feature selection technique. That specifies only the most essential features from the dataset that needs to be used to train the model. In addition to that, it uses a Support Vector Machine algorithm to classify the network flow data. Another research [38] developed a framework that uses the Random Forest technique to improve the false-positive rate. This method follows a multi-layer classification approach, in which the classifier first distinguishes if the incoming data is attack or not. If it is an attack, it will be further simplified by type of it. To balance the classification of attack types, they have used a cluster-based under-sampling method.

1.4.3 Attacks using Generative Adversarial Networks

With the recent developments towards machine learning techniques, intrusion detection systems are getting advanced with these methods. However, there is limited research testing the integrity of the advanced IDS against adversarial data.

According to a study by [39], the authors created a framework that generates adversarial malware using GAN to bypass the detection system. The objective of this research is to use a black-box malware detector because most of the attackers are unaware of the detection techniques used in the detection system. Instead of directly attacking the black-box detector, researchers created a model that can observe the target system with corresponding data. Then this model calculates the gradient computation from the GAN to create adversarial malware data. With this technique, the authors received a model accuracy of around 98%.

Some researchers have also examined the same methodology in generating adversarial attacks for Android applications. [40] Presented a model to generate adversarial android malware using Generative Adversarial Networks. The model consists of the Generator, the discriminator, and Malware Detector. The Generator gets a random noise vector and produces the adversarial data. The discriminator gets benign data and adversarial malware and then differentiates between real and perturbed data. The discriminator provides feedback in the form of loss to the generator. If the generated sample is distinguishable, it will increase loss and decrease it otherwise. They have used various classifiers like Support Vector Machine, Random Forest, Logistic Regression as the machine learning classifiers for the GAN model.

1.4.4 Prior work on the use of WGAN in security

This section covers some previous works on generating adversarial attack data using the Wasserstein GAN. The Wasserstein GAN model was introduced in [25], and it improves upon the traditional GAN. Wasserstein GAN is an extension of traditional GAN that finds an alternate method of training the Generator. In WGAN the Discriminator

provides a critic score that depicts how real or fake the data generated. More detailed information about this method, which is used in this research, is provided in *Section 2.5*.

To generate a malicious file [18] proposed a method that uses WGAN so that a detection system signifies the adversarial malicious file as a regular file. They have achieved an accuracy of around 99%, proving that their method can generate adversarial malicious files that can bypass the detection system.

A recent study in [41] uses Wasserstein GAN to generate simulated attack data. According to the authors, many tools can generate simulated attack data. However, this process could take a long time and a lot of resources. Using the proposed technique, they have produced millions of connection records with just one device and within a short period. They used the KDD Cup 1999 dataset as the training set. Their experiment suggests that as compared to GAN, the Wasserstein GAN learns faster and generates better results.

A paper published by Ring et al. [42] proposed a method that produces flow-based attack data using Wasserstein GAN. This research uses the CIDDS dataset to test and train the proposed method. They have suggested that the flow-based dataset consists of categorical features like IP address, port numbers, etc. The GAN is unable to process categorical data. They have also proposed a method to preprocess the categorical data and transform them into continuous data. Lastly, they have used several techniques to evaluate the quality standard of the adversarial data. Results suggest that it is possible to generate real network data using this method.

A recently published paper by Lin et al. [43] discussed the benefits of WGAN. It proposed a technique IDSGAN to generate adversarial attack data and test the attack

against the Intrusion Detection System. They have utilized the NSL-KDD as the benchmark dataset to generate an adversarial attack on an IDS. They have tested this technique with various machine learning classifiers like Support Vector Machine, Naïve Bayes, Multilayer Perceptron, Linear Regression, Decision Tree, Random Forrest. They have used four types of attacks, Probe, DoS, User to Root, Root to Local to generate adversarial attack data.

1.5 Research Gap

Some work in generating adversarial attack using GAN are focusing on generating adversarial data and test if the IDS can detect the attack. It was found that most of them do not focus on training the IDS with adversarial data generated by the GAN and test if the IDS can detect similar kinds of attacks in the following cycles. The prior research is also lacking the idea of polymorphic attacks, i.e., to update the feature profile by manipulating the features of training data and try to generate a new variety of adversarial data to evaluate the IDS. Apart from that, some researches are based on the GAN model developed by Goodfellow et al. [14]. Research shows that this traditional variant of GAN does not scale with a large dataset, so it is unstable with large scale applications [44].

1.6 Aim and Objective

This research aims to generate an AI-based adversarial DDoS attack using GAN. Moreover, this attack will be profile-based, which means the attack will change its feature profile for a certain period. To automate the updating of the feature profile, I have used the Reinforcement Learning technique. That will evaluate the success rate of the attack on IDS and update the feature profile as it moves forward.

The objective of the thesis is to build a model that generates automated feature profile based polymorphic DDoS attacks and evaluates if it can evade the IDS. This work also includes monitoring the performance of the GAN model, for how many cycles the polymorphic adversarial DDoS attack can evade the IDS. I have also used a feature selection technique, SHAP [45] (SHapley Additive exPlanations), to distinguish essential features from the entire dataset.

1.7 Thesis Contributions

This research aims to create a framework that generates adversarial polymorphic DDoS attacks using GAN, motivated by [43].

- This work begins with the important feature selection method using SHAP. I have identified the most critical features from the dataset that contribute to a DDoS attack.
- The next goal is to Generate adversarial data using the selected feature set and evaluate the IDS if it can detect the adversarial attack, followed by training the IDS with the generated adversarial data.
- I propose a polymorphic engine that updates the feature profile of the attack. There are two types of polymorphic engine,
    1) Manual feature update – In this technique, I will manually select and add new features in the training set after every adversarial DDoS attack and IDS training cycle. Another variant of the feature profile update is to shuffle the new features with the old features. I have used this technique as a baseline to evaluate the performance of the model.

2) Automated feature update – In this case, I use the Reinforcement Learning method to update the feature profile after every adversarial DDoS attack and IDS training cycle.

- I have conducted a comprehensive simulation and analyzed the results to compare the Reinforcement Learning method against the Manual Feature profile attacks and presented how many cycles an attacker can bypass an IDS with polymorphic adversarial DDoS attacks.

1.8 Thesis Outline

The rest of this thesis is structured as follows: Chapter 2 consists of a brief introduction to GAN, previous work that utilizes GAN to produce synthetic data, and an introduction to Wasserstein GAN. A detail about the dataset used in this research and feature selection techniques are discussed in Chapter 3. Chapter 4 explains the proposed methods that generate an adversarial polymorphic DDoS attack and automation of attack feature update profile using Reinforcement Learning. Chapter 5 describes the experiment environment, experiment scenarios, and analysis of this work. Chapter 6 concludes the work with suggested future work.

Chapter 2. Generative Adversarial Networks

2.1 Introduction

Generative Adversarial Network is a paradigm based on machine learning models that can generate synthetic data from the original input data. It consists of two neural networks known as the Generator and the discriminator. It was proposed by Goodfellow et al. [14] at the University of Montreal.

2.1.1 Discriminator

The discriminator can be simply called a classifier that distinguishes the generated data as original or fake. The discriminator takes two forms of data, original data, and the data generated by the Generator.

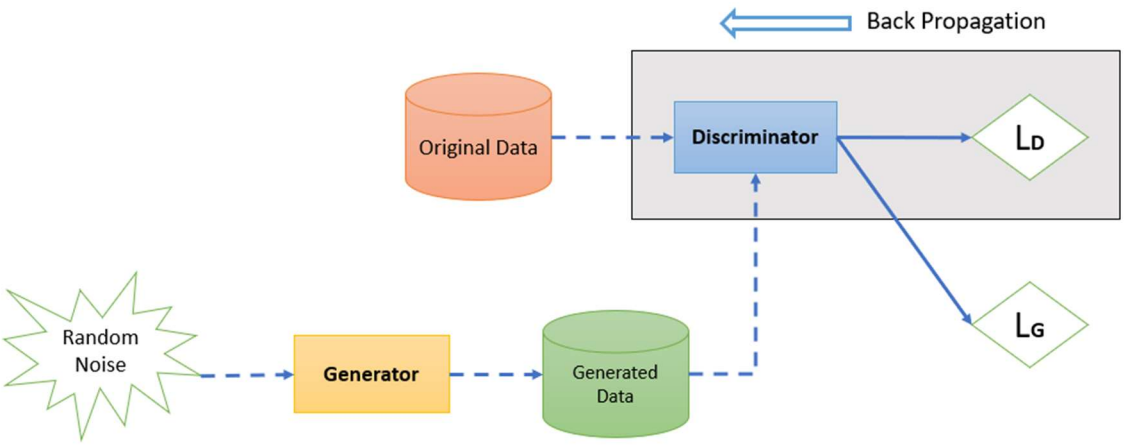

Figure 2: Backpropagation in the Discriminator [46]

The discriminator uses original data as a positive example and generated data as negative/adversarial examples during training. L_D represents the penalty to the discriminator when the discriminator cannot detect or correctly differentiate the data; the

penalty increases and decreases otherwise. To update the weights of the discriminator, it uses backpropagation. Another loss L_G represents a loss of the Generator [46].

2.1.2 Generator

The generator produces a synthetic set of data by receiving feedback from the discriminator and learns to produce data so that the discriminator classifies the synthetic data as the original.

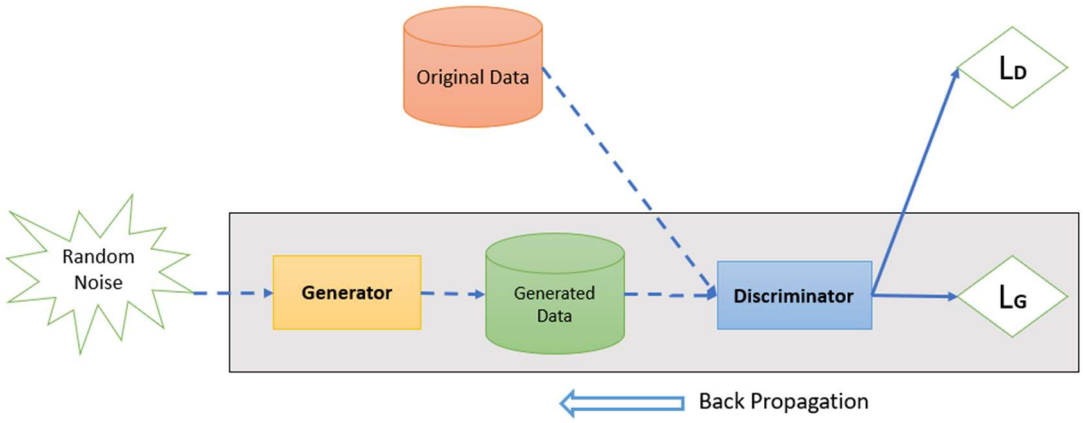

Figure 3: Backpropagation in the Generator [46]

The training of the Generator includes steps as follows.

1) A random input noise.
2) The Generator to produce data from the random data.
3) The discriminator, to distinguish the data and the output from the discriminator.
4) Loss L_G that fines the Generator if it is unable to produce data that can deceive the discriminator.

2.2 Training the GAN

As discussed, there are two variants of neural networks in the GAN, so it needs to train the Generator and the discriminator alternately. Also, it is hard to check if the GAN is converged or not. The alternative training works as follows.

1) Training of the Generator runs for some epochs.
2) Training of the discriminator runs for some epochs.
3) Continue repeating steps 1 and 2 until the GAN converges.

To train the GAN better, we need to keep either of the neural networks constant. For instance, while training the Generator, we need to keep the discriminator constant; otherwise, it will be difficult to converge [47]. While training the discriminator, the Generator needs to be constant because it needs to learn to differentiate between the generated and fake data.

2.3 Loss Function

The loss function represents the difference value between the generated data and the adversarial data.

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}\left[log\left(D(x)\right)\right] + E_{z\sim p_z(z)}\left[log\left(1-D\left(G(z)\right)\right)\right] \quad (1)$$

[14] discusses a loss function named min-max loss. Authors trained the discriminator D to maximize the average of the $log\ D(x)$, with *D(x)* denoting the estimated probability of data being recognized as original data and the $log\left(1-D\left(G(z)\right)\right)$, with *D(G(z))* denoting the estimated probability of data being recognized as synthetic. Moreover, the authors concurrently train the generator G seeks to minimize the *log(1−D(G(z)))* predicted

by the discriminator for synthetic data. The discriminator D and the generator G plays a min-max game with the following value function *V(D, G)*.

Here, E_X represents the expected value over the original data. *D(x)* is the approximation that if the original data is real or not. *G(z)* represents the output from the Generator from the noise *z*. *D(G(z))* is the approximate value of the discriminator that the generated data is real, $p_z(z)$ is input noise variables. Furthermore, E_Z represents the expected value of the random data inputs to the Generator. To minimize the loss of the Generator, we need to minimize the value of the *log(1 - D(G(z))*. Lower loss of G means the generator is producing synthetic data that can be classified as the Original.

2.4 Generating data using GAN

Generative Adversarial Networks are useful to generate numerous types of information in the form of images, sound, text, etc. This section shows several use cases in the field of data generation using GANs.

In computer vision, image generation using Machine Learning is becoming popular. According to [48], there is not much attention given to the gaming domain in the form of sprite generation. A *sprite* is a two-dimensional bitmap that can be integrated to generate a large screen in 2-D games. In [48], the authors proposed a framework based on Deep Convolutional GAN to generate new *sprites.* They have used three types of custom datasets that consist of human-like characters, faces, and creatures. The architecture of the Deep Convolutional GAN that was used by authors is as follows.

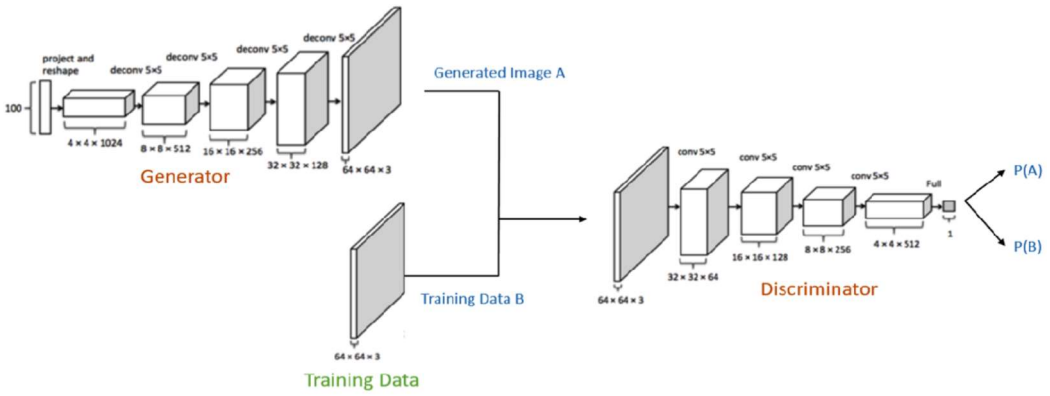

Figure 4: Generating Images using DCGAN [48]

The role of the Generator is to create random data in the form of images and keep improving the quality of an image by getting feedback from the Discriminator using backpropagation. The Discriminator then distinguishes the image from the generated data and training data. In the beginning, the Generator produces image samples that are easily classified by the Discriminator as fake. Afterward, the continuous training of the Generator improves the quality of an image that will be difficult to distinguish as fake.

Another exciting research domain is generating or manipulating text data using machine learning. Generating a text is more difficult because of the structure or the property of the Neural Networks. While training, Neural Network produces a new word by predicting previously generated words. However, with a lengthy list of words, the neural network will become uncertain and prone to errors [49]. To overcome the given situation, a recently published paper [49] proposed an architecture based on GAN to generate text called “tranGAN”.

Unlike traditional GAN, this proposed model has two generators; backward Generator and forward Generator. The backward Generator produces the first part of the sentence and

the second half is by the forward Generator. In comparison, the role of the discriminator is to differentiate between the original and the generated sentence.

Previous research [40], [44], [47] suggests that training a traditional GAN is hard. The most common problems are as follows.

Convergence problem - When the Generator gets better, the classification performance of the discriminator decreases in the following cycles. Training the GAN from this point means the generator trains from the less meaningful data; this state is known as the Convergence problem.

Mode Collapse – In ideal conditions, the GAN can produce a good variety of data. However, if a generator learns to produce a specific set of data so that the discriminator classifies them like the original, then the Generator will only produce these sets of data and easily deceive the discriminator. This condition is called mode collapse.

2.5 Wasserstein GAN

To overcome these issues, Arjovsky et al. proposed a method known as Wasserstein GAN [25]. Wasserstein GAN provides a better approximation of distributed data that was given in the training set. WGAN uses a discriminator with a critic that provides a score of how real or fake generated data is. In contrast, a discriminator in traditional GAN predicts and classifies the generated data as original or fake.

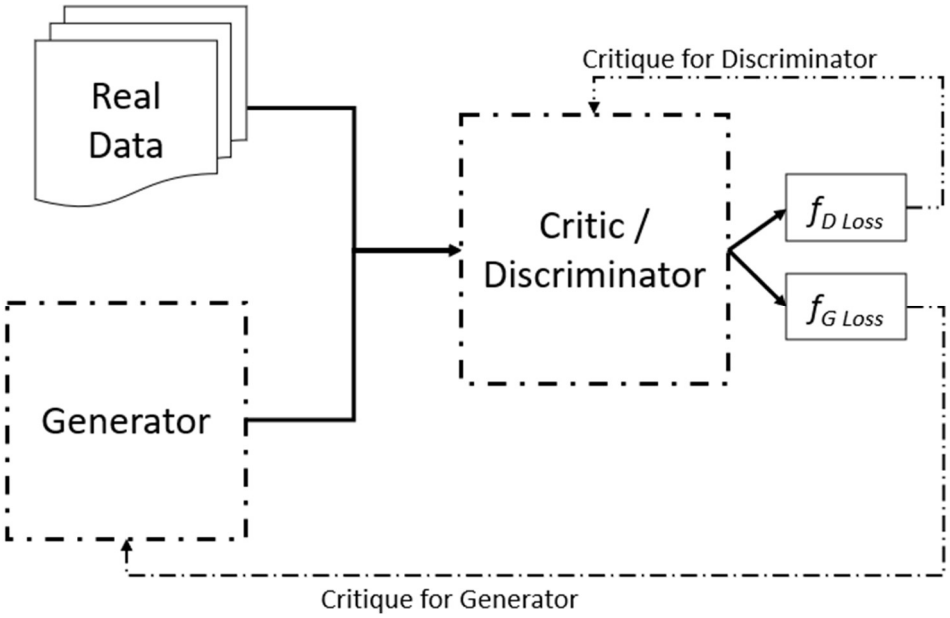

Figure 5: Network Diagram of WGAN

In this diagram, $f_{D\ loss}$ represents a Loss function that provides critique values for the Discriminator, and $f_{G\ loss}$ represents a Loss function for the generator. The following are the differences in the implementation of WGAN. A critic score < 0 depicts the real data, and a score > 0 depicts the fake or synthetic data.

1) A linear Activation function is being used in WGAN, whereas GAN uses Sigmoid as the activation function.
2) It trains or updates the Discriminator/Critic multiple times compared to Generator in each cycle.

Two loss functions WGAN uses are Discriminator/Critic Loss and Generator Loss that is as following.

$$L_D = \nabla_w \frac{1}{m} \sum_{i=1}^{m} [\, f_w(x^{(i)}) - f_w(g_\theta(z^{(i)})) \,] \qquad (2)$$

This function [25] specifies the Discriminator/Critic loss that can be simplified as a difference between the average critic score of real data and the average critic score of fake data. Here, $f_w(x^{(i)})$ represents an average critic score on real data and $f_w(g_\theta(z^{(i)}))$] represents an average critic score on fake/generated data.

$$L_G = \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w\left(g_\theta\left(z^{(i)}\right)\right) \quad (3)$$

The above function specifies the Generator loss that can be simplified as "*1 – average critic score of fake data"*. Where, $f_w\left(g_\theta\left(z^{(i)}\right)\right)$ depicts the average critic score of fake data. Overall contributions of the WGAN are that their experiments do not have the mode collapse, and the Generator learns well even if the critic accurately discriminates the adversarial data. Both the loss functions motivate a separation between a score for synthetic data and real data, which is not necessarily positive and negative [50].

It suffices to say that WGAN has been proven to generate high-quality synthetic attack data. I have followed a similar methodology to generate synthetic DDoS attack data. The following chapters depict the stages of the work done in this research.

Chapter 3. Dataset and Feature Selection

3.1 Dataset

I use a dataset published by the Canadian Institute of Cyber Security, CIC-IDS2017, published in [13] by Lashkari et al., which, according to the authors, supersedes the datasets generated earlier by the institute. CICIDS2017 consists of eight different files that contain regular traffic and attack traffic data. The following table represents the activity captured in each file.

Table 1: Description of files from CICIDS2017

	NAME OF FILES	DAY ACTIVITY	ATTACKS FOUND
1	Monday WorkingHours.pcap_ISCX.csv	Monday	Benign (Normal human activities)
2	Tuesday WorkingHours.pcap_ISCX.csv	Tuesday	Benign, FTP-Patator, SSH-Patator
3	Wednesday workingHours.pcap_ISCX.csv	Wednesday	Benign, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris, Heartbleed
4	Thursday-working hours Morning-WebAttacks.pcap_ ISCX.csv	Thursday	Benign, Web Attack – Brute Force, Web Attack – SQL Injection, Web Attack – XSS
5	Thursday-working hours Afternoon-Infilteration.pcap_ ISCX.csv	Thursday	Benign, Infiltration
6	Friday-working hours Morning.pcap_ISCX.csv	Friday	Benign, Bot

7	Friday-WorkingHours-Afternoon PortScan.pcap_ISCX.csv	Friday	Benign, PortScan
8	Friday-WorkingHours-Afternoon DDos.pcap_ISCX.csv	Friday	Benign, DDoS

Following is the table that depicts the properties of the dataset.

Table 2: Properties of CICIDS2017 Dataset

#	FEATURE	VALUES
1	Total number of flows	2830540
2	Total number of features	83
3	Number of classes/labels	15

Moreover, this dataset consists of various types of attacks along with the normal network flow. The following table consists of the attack and benign labels available in the dataset.

Table 3: Different labels in CICIDS2017

#	NORMAL / ATTACK LABELS	NUMBER OF FLOWS
1	BENIGN	2359087
2	Bot	1966
3	DDoS	41835
4	DoS GoldenEye	10293
5	DoS Hulk	231072
6	DoS Slow httptest	5499
7	DoS slowloris	5796
8	FTP-Patator	7938
9	Heartbleed	11

10	Infiltration	36
11	PortScan	158930
12	SSH-Patator	5897
13	Web Attack – Brute Force	1507
14	Web Attack – SQL Injection	21
15	Web Attack – XSS	652

Moreover, this dataset also covers all the available standard protocols like HTTP, HTTPS, FTP, SSH, and email protocols.

3.2 Features in the Dataset

The dataset consists of more than 70 features that are important as per the latest network standards, and most of them were not available in the previously known datasets. Following is the list of the features.

Table 4: List of features in CICIDS2017

No.	Feature	No.	Feature	No.	Feature
1	Flow Duration	**27**	Bwd IAT Std	**53**	Avg Fwd Segment Size
2	Total Fwd Packets	**28**	Bwd IAT Max	**54**	Avg Bwd Segment Size
3	Total Bwd Packets	**29**	Bwd IAT Min	**55**	Fwd Avg Bytes/Bulk
4	Total len of Fwd Packet	**30**	Fwd PSH Flags	**56**	Fwd Avg Packets/Bulk
5	Total len of Bwd Packet	**31**	Bwd PSH Flags	**57**	Fwd Avg Bulk Rate
6	Fwd Packet Length Max	**32**	Fwd URG Flags	**58**	Bwd Avg Bytes/Bulk
7	Fwd Packet Length Min	**33**	Bwd URG Flags	**59**	Bwd Avg Packets/Bulk
8	Fwd Packet Length Mean	**34**	Fwd Header Length	**60**	Bwd Avg Bulk Rate

9	Fwd Packet Length Std	**35**	Bwd Header Length	**61**	Subflow Fwd Packets
10	Bwd Packet Length Max	**36**	Fwd Packets/s	**62**	Subflow Fwd Bytes
11	Bwd Packet Length Min	**37**	Bwd Packets/s	**63**	Subflow Bwd Packets
12	Bwd Packet Length Mean	**38**	Min Packet Length	**64**	Subflow Bwd Bytes
13	Bwd Packet Length Std	**39**	Max Packet Length	**65**	Init_Win_bytes_fwd
14	Flow Bytes/s	**40**	Packet Length Mean	**66**	Act_data_pkt_fwd
15	Flow Packets/s	**41**	Packet Length Std	**67**	Min_seg_size_fwd
16	Flow IAT Mean	**42**	Packet Len. Variance	**68**	Active Mean
17	Flow IAT Std	**43**	FIN Flag Count	**69**	Active Std
18	Flow IAT Max	**44**	SYN Flag Count	**70**	Active Max
19	Flow IAT Min	**45**	RST Flag Count	**71**	Active Min
20	Fwd IAT Total	**46**	PSH Flag Count	**72**	Idle Mean
21	Fwd IAT Mean	**47**	ACK Flag Count	**73**	Idle Packet
22	Fwd IAT Std	**48**	URG Flag Count	**74**	Idle Std
23	Fwd IAT Max	**49**	CWE Flag Count	**75**	Idle Max
24	Fwd IAT Min	**50**	ECE Flag Count	**76**	Idle Min
25	Bwd IAT Total	**51**	Down/Up Ratio	**77**	Label
26	Bwd IAT Mean	**52**	Average Packet Size		

3.3 Feature Selection

Feature selection is an essential aspect of the machine learning technique. If we train the model without determining the critical features of the dataset, the predicted results will have more noise and uncertain results. Moreover, while using a dataset with a high number of feature sets, it is unnecessary to use all the available features because the machine

learning method uses more resources and time to process a large volume of feature sets. There are several techniques to select important features from a dataset. Shikh et al. [51] mentioned three methods to select features based on the type of dataset that are as follows.

(1) **Univariate Selection** – This method can select those features that have the most stable relationship with the output variable.

(2) **Feature Importance** – This method will be used to extract the features by their importance; it means every feature from the dataset will be given an importance score to determine the required features from the dataset.

(3) **Correlation Matrix with Heatmap** – This method states the relationship of the features to each other and the output variable using the Heatmap. A value of a correlation can be positive or negative according to the importance of the feature.

3.3.1 Feature Selection using SHAP

SHAP (Shapley Additive exPlanations) [44] is one of the new feature selection techniques. The goal of the proposed method is to signify the contribution of each feature to the predicted value. Two critical measures to define feature importance are Consistency and Accuracy. The authors of the paper discuss that SHAP is the method that satisfies these qualities. The SHAP values explained by the authors are based on Shapley values that are a concept from game theory. The idea behind Shapely values is that the outcome of each possible combination (or coalition) of each feature needs to be examined to determine the importance of a single feature. The mathematical explanation of this is as follows:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z_j' \qquad (4)$$

Here, g represents the overall result of the Shapely values, $z' \epsilon \{0, 1\}^M$ is a coalition vector, M is the max coalition size, and ϕ_j represents the presence of feature j that contributes towards the final output. The authors have described a coalition vector as simplified features in the paper. In coalition vector, 0 means the corresponding value is "not present" and 1 means it is "present."

Equation 4 can be called a power set and can be explained as a tree as follows.

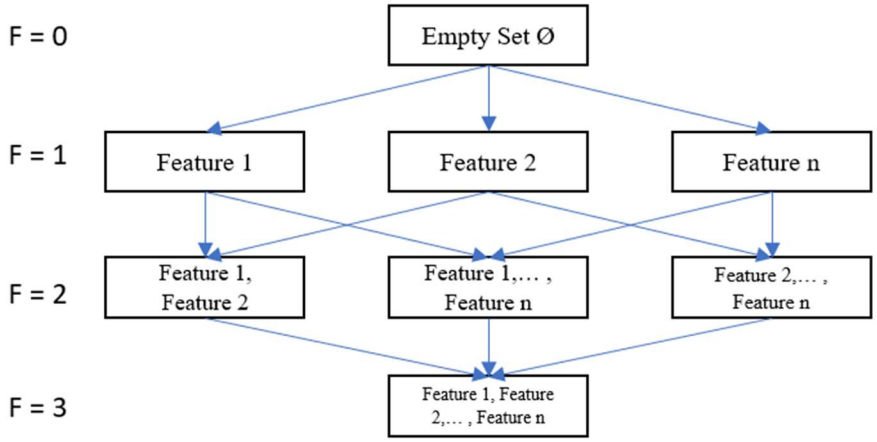

Figure 6: Power set of features

Each node here represents a coalition of features. Edges represent the inclusion of a feature that was not present in the previous coalition. Equation 4 trains each coalition in the power set of the features to find the most critical feature from the dataset.

3.3.2 Benefits of SHAP

The advantages of using SHAP are as follows:

1) *Global Interpretability*: This technique provides essential features from a dataset and a contribution of each feature for a target result and effect of the feature. To calculate global importance, we need to find an average of SHAP values.

$$I_j = \sum_{i=1}^{n} |\phi_j^{(i)}| \quad (5)$$

2) *Local Interpretability*: With this method, we can get an impact of an individual feature across the whole dataset.

3.3.3 Features Selected by SHAP

I ran the SHAP explainability model on the CICIDS2017 data file. The following results were obtained that show the list of essential features responsible for the DDoS attack in the order of most important to least important.

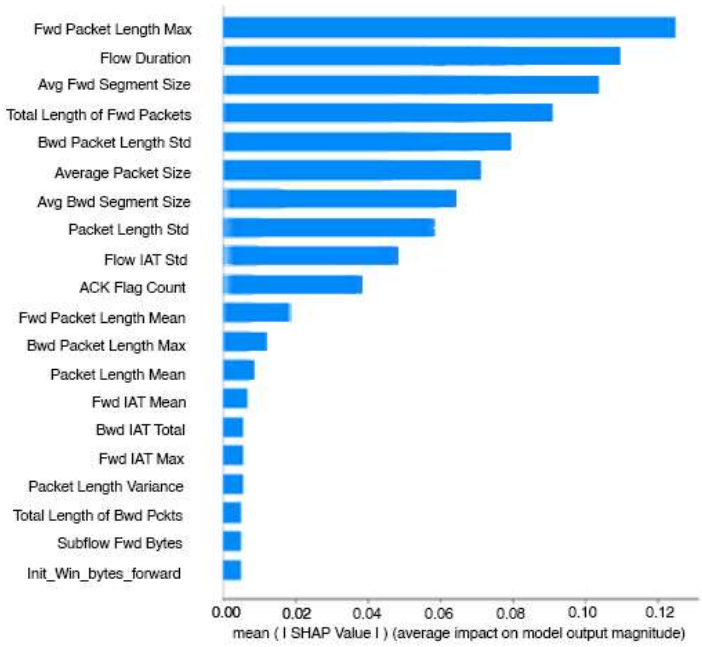

Figure 7: Feature Importance using SHAP

The result does not show enough information apart from the feature importance. However, another plot is known as a summary plot that can represent an effect of the feature, either positive or negative, on the result. Furthermore, the dark red color represents a higher impact of a feature, and the blue color represents a lower impact of a feature on the output value.

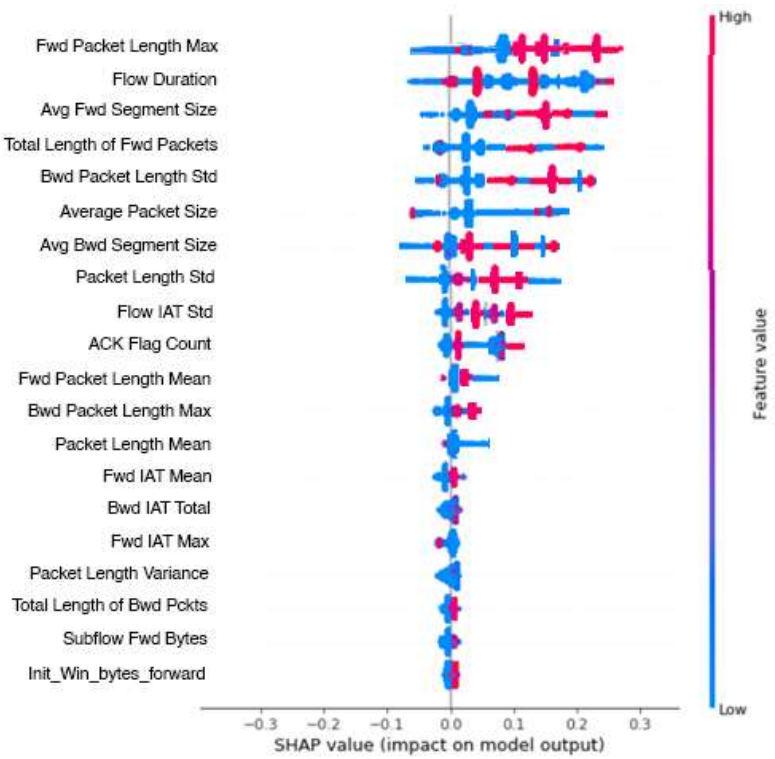

Figure 8: Summary Plot with Feature Impact using SHAP

So, from the results, I have used these features like functional features that contribute to the DDoS attacks.

Chapter 4. Proposed Framework

In this chapter, I will discuss the methodologies used in this research. It involves the Generative Adversarial model that produces adversarial attacks, training the IDS with previously generated polymorphic data, the polymorphic engine to generate polymorphic DDoS attacks, and use the polymorphic data to attack the IDS.

4.1 Adversarial Attack Generation using Wasserstein GAN

In this section, I will discuss the first stage of the framework, i.e., Generate the adversarial attack.

For this work, I have used DDoS attack data from the CICIDS2017 [13] to train the model. To generate an adversarial attack, I considered a combination of a random noise vector of the same size as the selected features from the dataset.

The Generator in this framework is a feed-forward neural network that consists of 5 linear layers. The input layer consists of neurons as per the selected number of features, and the output layer consists of 1 neuron.

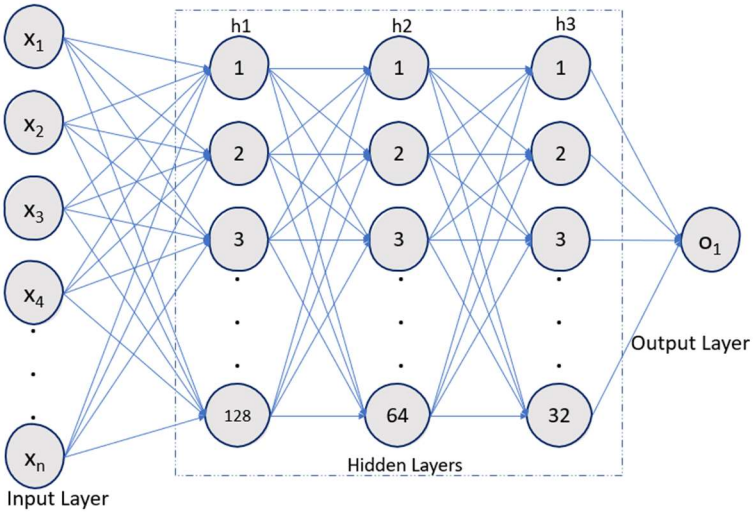

Figure 9: Neural network of the Generator

The input layer receives several numbers of features according to the experiment, and the output layer generates the desired data. The Generator consists of 3 hidden layers that are optimal for this scenario; my results showed fewer layers would underfit the training data. Anything more than that overfits the training data.

In the next step, the generated adversarial attack combined with the benign or normal network flow data will be fed to the Intrusion Detection System.

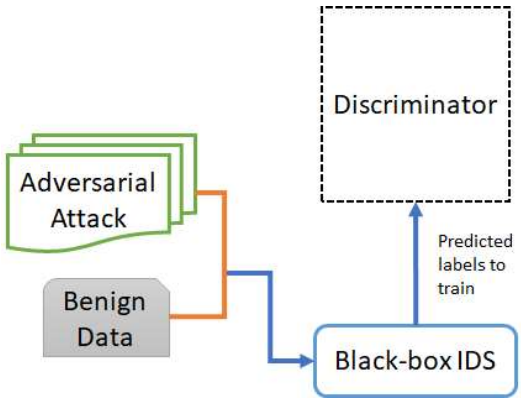

Figure 10: Training the Black-box IDS

The IDS will detect the attack and sends predicted labels to the Discriminator, the detection success rate, and the Discriminator will send the critique to the Generator using the backpropagation so that in the next cycle, the Generator can improve the production of adversarial DDoS attack. The IDS consists of 4 layers, from which the input and output layer consists of 2 neurons each. The IDS consists of 2 hidden layers that are ideal because it only detects if the test data consists of an attack or benign.

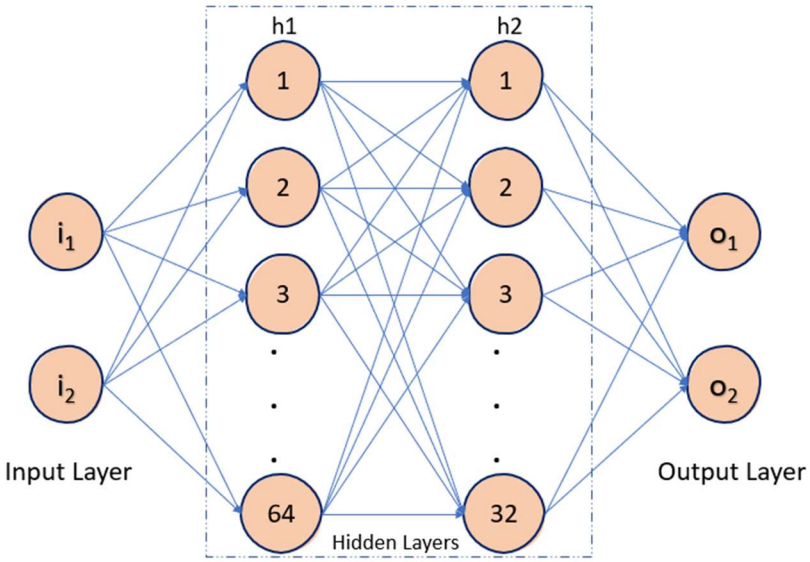

Figure 11: Neural network of the IDS and the Discriminator

There are two types of IDS available, i.e., white-box IDS and black-box DIS. I used a signature-based black-box intrusion detection system to test the detection rate of the adversarial DDoS attacks. The reason for using this is that most of the time, the type of attack detection system is unknown to the attackers. Attackers rely on the responses received from the detection system, and black-box IDS is the right choice for this model.

Finally, the Critic or Discriminator consists of 4 layers. The input layer accepts two types of data from the black-box IDS. The output layer provides two critics, one for the Generator and one for itself.

To calculate the Loss, I have used loss functions [43] for the Generator and the discriminator, which are as follows.

$$P_G = E_{M \in S_{attack},\ N} - D(G(M, N)) \qquad (6)$$

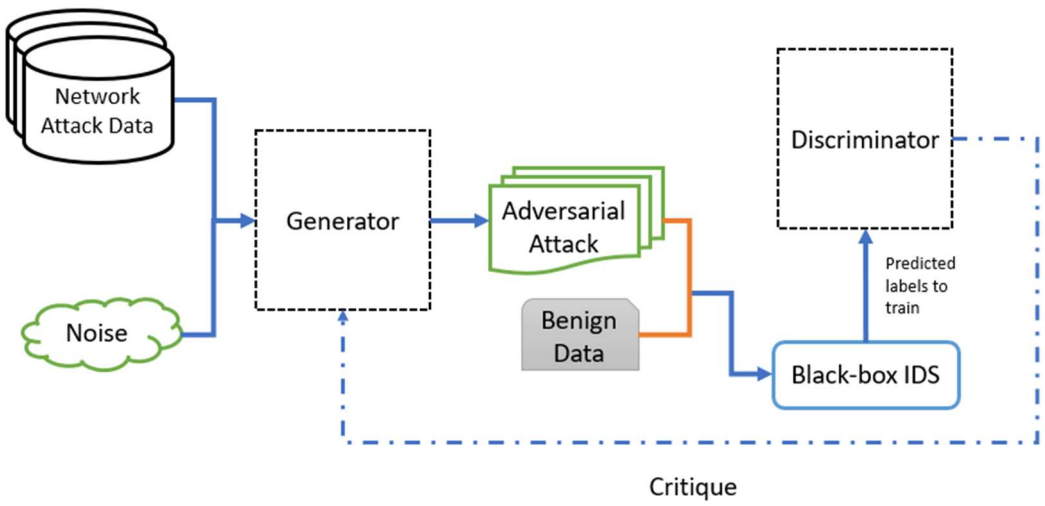

Figure 12: Generating Adversarial DDoS Attack

Here, P_G represents the Penalty to the Generator. M is an m-dimensional attack vector, and N is an n-dimensional noise vector. E is the estimated value over the random inputs to the Generator. S_{attack} represents. The lesser the penalty to the Generator means it is performing well and produces attack data that can bypass the IDS.

$$P_D = A_{S \in B_{Benign}} D(s) + A_{S \in B_{Attack}} - E_{S \in B_{Attack}} D(s) \quad (7)$$

Here, P_D represents the Penalty to the discriminator. "*E"* is the overall estimated feature values of the generated attack data. "*A"* is the actual feature value of benign and the attack data. The lesser the penalty to the discriminator means the discriminator performs well. It calculates if the generated data is closer to the DDoS attack or benign or regular data.

Algorithm – 1 shows the process that was represented in figure–11.

Algorithm 1: Adversarial Attack Generation

Input:

Generator – noise vector N, DDoS Attack Data

Critic / *Discriminator* - S_{attack}, and S_{benign}

Output:

Trained Critic / Discriminator and Generator

1: ***for*** epochs = 1, … , MAX EPOCHS ***do***

2: ***for*** G-iterations, ***do***

3: Generator creates adversarial network attacks using S_{attack}, and

Update the penalty using P_G function once it receives the critique.

4: ***end loop***

5: While generating adversarial DDoS data and feed the data to IDS to test if it detects the attack.

6: ***for*** D-iterations, ***do***

7: receive detected labels from the ***IDS*** and sends a *critic* to the *Generator.*

Update the penalty using P_D function.

8: ***end loop***

9: ***end loop***

4.2 How the Generator fabricate an adversarial attack

In this section, I will specify the details about the learning process of the Generator and how it produces adversarial data.

If the generator continuously generates random data, the data will be unmeaningful, which can change the entire network flow data. So, the Generator needs to produce the data to maintain the intensity of an attack. To ensure that, we need to maintain the feature values constant that have higher SHAP values. As seen in Figures 7 and 8, the following are the features that need to be constant.

- *Fwd Packet Length Max* – Max packet size sent in a forward direction
- *Flow Duration* – Duration of the flow in milliseconds
- *Avg Fwd Segment Size* – Average segment size sent in a forward direction
- *Total Length of Fwd Packets* – Total length of packets sent in a forward direction
- *Bwd Packet Length Std* – Standard packet size sent in a backward direction
- *Average Packet Size* – Average size of a packet while in transmission
- *Avg Bwd Segment Size* – Average segment size sent backward direction
- *Packet Length Std* – Standard deviation of packet length
- *Flow IAT Std* – Standard deviation of inter-arrival time between two flows
- *ACK Flag Count* – Packets count with ACK
- *Bwd Packet Length Mean* – Mean of number of packets sent in a backward direction

Here is the sample of how the Generator produces an adversarial attack by the proposed technique. In this diagram, the darker shade explains the feature values of the features that are contributing to the attack. Whereas non highlighted values depict the feature value of a regular or non-attack feature.

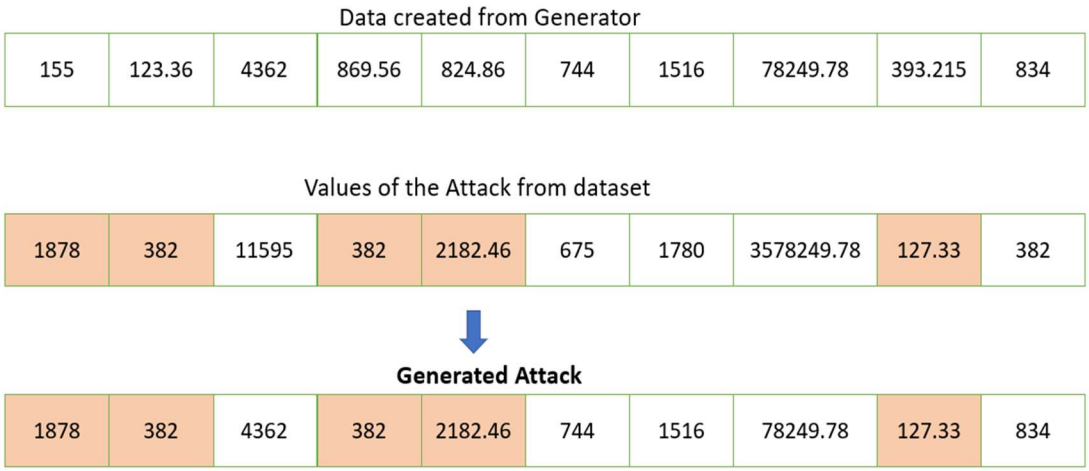

Figure 13: The process to generate adversarial DDoS attack

This figure explains that to maintain the intensity of the attack, and we need to keep that functional attack features static and only change the feature values that are not contributing to the attack. So to evade the black-box IDS, the generator changes the values of the features that are not contributing to the DDoS attack.

4.3 Training an IDS with the previously generated adversarial data

In this section, I will discuss the training of the IDS so that I can evaluate the performance of the IDS with the adversarial data. Following is the diagram that depicts the training process.

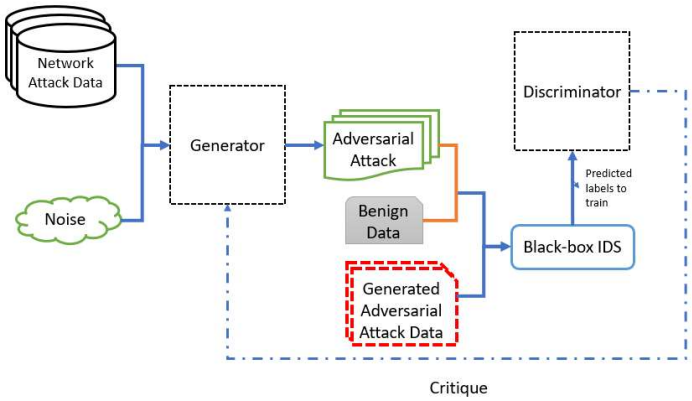

Figure 14: Training the Black-Box IDS

I considered three inputs to train the IDS: normal or benign data, new adversarial data, and previously generated adversarial data. The IDS learns about the adversarial data and tries to detect the DDoS attack data. Algorithm 2 suggests the overall process for the same.

Algorithm 2: Training IDS with Adversarial DDoS data

Input:

Generator – N noise + Original Attack Data

IDS – Benign or Normal Data, Adversarial Data, and Previously Generated Adversarial Data

Critic / *Discriminator* – S_{attack} and S_{benign}

Output:

Critic / Discriminator, Generator, and trained IDS

1: ***for*** epochs = 1 , … , MAX EPOCHS ***do***

2: ***for*** G-iterations, ***do***

3: Generator creates adversarial network attacks using S_{attack}

Update loss using P_G function

4: ***end loop***

5: ***for*** D-iterations, ***do***

6: Critic / Discriminator classifies the network data to

B_{benign} and B_{attack}

7: Update loss using P_D function

8: Feed B_{attack} (Adversarial data) and Previously Generated Adversarial Data

9: ***end loop***

10: ***end loop***

4.4 Polymorphic Engine to generate Polymorphic Attack

This section will discuss different methods to update the feature profile of the attack that generates polymorphic adversarial DDoS attacks.

Three different methods used for the Polymorphic engine are as follows.

1) Update new features in the attack profile after the IDS detects previous adversarial attacks. Algorithm 3 will discuss the process.

Algorithm 3:

Input – Use five functional attack features with a high impact score from the shortlisted features and five normal features.
1: Generate adversarial DDoS data and attack the IDS.
2: Train the IDS so that it can detect previously generated adversarial DDoS data.
3: Use the same set of features to generate an adversarial DDoS attack. Again, go to step – 2. If the Generator fails to evade the IDS, choose one functional feature with a high feature score, one normal or benign feature from the predefined set of features, and swap them with the used features.
4: Go to step – 1.
5: In the end, the IDS will detect all the Polymorphic adversarial DDoS attacks; the program will stop.

---

2) Add new features from the predefined list of features in the current attack profile after the IDS detects previous adversarial attacks, and the following algorithm will discuss the process.

Algorithm 4:

Input – Use five functional attack features with a high impact score from the shortlisted features and five normal features.
***1*:** Generate adversarial DDoS data and attack the IDS.
2: Train the IDS so that it can detect previously generated adversarial DDoS data.
3: Use the same set of features to generate an adversarial DDoS attack. Again, go to step – 2. If the Generator cannot deceive the IDS with the same set of features, choose

one new functional feature with a high impact score, one feature that represents benign data, and add them to the previous attack profile.

4: Go to step – 1.

5: At the end, the IDS will detect all the Polymorphic adversarial DDoS attacks. The program will stop.

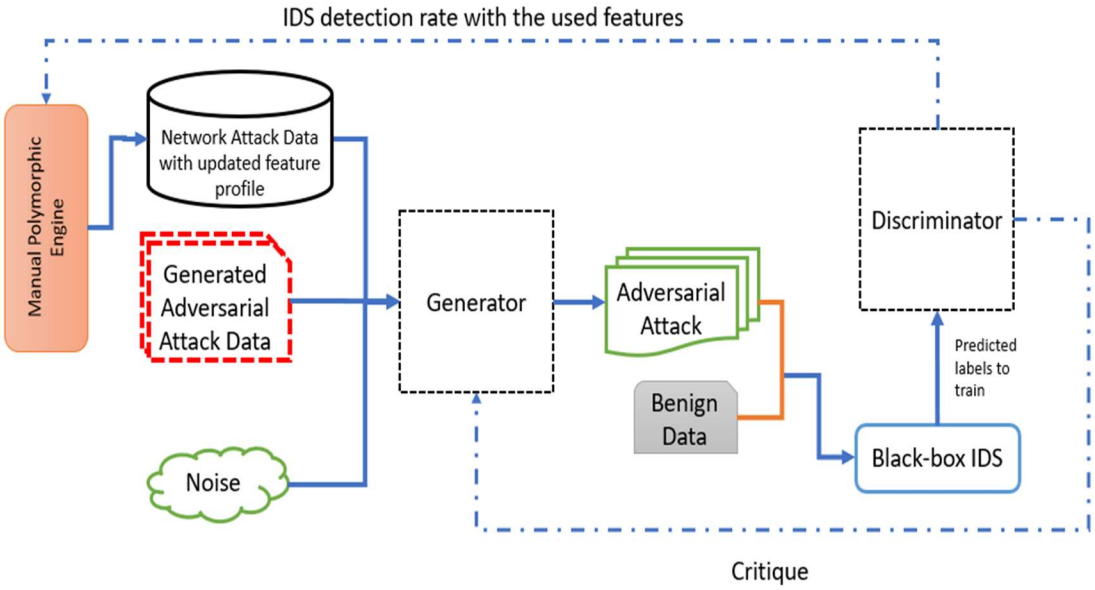

Figure 15: Manual process to generate Polymorphic adversarial attack

In the above methods, I assumed that an attacker would modify the feature profile manually and train the model with the new feature profile every time after the ISD detects a polymorphic attack. I considered using only a total of 20 features that were provided by the SHAP method.

3) It will be challenging to keep manually changing the feature profile if we want to use more than 20 features. So as an alternative method, I used a Reinforcement Learning method to automate the feature profile selection for generating a polymorphic attack.

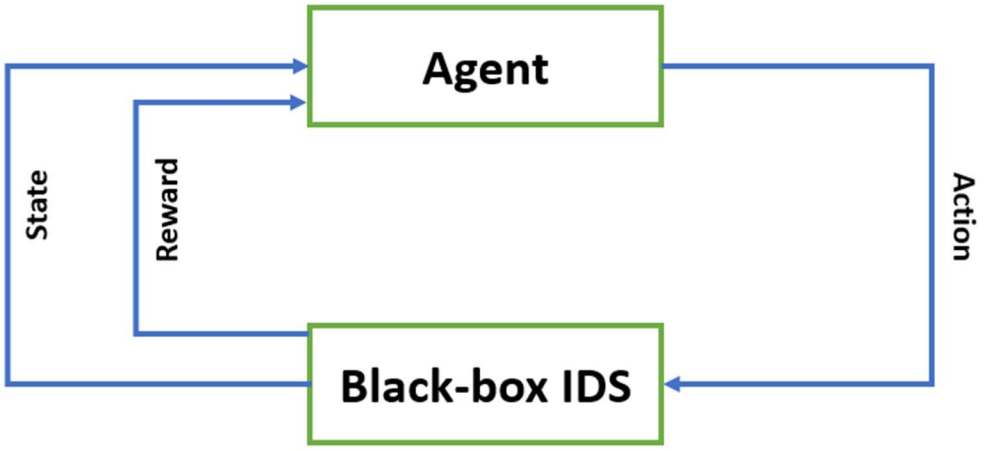

Figure 16: Function of RL in this framework

The Reinforcement Learning method is an ML-based technique that focuses on retraining the algorithm following a trial-and-error approach. The agent in this architecture evaluates the current IDS attack detection score. Then the agent takes action and receives feedback from IDS. Positive feedback is a reward, and negative feedback is a penalty to the agent. The following algorithm will explain the process. The overall process of generating a polymorphic attack is explained in the following algorithm 5.

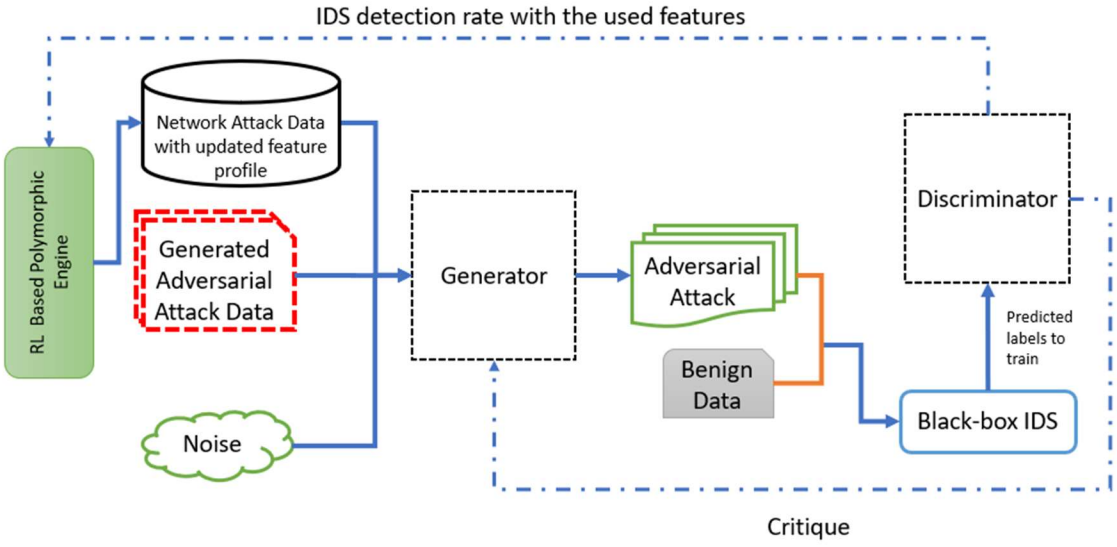

Figure 17: Automated RL that generates Polymorphic adversarial attack

Algorithm 5:

Input – Use any five features with a high impact score and any 5 with the lowest score from the shortlisted features.

1**:** Generate adversarial DDoS data and attack the IDS.

2: Train the IDS and check if the adversarial attack evades the IDS. Continue using the current feature set to generate an attack.

3: Get the attack success rate; if the attack ***FAILS to evade****,* The RL algorithm adds new features in the existing feature set to generate a polymorphic attack.

4: If the new polymorphic attack fails to evade the IDS, the RL algorithm will get a ***penalty***. The RL will ignore these features, and if the new polymorphic attack evades the IDS, the RL will get a ***reward.***

5: The RL **agent** will learn combinations of the attack feature profile and generate a new polymorphic adversarial DDoS attack.

6: The algorithm stops when the Generator can no longer generate a polymorphic adversarial attack.

Chapter 5. Experiment setup, Results, and Analysis

5.1 Experiment Setup

This section describes the libraries and hyper-parameters used in this research.

5.1.1 Libraries

The following are the libraries used in the overall program of this research.

PyTorch [52]

It is an open-source machine learning platform that is based on the Torch library. I used the PyTorch library to create neural networks for Black-box IDS, the Generator, and the Discriminator or critic. For example, to generator random noise, I have used a **torch.Tensor** method.

Scikit-learn [53]

It is a machine learning library for python that supports various classification, regression, and clustering techniques. Examples: sklearn.utils, sklearn.metrics.

Pandas [54]

A python library that is used to read, manipulate, and analyze the dataset. For example, to read CSV files, we use the read_csv() method from this library.

Numpy [55]

It is a library that provides a huge collection of mathematical functions used to format and process datasets.

Matplotlib [56]

A python library is used to plot mathematical graphs. In this research, I used this library to plot a detection rate of an IDS, accuracy of a model.

5.1.2 Hyper-parameters

Hyper-parameters are essential properties that define the characteristics of the training process of the machine learning model. They include a list of variables that explain the structure of a neural network. The following table depicts hyper-parameters that are used in this research.

Table 5: Hyper-parameters

#	HYPER-PARAMETER	DESCRIPTION
1	Batch_Size	Defines the number of samples to consider for one iteration.
2	learning_rate	Controls the weights of a neural network
3	Critic_Iters	Critic_iters for each Generator cycles
4	Optimizer	Methods used to update the attributes of the neural networks, e.g., Adam, Rmsprop, Adagrad
5	Epochs	A number of cycles pass through an entire dataset.

Here, Batch_size, epochs, learning_rate, ciritic_iters are optimization hyperparameters related to the optimization and training process of the model. In comparison, an optimizer is a model-specific hyperparameter.

5.1.3 Evaluation Metrics

To evaluate the performance and the results of this work, I used the following parameters.

- **Accuracy** – represents the fraction of precisely classified data in comparison to the total processed data. The formula to calculate accuracy is as follows

  $$\textbf{\textit{Accuracy}} = \frac{TP + TN}{TP + TN + FN + FP}$$

- **Precision** – a ratio between True Positive values and all the positive values received from the machine learning model.

  $$\textbf{\textit{Precision}} = \frac{TP}{TP + FP}$$

- **Recall** – a ratio between correctly detected samples over total sample data. It is also known as a ratio between True Positives and the sum of True Positives and False Negatives.

  $$\textbf{\textit{Recall}} = \frac{TP}{TP + FN}$$

- **F1-Score** – a calculation of a mean of precision and recall.

  $$\textbf{\textit{F1-Score}} = 2 \text{ X } \frac{precision \ X \ recall}{precision \ + \ recall}$$

5.2 Experiment Scenarios and Results

This section describes the results of various experiments for different scenarios and analyses of findings.

5.2.1 Adversarial attack generation

The first step of the research is to generate adversarial DDoS data that can evade the Black-box IDS. As seen in the graph initially, the Generator produces data that is unable to bypass the IDS. After training the Generator for 100 epochs, it learns to generate adversarial data to deceive the IDS.

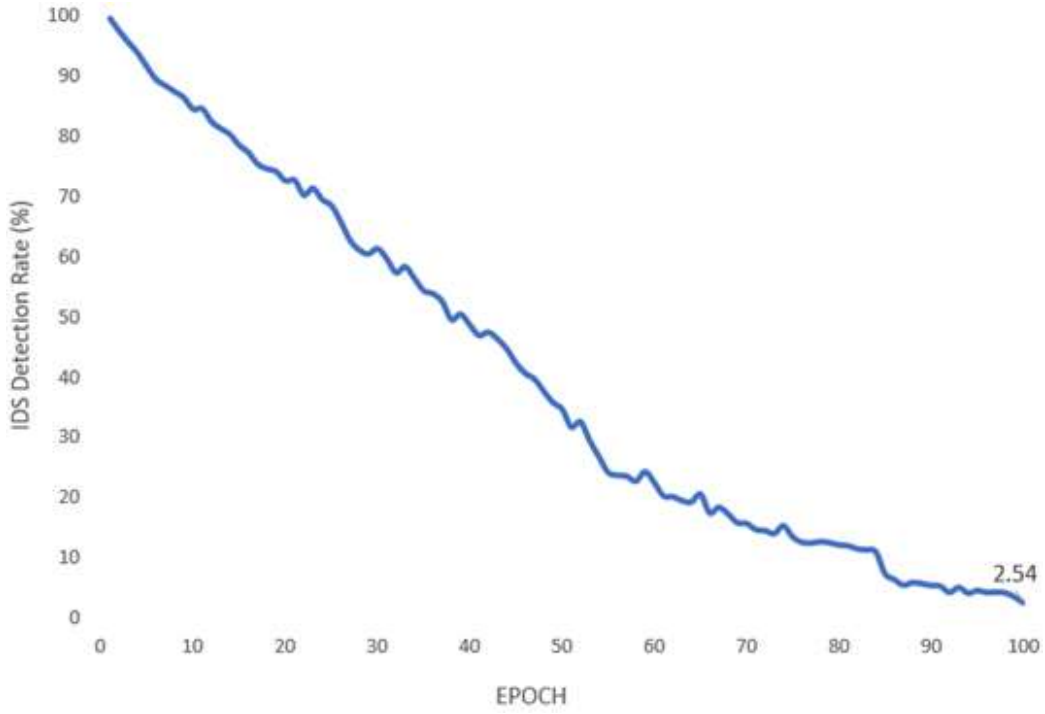

Figure 18: Adversarial DDoS Attack generation

5.2.2 Training IDS with the Adversarial DDoS data

The next step is to train the IDS with the previously generated adversarial DDoS data. Following is the result of the detection rate of the IDS after training. In the initial cycles, the IDS struggles to detect the attacks. After training it for 100 epochs, it detects almost all the attacks.

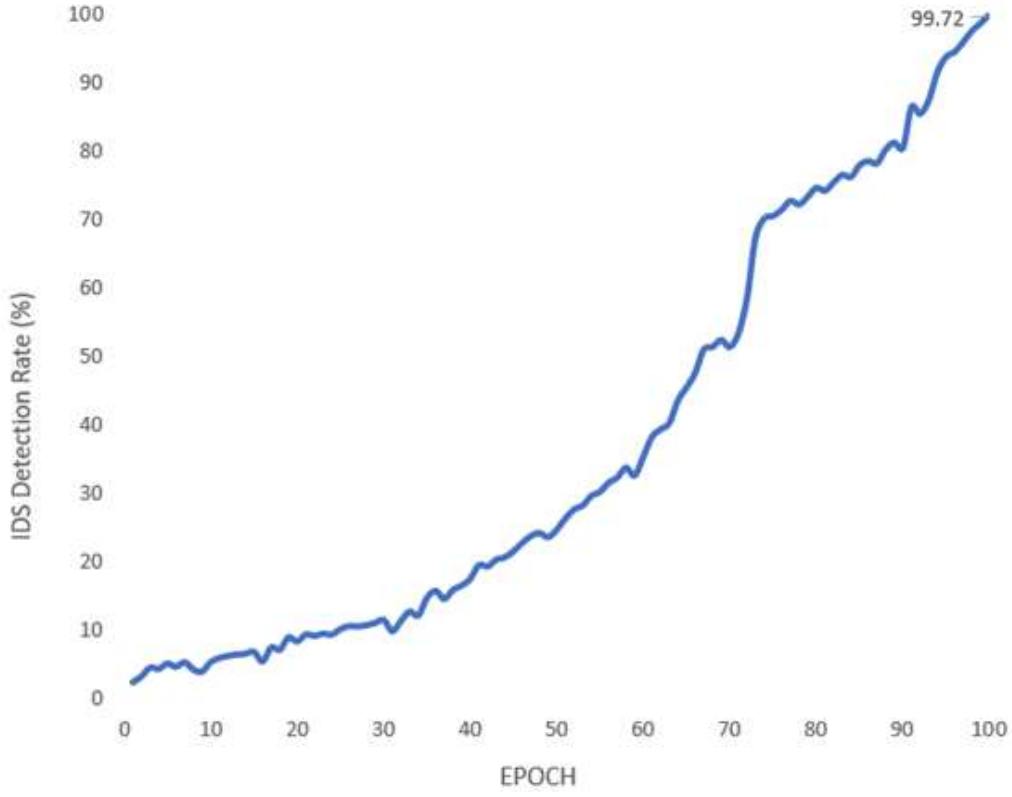

Figure 19: Detection rate after Training the IDS

5.2.3 Polymorphic adversarial DDoS attack generation

This section illustrates the detection rate of the Black Box IDS under the generation of polymorphic adversarial attacks.

In the first experiment, I manually selected new features to produce polymorphic attacks. For this test, I used only limited features from the dataset. The following is the initial result using algorithm 3.

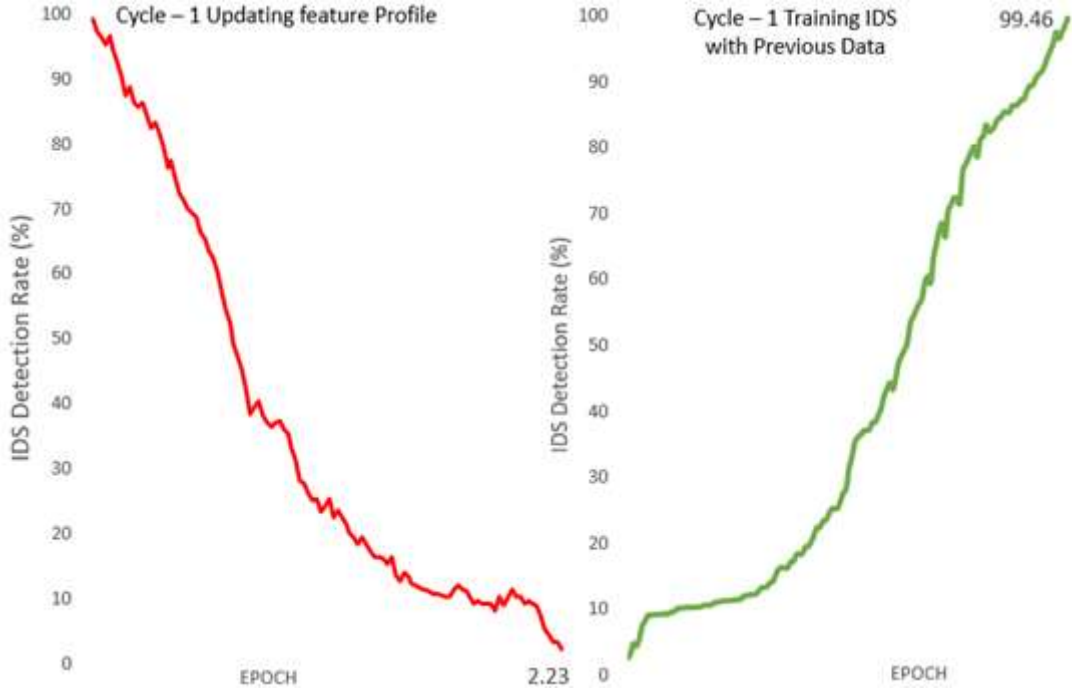

Figure 20: Polymorphic adversarial DDoS attack using Algorithm 3

In the above result, a ***red-colored graph*** suggests a ***polymorphic attack*** being generated and proceed towards the ***BlackBox IDS***. As seen, the polymorphic attack can deceive the IDS. The ***green-colored graph*** depicts the training of the IDS with the ***previously generated*** polymorphic adversarial DDoS data. A*fter 100 epochs,* the IDS detects the polymorphic adversarial DDoS attack. The following result indicates all the cycles of polymorphic attacks on the IDS. The Generator utilizes the same combination of the features to generate attacks until an IDS detects all the previous attacks.

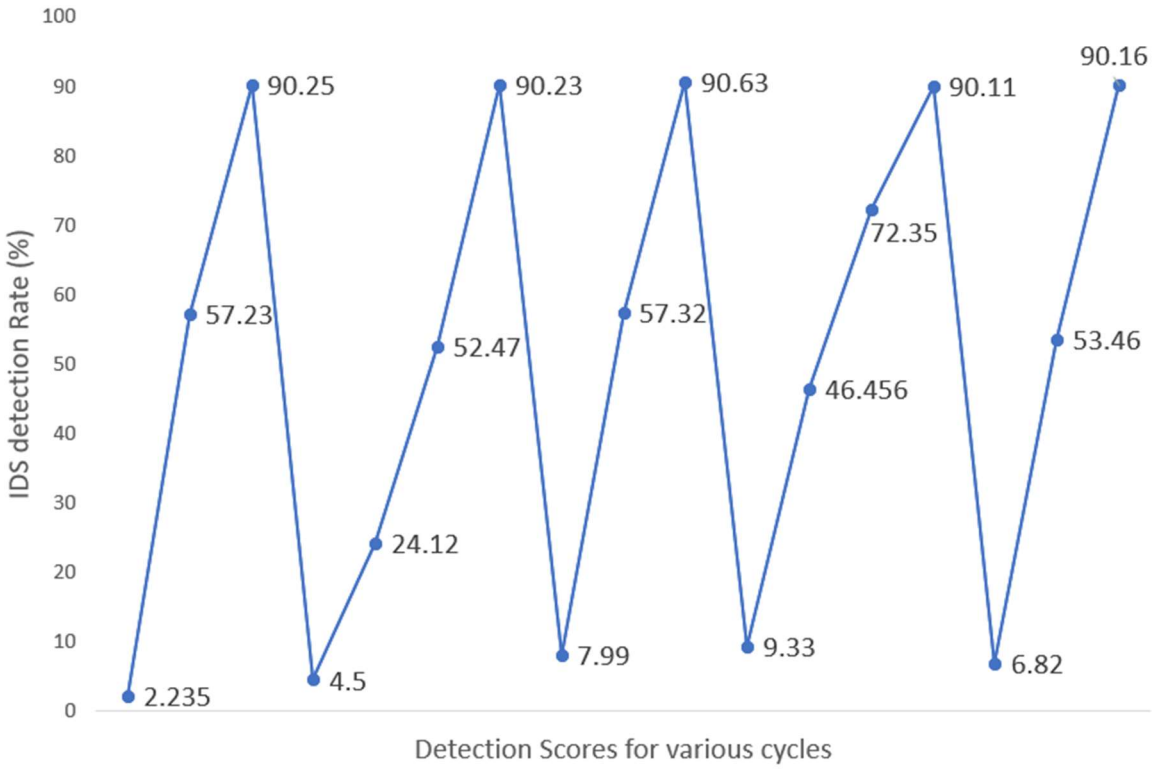

Figure 21: IDS detection rate for each attack cycle (using algorithm 3)

Each data point in figure 21 depicts the IDS detection rate. Once the IDS detects all the previous versions of the polymorphic DDoS attack that uses the same feature set (as seen in figure 21), the generator manually selects new predefined features and generates a new polymorphic adversarial DDoS attack. For this test, I used only a group of 10 features. In this test, the generator can evade the IDS up to 16 cycles, as seen in the appendix results B-1.

In the next test, I used a technique that follows algorithm 4 to update the feature profile of the attack to generate a polymorphic adversarial DDoS attack. For this experiment, I began with ten features to generate polymorphic attack data. To generate a new

polymorphic attack, I will add two new features in the existing attack data and used a total of 20 features. The following is the first result of the initial polymorphic attack.

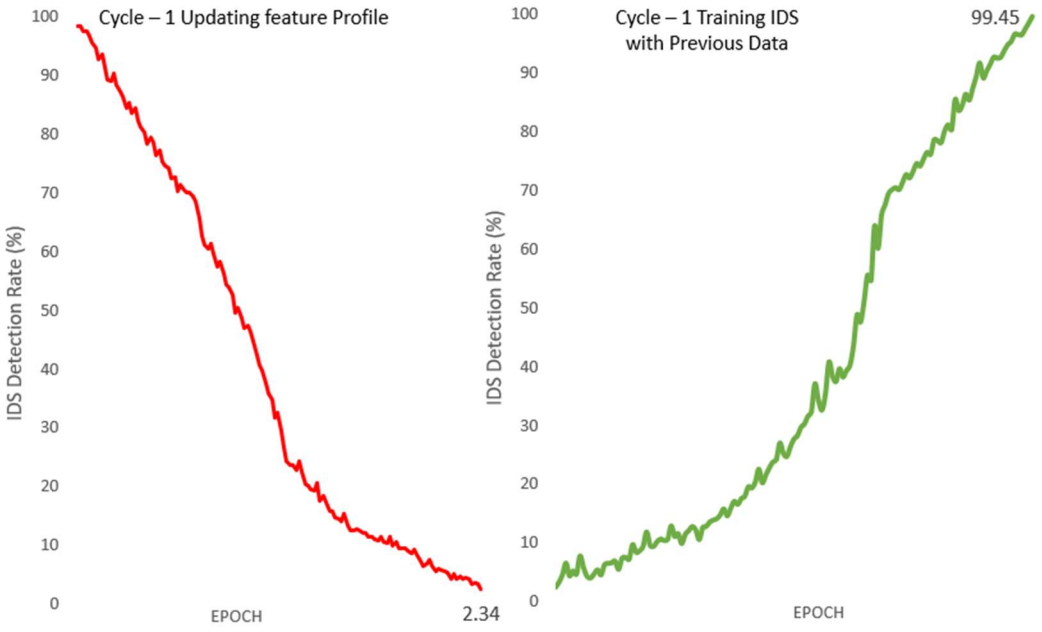

Figure 22: Polymorphic adversarial DDoS attack using Algorithm 4

Each data point in figure 23 depicts the IDS detection rate. Once the IDS detects all the previous versions of the polymorphic DDoS attack that uses the same feature set, the generator manually selects new predefined features and generates a new polymorphic adversarial DDoS attack. For this test, I have used a group of 20 features. In this test, the Generator can deceive the IDS for a total of 18 cycles using this technique.

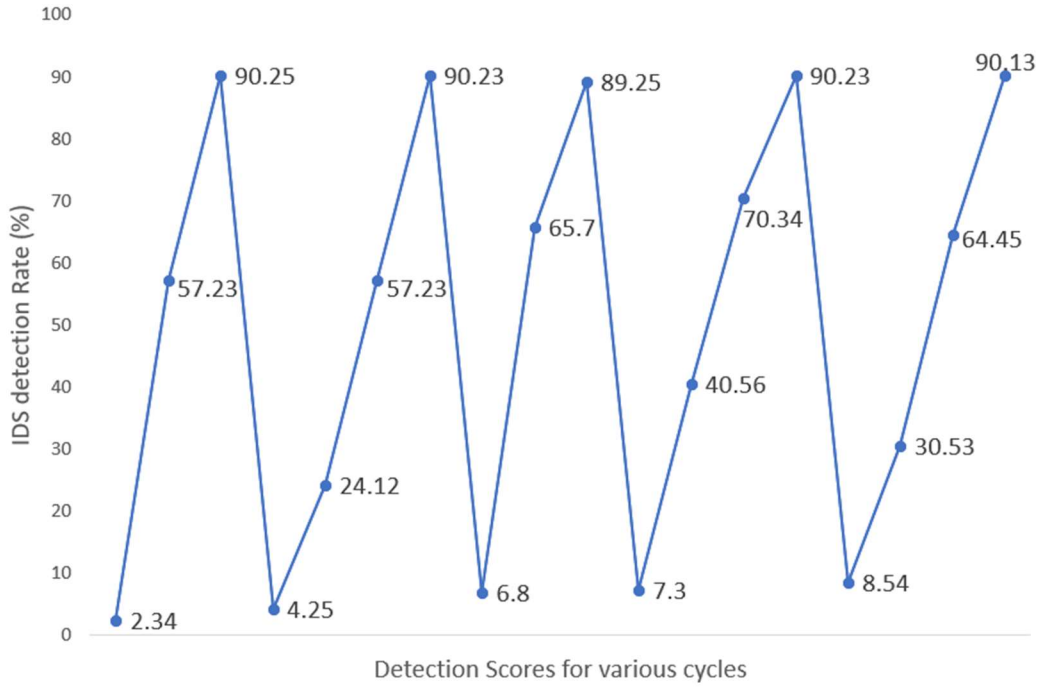

Figure 23: IDS detection rate for each attack cycle (using algorithm 4)

The first two experiments focus on testing if the Generator can produce polymorphic adversarial DDoS attack data by updating the feature profile manually. After confirming the possibility of doing so, the next step is to automatically select features and manipulate the attack feature profile to generate polymorphic adversarial attack data.

To automate this task, I applied the Reinforcement Learning technique. It receives an IDS detection rate and learns to select new features and add them to the old feature set and create a new feature set. This experiment also indicates the number of times a generator can produce polymorphic adversarial DDoS data. To examine this, I used four sets of feature combinations for each test to generate the automated Polymorphic adversarial DDoS attack.

- The first test includes a total of 40 features from the dataset

- The second test includes a total of 50 features from the dataset
- The third test includes a total of 60 features from the dataset
- The fourth test includes a total of 76 features from the dataset

The above experiments begin with ten features, from which 5 are a functional feature with a high impact score, and 5 are usual or benign.

The results of the above tests can be seen in Appendix B.

5.2.4 Test Evaluation

In this section, I have stated the overall values for the Precision, Recall, and F1-score for each test.

Table 6: Model Evaluation

#	TEST	ACCURACY	PRECISION	RECALL	F1-SCORE
1	Manual Test – 1 (using Algorithm 3)	98.58	96.24	92.91	0.953
2	Manual Test – 2 (using Algorithm 4)	98.08	95.22	92.15	0.946
3	Automated Test using 40 features (using Algorithm 5)	98.27	94.41	92.44	0.935
4	Automated Test using 50 features (using Algorithm 5)	96.97	93.58	91.69	0.928
5	Automated Test using 60 features (using Algorithm 5)	96.34	93.22	91.43	0.921

6	Automated Test using 76 features (using Algorithm 5)	94.42	91.79	91.58	0.916

5.3 Analysis

As mentioned earlier, I ran 6 test scenarios with different feature combinations. 2 experiments consist of a manual feature selection technique to generate polymorphic adversarial DDoS attack data and four tests with an Automated feature selection technique.

I utilized a manual feature selection technique as a benchmark and compared this technique to the automated feature selection technique to analyze for how many cycles the polymorphic attack evades the Black-Box IDS.

The following graphs will be useful to compare these six different scenarios.

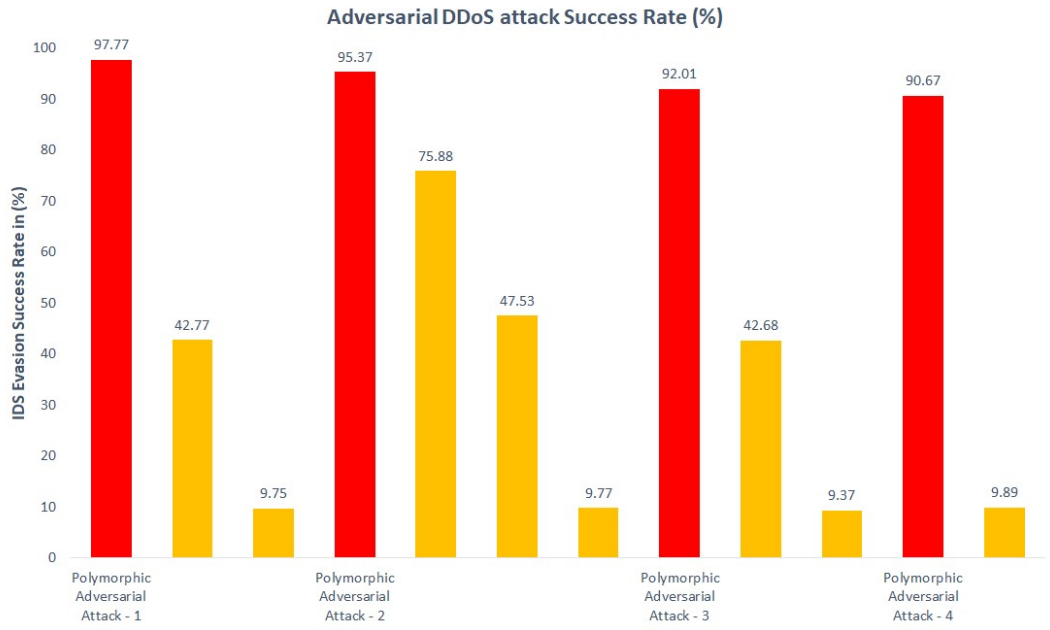

Figure 24: ***Test - 1*** *Polymorphic Adversarial attacks using* ***Manual feature selection***

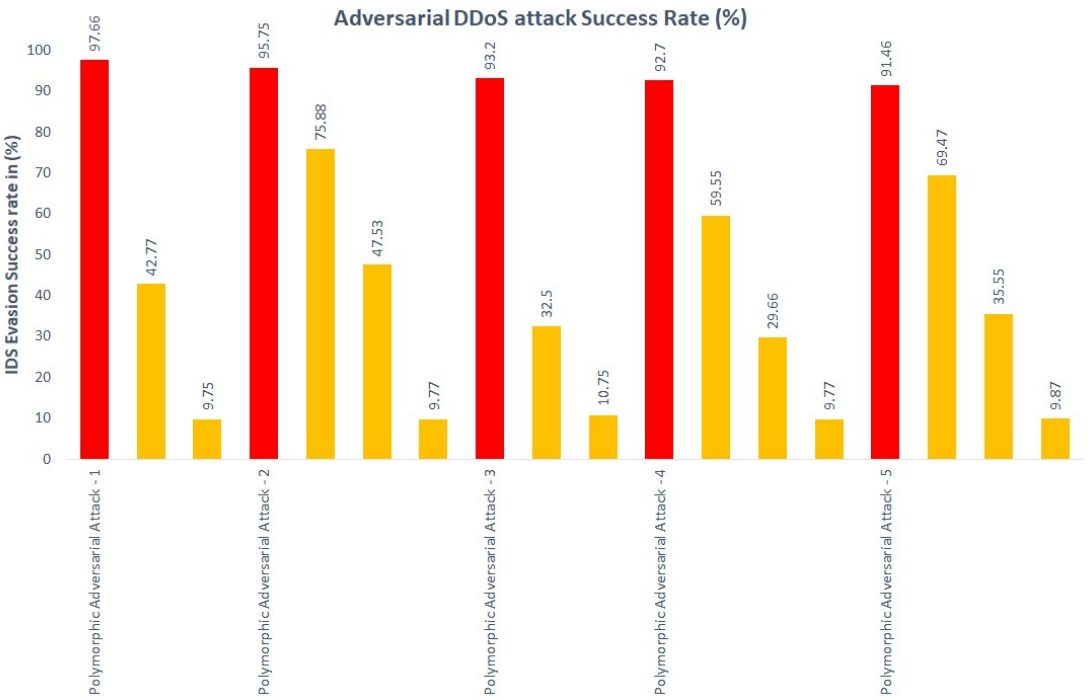

Figure 25: ***Test - 2*** *Polymorphic Adversarial attacks using* ***Manual feature selection***

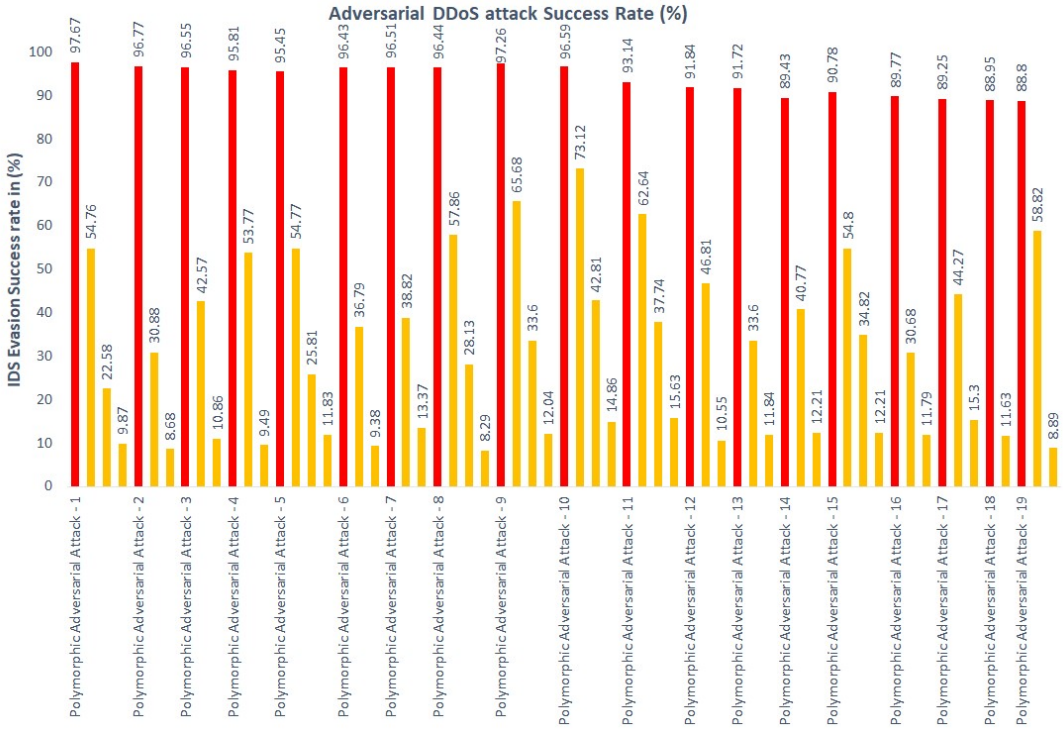

Figure 26: ***Test - 3*** *Polymorphic Adversarial attacks using* ***Automated feature selection***

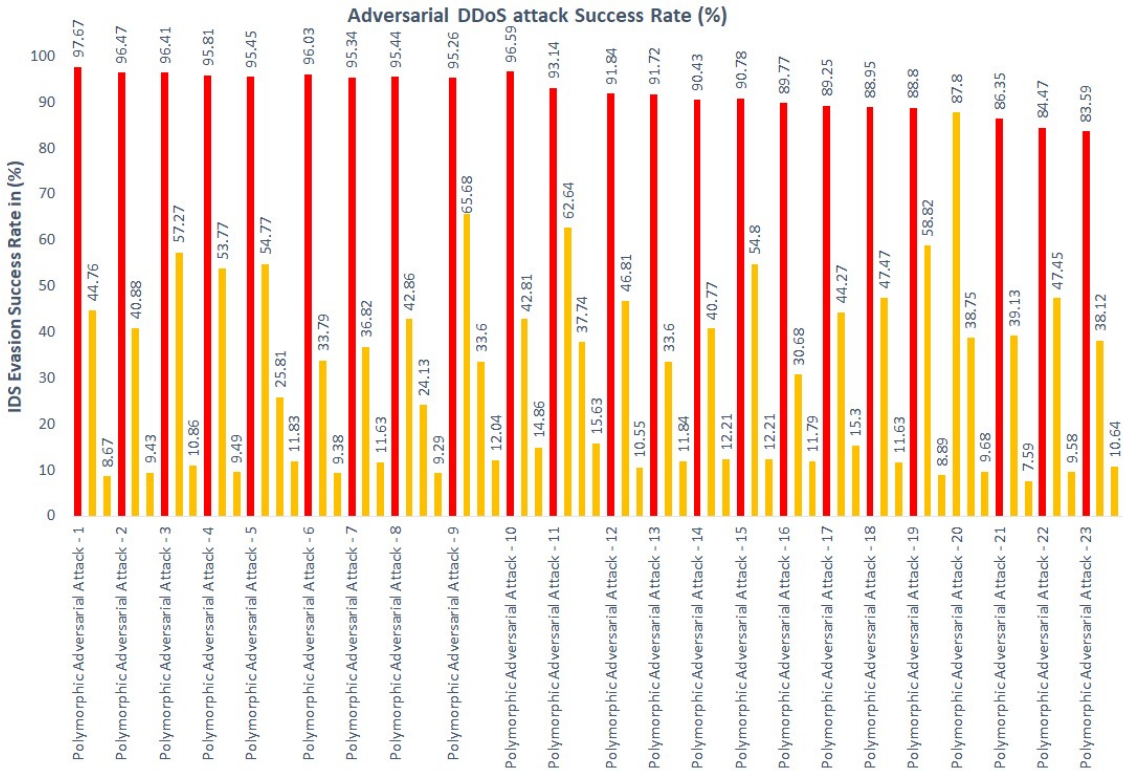

*Figure 27: **Test - 4** Polymorphic Adversarial attacks using **Automated feature selection***

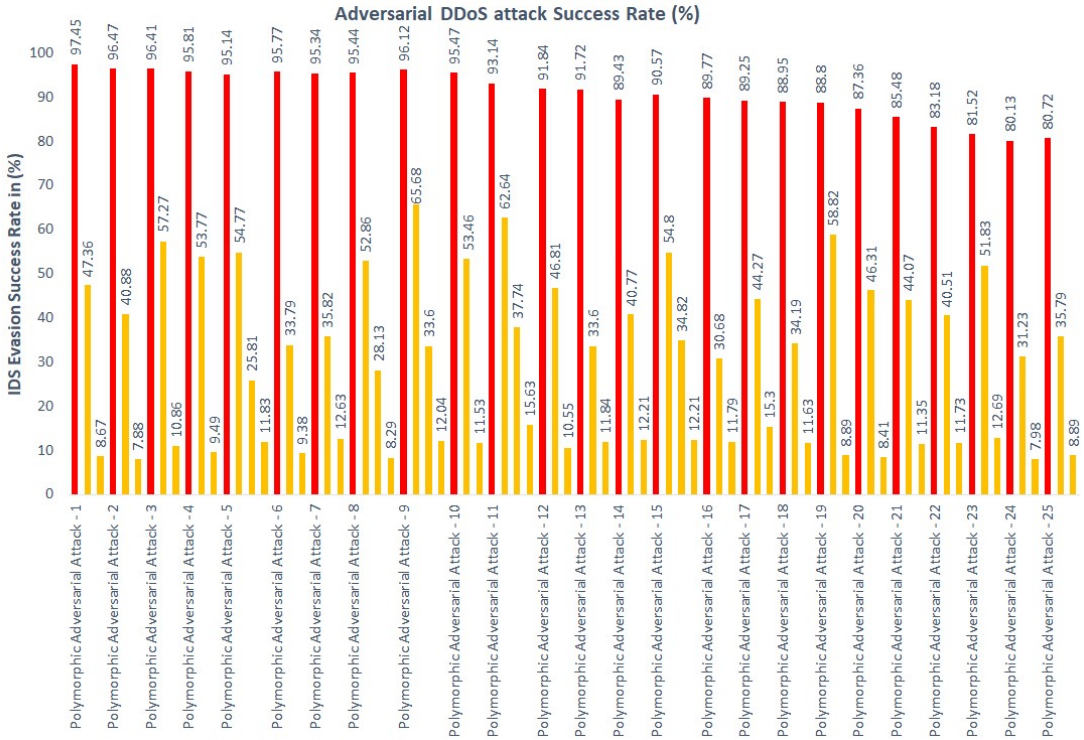

*Figure 28: **Test - 5** Polymorphic Adversarial attacks using **Automated feature selection***

Figure 29: ***Test - 6*** *Polymorphic Adversarial attacks using* ***Automated feature selection***

In all the above results, the Polymorphic DDoS adversarial attack successfully evading the IDS; the orange bar suggests the polymorphic attack is becoming weak once the IDS detects them. By counting the red bar, we can see how many times the Generator produced a polymorphic attack in each cycle.

Figures 25,26 suggest that when the Generator uses a small number of features, more than 90% of the polymorphic attack evades the IDS. By noticing these figures, it is clear that using fewer features to generate a polymorphic attack has a higher evasion rate but fewer chances of generating more polymorphic attacks.

Figures 27,28,29,30 suggest that initially, more than 90% of the polymorphic attacks can evade the IDS. However, results propose that if the Generator utilizes more features to generate a polymorphic DDoS attack, the success rate gets lower each time.

Comparing all the results confirms that while using a fewer number of features to generate polymorphic adversarial DDoS attacks, the attack success rate stays up to the acceptable amount. However, when we use more features, the attack success rate depletes after certain cycles.

Now the following table describes the total runtime for each experiment.

Table 7: Total runtime of each test

#	TEST	TOTAL RUNTIME
1	Test – 1 Manual Feature profile update (with a total of 10 features)	30.43 minutes
2	Test – 2 Manual Feature profile update (with a total of 20 features)	46.21 minutes
3	Test – 3 Automated Feature profile update (with a total of 40 features)	75.31 minutes
4	Test – 5 Automated Feature profile update (with a total of 50 features)	90.45 minutes
5	Test – 6 Automated Feature profile update (with a total of 60 features)	145.37 minutes
6	Test – 5 Automated Feature profile update (with a total of 76 features)	173.55 minutes

As observed from the above table, if the test uses a small number of features, it takes less time to run the simulation. The run time rises upon increasing features to generate a polymorphic DDoS attack.

Chapter 6. Conclusions and Future work

With this work, I proposed a framework to generate polymorphic adversarial DDoS attacks using a CICIDS2017 dataset using a Wasserstein GAN. To generate polymorphic attacks, I proposed three different techniques that change the feature profile of the attack. In the first two techniques, I have selected new features manually each time to generate polymorphic adversarial attacks. Furthermore, to automate the feature selection to generate polymorphic attacks, I applied a Reinforcement Learning technique in each technique; the Generator creates a polymorphic attack until no more new features are remaining to choose from the feature set.

From the results, I have demonstrated that the Generator can produce polymorphic adversarial DDoS. Results also depict that while using a small number of features to create a polymorphic attack, the attacks were successfully deceiving the IDS with more than a 90% success rate while using a manual selection of features. However, when I utilized more than 40 features to generate polymorphic attacks, the evasion rate went down, and only 70% of attacks deceived the IDS at the end. One more thing I have noticed in this research is that using more sets of features takes more time to generate polymorphic adversarial DDoS attacks.

In the future, it could be interesting to consider using other variants of GAN like DCGAN [57], Conditional GAN [58], BiGAN [59], Cycle GAN [60] to generate adversarial network attack data and evaluate the detection systems. Another limitation of this research is that it focused on generating only one type of attack, as every attack has different functional features. It would be difficult to use one Generator to create other types

of attacks with the same generator. So it would be interesting to use multiple generators for each type of attack and evaluate the performance of the IDS against all types of polymorphic adversarial network attacks.

The focus of this research is only to generate polymorphic attacks using a GAN that can deceive the IDS. In the future, it would be interesting to use a similar methodology for the detection system and see the overall result of how a Black Box IDS responds to an unknown, polymorphic adversarial attack without being retrained.

REFERENCES

[1] V. Wang, M. Button, F. K, Motha, S. J, and W. Y, Cyber Security Breaches Survey 2018.

[2] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, “Intrusion detection system: A comprehensive review,” Journal of Network and Computer Applications, vol. 36, no. 1, pp. 16–24, Jan. 2013, DOI: 10.1016/j.jnca.2012.09.004.

[3] U. Sabeel, S. S. Heydari, H. Mohanka, Y. Bendhaou, K. Elgazzar and K. El-Khatib, "Evaluation of Deep Learning in Detecting Unknown Network Attacks," 2019 International Conference on Smart Applications, Communications and Networking (SmartNets), Sharm El Sheik, Egypt, 2019, pp. 1-6, doi: 10.1109/SmartNets48225.2019.9069788.

[4] M. Gadelrab, A. A. El Kalam, and Y. Deswarte, “Manipulation of Network Traffic Traces for Security Evaluation,” in 2009 International Conference on Advanced Information Networking and Applications Workshops, May 2009, pp. 1124–1129, DOI: 10.1109/WAINA.2009.36.

[5] F. Skopik, G. Settanni, R. Fiedler, and I. Friedberg, “Semi-synthetic data set generation for security software evaluation,” in 2014 Twelfth Annual International Conference on Privacy, Security and Trust, Jul. 2014, pp. 156–163, DOI: 10.1109/PST.2014.6890935.

[6] “CrowdStrike Introduces Enhanced Endpoint Machine Learning Capabilities and Advanced Endpoint Protection Modules.” Internet: https://www.crowdstrike.com/resources/news/crowdstrike-introduces-enhanced-endpoint-machine-learning-capabilities-and-advanced-endpoint-protection-modules

[7] M. Berninger and A. Sopan. “Reverse Engineering the Analyst: Building Machine Learning Models for the SOC.” Internet: https://www.fireeye.com/blog/threat-research/2018/06/buildmachine-learning-models-for-the-soc.html

[8] “Use Cases: Demisto’s Top Machine Learning Use Cases – Part 1.” Internet: https://blog.demisto.com/demistos-top-machine-learning-use-cases-part-1

[9] “Big Data Analytics for Advanced Security.” Internet: https://logrhythm.com/solutions/security/security-analytics

[10] “Cognito Detect is the most powerful way to find and stop cyberattackers in real time.” Internet: https://content.vectra.ai/rs/748MCE447/images/ProductCompanyOverview_2019_Cognito_Detect_AIpowered_attacker_detection_English.pdf

[11] T. Park, D. Cho, and H. Kim, “An Effective Classification for DoS Attacks in Wireless Sensor Networks,” in 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), Jul. 2018, pp. 689–692, DOI: 10.1109/ICUFN.2018.8436999.

[12] S. Bhattacharya and S. Selvakumar, “SSENet-2014 Dataset: A Dataset for Detection of Multiconnection Attacks,” in 2014 3rd International Conference on Eco-friendly

Computing and Communication Systems, Dec. 2014, pp. 121–126, DOI: 10.1109/Eco-friendly.2014.100.

[13] I. Sharafaldin, A. Lashkari, and A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

[14] I. Goodfellow et al., “Generative Adversarial Nets,” in Advances in Neural Information Processing Systems 27, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[15] S. Yu, H. Dong, F. Liang, Y. Mo, C. Wu, and Y. Guo, “SIMGAN: Photo-Realistic Semantic Image Manipulation Using Generative Adversarial Networks,” in 2019 IEEE International Conference on Image Processing (ICIP), Sep. 2019, pp. 734–738, DOI: 10.1109/ICIP.2019.8804285.

[16] C. Wan, S. Chuang, and H. Lee, "Towards Audio to Scene Image Synthesis Using Generative Adversarial Network," ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Brighton, United Kingdom, 2019, pp. 496-500, DOI: 10.1109/ICASSP.2019.8682383.

[17] Y. Yang, X. Dan, X. Qiu, and Z. Gao, "FGGAN: Feature-Guiding Generative Adversarial Networks for Text Generation," in IEEE Access, vol. 8, pp. 105217-105225, 2020, DOI: 10.1109/ACCESS.2020.2993928.

[18] J. Zhang, Q. Yan, and M. Wang, “Evasion Attacks Based on Wasserstein Generative Adversarial Network,” *2019 Computing, Communications and IoT Applications (ComComAp)*, 2019, doi: 10.1109/ComComAp46287.2019.9018647.

[19] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, “Polymorphic Blending Attacks,” Jan. 2006.

[20] R. Best, “How AI is Leading to More Business Phishing Attacks.” https://www.infotech.co.uk/blog/how-ai-is-leading-to-more-business-phishing-attacks

[21] A. M. on January 7 and 2020, “Hacking the Hackers: Adversarial AI and How to Fight It,” Security Boulevard, Jan. 07, 2020. https://securityboulevard.com/2020/01/hacking-the-hackers-adversarial-ai-and-how-to-fight-it/

[22] G. Yaltirakli, gkbrk/slowloris, Internet: https://github.com/gkbrk/slowloris.

[23] J. Seidl, jseidl/GoldenEye, Internet: https://github.com/jseidl/GoldenEye.

[24] D. | Mr4FX, Mr4FX/Hulk-ddos-attack. Internet: https://github.com/Mr4FX/Hulk-ddos-attack

[25] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” *arXiv:1701.07875 [cs, stat]*, Dec. 2017, Accessed: Aug. 19, 2020. [Online]. Available: http://arxiv.org/abs/1701.07875.

[26] O. Yavanoglu and M. Aydos, “A Review on Cyber Security Datasets for Machine Learning Algorithms,” Dec. 2017, DOI: 10.1109/BigData.2017.8258167.

[27] “CSIC 2010 HTTP Dataset in CSV Format (for Weka Analysis),” Peter Scully PhD, May 15, 2018. https://petescully.co.uk/research/csic-2010-http-dataset-in-csv-format-for-weka-analysis/.

[28] C.-C. Li, A. Guo, and D. Li, “Application Research of Support Vector Machine in Network Security Risk Evaluation,” in 2008 International Symposium on Intelligent Information Technology Application Workshops, Dec. 2008, pp. 40–43, DOI: 10.1109/IITA.Workshops.2008.91.

[29] Z. Guan, L. Bian, T. Shang, and J. Liu, “When Machine Learning meets Security Issues: A survey,” in *2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, Aug. 2018, pp. 158–165, DOI: 10.1109/IISR.2018.8535799.

[30] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, “A Detailed Investigation and Analysis of Using Machine Learning Techniques for Intrusion Detection,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 686–728, Firstquarter 2019, DOI: 10.1109/COMST.2018.2847722.

[31] K. Ali and R. Boutaba, “Applying kernel methods to anomaly based intrusion detection systems,” in 2009 Global Information Infrastructure Symposium, Jun. 2009, pp. 1–4, DOI: 10.1109/GIIS.2009.5307054.

[32] A. Dawoud, S. Shahristani, and C. Raun, “Deep Learning for Network Anomalies Detection,” in 2018 International Conference on Machine Learning and Data Engineering (iCMLDE), Dec. 2018, pp. 149–153, DOI: 10.1109/iCMLDE.2018.00035.

[33] C.-F. Tsai, Y.-F. Hsu, C.Y. Lin and W.-Y. Lin, "Intrusion detection by machine learning: a review", Expert Systems with Applications, vol. 36, no. 10, pp. 11994-12000, 2009.

[34] L. Haripriya and M. A. Jabbar, “Role of Machine Learning in Intrusion Detection System: Review,” in 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), Mar. 2018, pp. 925–929, DOI: 10.1109/ICECA.2018.8474576.

[35] S. Kumar and A. Bhatia, “Detecting Domain Generation Algorithms to prevent DDoS attacks using Deep Learning,” in *2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec. 2019, pp. 1–4, DOI: 10.1109/ANTS47819.2019.9118156.

[36] A. M. V. Bharathy, N. Umapathi, and S. Prabaharan, “An Elaborate Comprehensive Survey on Recent Developments in Behaviour Based Intrusion Detection Systems,” in 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Feb. 2019, pp. 1–5, DOI: 10.1109/ICCIDS.2019.8862119.

[37] B. Senthilnayaki, K. Venkatalakshmi, and A. Kannan, “Intrusion detection using optimal genetic feature selection and SVM based classifier,” in 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), Mar. 2015, pp. 1–4, DOI: 10.1109/ICSCN.2015.7219890.

[38] Md. O. Miah, S. Shahriar Khan, S. Shatabda, and D. Md. Farid, “Improving Detection Accuracy for Imbalanced Network Intrusion Classification using Cluster-based Under-sampling with Random Forests,” in 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), May 2019, pp. 1–5, DOI: 10.1109/ICASERT.2019.8934495.

[39] M. Kawai, K. Ota, and M. Dong, "Improved MalGAN: Avoiding Malware Detector by Leaning Cleanware Features," in 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Feb. 2019, pp. 040–045, DOI: 10.1109/ICAIIC.2019.8669079.

[40] M. Shahpasand, L. Hamey, D. Vatsalan, and M. Xue, "Adversarial Attacks on Mobile Malware Detection," in 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile), Feb. 2019, pp. 17–20, DOI: 10.1109/AI4Mobile.2019.8672711.

[41] H. Xie, K. Lv, and C. Hu, "An Effective Method to Generate Simulated Attack Data Based on Generative Adversarial Nets," in 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), Aug. 2018, pp. 1777–1784, DOI: 10.1109/TrustCom/BigDataSE.2018.00268.

[42] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based Network Traffic Generation using Generative Adversarial Networks," Computers & Security, vol. 82, pp. 156–172, May 2019, DOI: 10.1016/j.cose.2018.12.012.

[43] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection," *arXiv:1809.02077 [cs]*, Jun. 2019, Available: http://arxiv.org/abs/1809.02077.

[44] J. Hui, "GAN — DCGAN (Deep convolutional generative adversarial networks)," Medium, Jun. 24, 2018. https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f

[45] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774.

[46] "Overview of GAN Structure | Generative Adversarial Networks." https://developers.google.com/machine-learning/gan/gan_structure (accessed Oct. 26, 2020).

[47] Z. Zhang, M. Li, and J. Yu, "On the convergence and mode collapse of GAN," in SIGGRAPH Asia 2018 Technical Briefs on - SA '18, Tokyo, Japan, 2018, pp. 1–4, doi: 10.1145/3283254.3283282.

[48] L. Horsley and D. Perez-Liebana, "Building an automatic sprite generator with deep convolutional generative adversarial networks," in 2017 IEEE Conference on Computational Intelligence and Games (CIG), Aug. 2017, pp. 134–141, DOI: 10.1109/CIG.2017.8080426.

[49] C. Zhang, C. Xiong, and L. Wang, "A Research on Generative Adversarial Networks Applied to Text Generation," in 2019 14th International Conference on Computer Science Education (ICCSE), Aug. 2019, pp. 913–917, DOI: 10.1109/ICCSE.2019.8845453.

[50] J. Brownlee, "How to Implement Wasserstein Loss for Generative Adversarial Networks," Machine Learning Mastery, Jul. 14, 2019.

https://machinelearningmastery.com/how-to-implement-wasserstein-loss-for-generative-adversarial-networks.
[51] R. Shaikh, “Feature Selection Techniques in Machine Learning with Python,” Medium, 28-Oct-2018. https://towardsdatascience.com/featureselection-techniques-in-machine-learning-with-python-f24e7da3f36e
[52] “torch — PyTorch 1.6.0 documentation.” https://pytorch.org/docs/stable/torch.html
[53] “scikit-learn: machine learning in Python – scikit-learn 0.23.2 documentation.” https://scikit-learn.org/stable/
[54] “pandas documentation – pandas 1.1.2 documentation.” https://pandas.pydata.org/docs/
[55] “Overview – NumPy v1.19 Manual.” https://numpy.org/doc/stable/
[56] “Overview — Matplotlib 3.3.2 documentation.” https://matplotlib.org/contents.html
[57] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” arXiv:1511.06434 [cs], Jan. 2016, Available: http://arxiv.org/abs/1511.06434.
[58] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” arXiv:1411.1784 [cs, stat], Nov. 2014, Available: http://arxiv.org/abs/1411.1784.
[59] U. Mutlu and E. Alpaydın, “Training bidirectional generative adversarial networks with hints,” Pattern Recognition, vol. 103, p. 107320, Jul. 2020, doi: 10.1016/j.patcog.2020.107320.
[60] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks,” in 2017 IEEE International Conference on Computer Vision (ICCV), Oct. 2017, pp. 2242–2251, doi: 10.1109/ICCV.2017.244.

Appendices

Appendix A. (Feature Description of CICIDS2017)

Table 8: Detailed description of features

#	Feature	Description
1	Flow Duration	Duration of the Flow in milliseconds
2	Total Fwd Packets	Total packet flow in a forward direction
3	Total Bwd Packets	Total packet flow in a backward direction
4	Total len of Fwd Packet	Total length of the packet in a forward direction
5	Total len of Bwd Packet	Total length of the packet in a backward direction
6	Fwd Packet Length Max	Max length of the packet in a forward direction
7	Fwd Packet Length Min	Min length of the packet in a forward direction
8	Fwd Packet Length Mean	Mean of packet length in a forward direction
9	Fwd Packet Length Std	The standard deviation of packet length in a forward direction
10	Bwd Packet Length Max	Max length of the packet in a backward direction
11	Bwd Packet Length Min	Min length of the packet in a backward direction

12	Bwd Packet Length Mean	Mean of packet length in a backward direction
13	Bwd Packet Length Std	The standard deviation of packet length in a backward direction
14	Flow Bytes/s	Number of bytes flow per second
15	Flow Packets/s	Number of packet flows per second
16	Flow IAT Mean	Mean of Inter-arrival Time between two flow
17	Flow IAT Std	The standard deviation of inter-arrival time between two flow
18	Flow IAT Max	Max inter-arrival time between two flow
19	Flow IAT Min	Min inter-arrival time between two flow
20	Fwd IAT Total	Total inter-arrival time between 2 packets in a forward direction
21	Fwd IAT Mean	Mean of inter-arrival time between 2 packets in a forward direction
22	Fwd IAT Std	The standard deviation of inter-arrival time between two packets in a forward direction
23	Fwd IAT Max	Max inter-arrival time between 2 packets in a forward direction
24	Fwd IAT Min	Min inter-arrival time between 2 packets in a forward direction
25	Bwd IAT Total	Total inter-arrival time between 2 packets in a backward direction
26	Bwd IAT Mean	Mean of inter-arrival time between 2 packets in a backward direction

27	Bwd IAT Std	The standard deviation of inter-arrival time between 2 packets in a backward direction
28	Bwd IAT Max	Max inter-arrival time between 2 packets in a backward direction
29	Bwd IAT Min	Min inter-arrival time between 2 packets in a backward direction
30	Fwd PSH Flags	No of PSH Flags set in packets in a forward direction
31	Bwd PSH Flags	No of PSH Flags set in packets in a backward direction
32	Fwd URG Flags	No of URG Flags set in packets in a forward direction
33	Bwd URG Flags	No of URG Flags set in packets in a backward direction
34	Fwd Header Length	Byte length of a header sent in a forward direction
35	Bwd Header Length	Byte length of a header sent in a backward direction
36	Fwd Packets/s	Packet length sent in forward direction per second
37	Bwd Packets/s	Packet length sent in backward direction per second
38	Min Packet Length	Min packet length in a flow
39	Max Packet Length	Max packet length in a flow
40	Packet Length Mean	Average packet length per-flow

41	Packet Length Std	The standard deviation of packet length per-flow
42	Packet Len. Variance	Arrival time of the packet
43	FIN Flag Count	Packets with a FIN flag
44	SYN Flag Count	Packets with SYN flag
45	RST Flag Count	Packets with RST flag
46	PSH Flag Count	Packets with PSH flag
47	ACK Flag Count	Packets with ACK flag
48	URG Flag Count	Packets with URG flag
49	CWE Flag Count	Packets with CWE flag
50	ECE Flag Count	Packets with ECE flag
51	Down/Up Ratio	The ratio of download and upload
52	Average Packet Size	The average size of a packet
53	Avg Fwd Segment Size	Average of segment size in a forward direction
54	Avg Bwd Segment Size	Average of segment size in a backward direction
55	Fwd Avg Bytes/Bulk	Bytes/Bulk average sent in a forward direction
56	Fwd Avg Packets/Bulk	Packets/Bulk average sent in a forward direction
57	Fwd Avg Bulk Rate	Bulk rate average in a forward direction
58	Bwd Avg Bytes/Bulk	Bytes/Bulk average sent in a backward direction

59	Bwd Avg Packets/Bulk	Packets/Bulk average sent in a backward direction
60	Bwd Avg Bulk Rate	Bulk rate average in a backward direction
61	Subflow Fwd Packets	No of packets in each sub-flow sent in a forward direction
62	Subflow Fwd Bytes	No of bytes in each sub-flow sent in a forward direction
63	Subflow Bwd Packets	No of packets in each sub-flow sent in a backward direction
64	Subflow Bwd Bytes	No of bytes in each sub-flow sent in a backward direction
65	Init_Win_bytes_fwd	Bytes sent at an initial window in a forward direction
66	Act_data_pkt_fwd	No of packets sent with min 1 byte in a forward direction
67	Min_seg_size_fwd	Min segment size sent in a forward direction
68	Active Mean	Active mean time of an active flow before becoming idle
69	Active Std	Standard deviation Active of an active flow before becoming idle
70	Active Max	Max time of an active flow before becoming idle
71	Active Min	Min time of an active flow before becoming idle

72	Idle Mean	Mean time of idle flow before becoming active
73	Idle Packet	Mean time of idle packet before becoming active
74	Idle Std	Standard deviation time of idle flow before becoming active
75	Idle Max	Max time of an idle flow before becoming active
76	Idle Min	Min time of an idle flow before becoming active
77	Label	Types of labels of data flow like benign, DoS, etc.

Appendix B. (Result of Polymorphic Attack on IDS using GAN)

B.1 Overall results of a manual polymorphic attack – 1

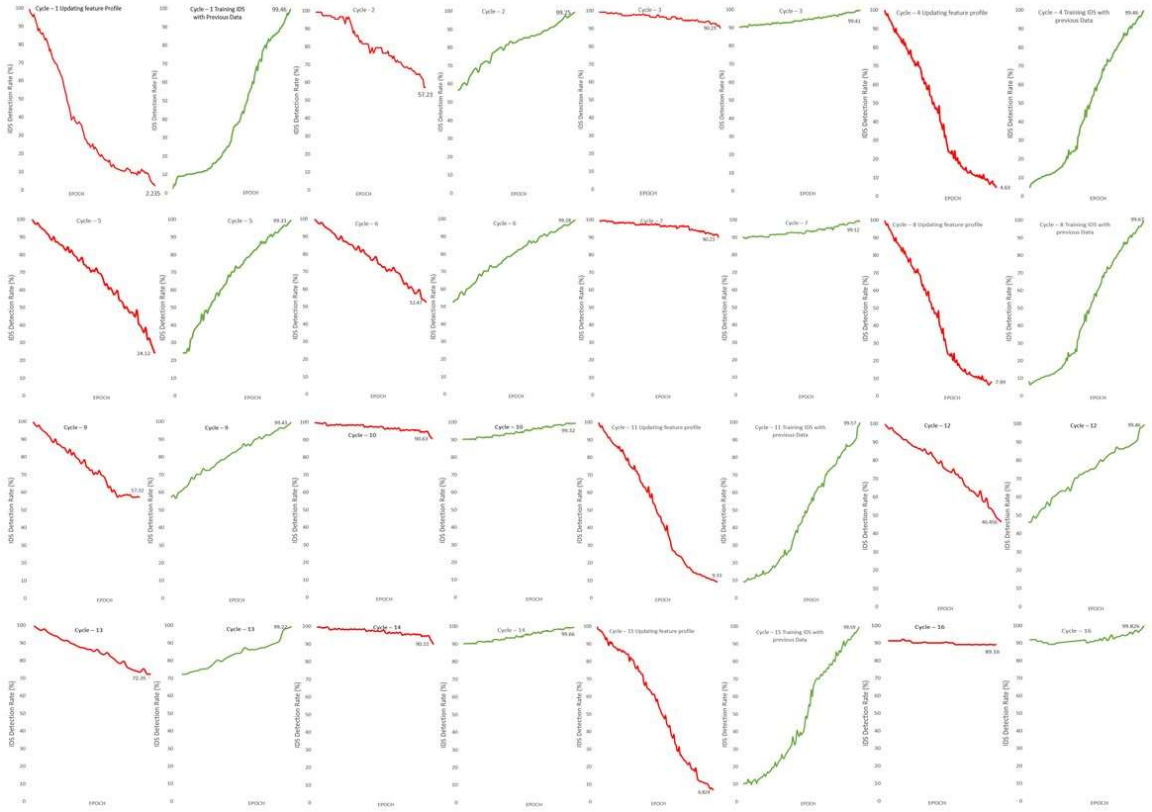

Figure 30: Overall result of the Polymorphic adversarial attack using algorithm 3

B.2 Overall results of a manual polymorphic attack – 2

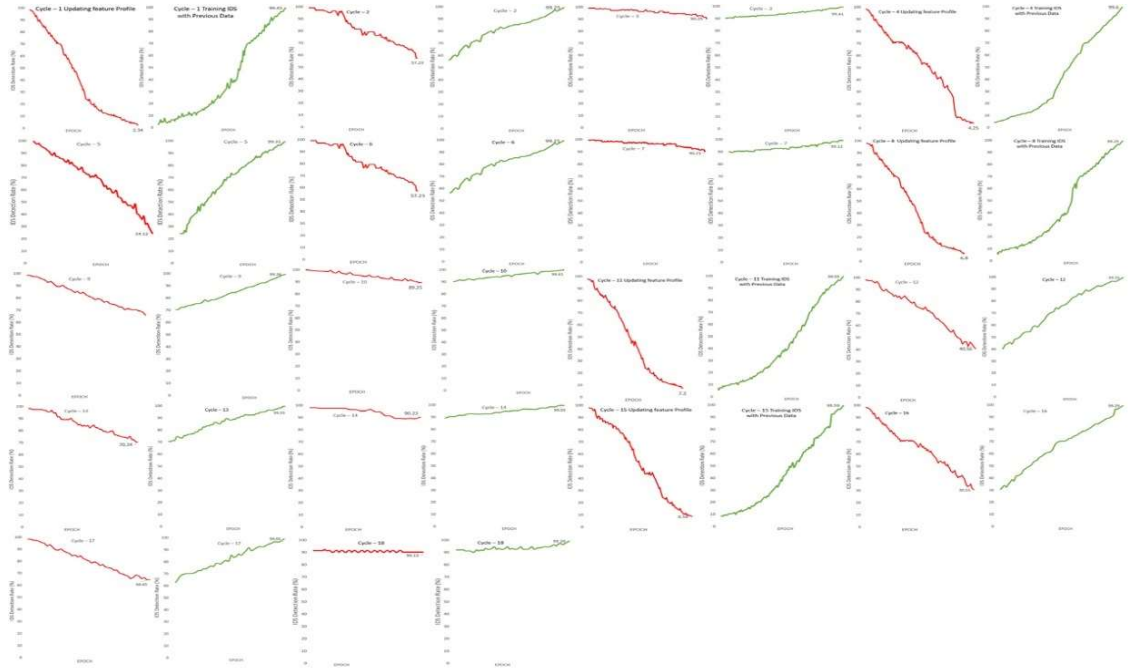

Figure 31: More results of Polymorphic adversarial attack using Algorithm 4

B.3 Automated Polymorphic DDoS Attack with 40 features

Table 9: Attack results with a total of 40 features

Cycles	IDS Detection Rate (%)
1 – **Attack**	**2.33**
1 – **Training IDS**	**99.45**
2 – Attack	45.24
2 – Training IDS	99.32
3 – Attack	77.42
3 – Training IDS	99.35
4 – Attack	90.13
4 – Training IDS	99.52
5 – **Attack**	**3.23**
5 – **Training IDS**	**99.73**
6 – Attack	69.12
6 – Training IDS	99.31
7 – Attack	91.32
7 – Training IDS	99.37
8 – **Attack**	**3.45**
8 – **Training IDS**	**99.09**
9 – Attack	57.43
9 – Training IDS	99.39

10 – Attack	89.14
10 – Training IDS	99.13
11 – **Attack**	**4.19**
11 – **Training IDS**	**99.14**
12 – Attack	46.23
12 – Training IDS	99.67
13 – Attack	90.51
13 – Training IDS	99.14
14 – **Attack**	**4.55**
14 – **Training IDS**	**99.24**
15 – Attack	45.23
15 – Training IDS	99.54
16 – Attack	74.19
16 – Training IDS	99.27
17 – Attack	88.17
17 – Training IDS	99.35
18 – **Attack**	**3.57**
18 – **Training IDS**	**99.07**
19 – Attack	63.21
19 – Training IDS	99.22
20 – Attack	90.62
20 – Training IDS	99.24

21 – **Attack**	**3.49**
21 – **Training IDS**	**99.58**
22 – Attack	61.18
22 – Training IDS	99.35
23 – Attack	86.63
23 – Training IDS	99.74
24 – **Attack**	**3.56**
24 – **Training IDS**	**99.03**
25 – Attack	42.14
25 – Training IDS	99.24
26 – Attack	71.87
26 – Training IDS	99.29
27 – Attack	91.71
27 – Training IDS	99.08
28 – **Attack**	**4.74**
28 – **Training IDS**	**99.21**
29 – Attack	34.32
29 – Training IDS	99.61
30 – Attack	66.40
30 – Training IDS	99.38
31 – Attack	87.96
31 – Training IDS	99.05

32 – **Attack**	**4.41**
32 – **Training IDS**	**99.15**
33 – Attack	26.88
33 – Training IDS	99.01
34 – Attack	57.19
34 – Training IDS	99.06
35 – Attack	85.14
35 – Training IDS	99.29
36 – **Attack**	**6.86**
36 – **Training IDS**	**99.46**
37 – Attack	37.36
37 – Training IDS	99.04
38 – Attack	62.26
38 – Training IDS	99.19
39 – Attack	84.37
39 – Training IDS	99.22
40 – **Attack**	**8.16**
40 – **Training IDS**	**99.36**
41 – Attack	53.19
41 – Training IDS	99.48
42 – Attack	89.45
42 – Training IDS	99.56

43 – **Attack**	**8.28**
43 – **Training IDS**	**99.30**
44 – Attack	66.40
44 – Training IDS	99.35
45 – Attack	88.16
45 – Training IDS	99.02
46 – **Attack**	**10.57**
46 – **Training IDS**	**99.31**
47 – Attack	59.23
47 – Training IDS	99.26
48 – Attack	87.79
48 – Training IDS	99.14
49 – **Attack**	**9.22**
49 – **Training IDS**	**99.09**
50 – Attack	45.20
50 – Training IDS	99.26
51 – Attack	65.18
51 – Training IDS	99.24
52 – Attack	87.79
52 – Training IDS	99.13
53 – **Attack**	**10.23**
53 – **Training IDS**	**99.77**

54 – Attack	69.32
54 – Training IDS	99.18
55 – Attack	88.21
55 – Training IDS	99.45
56 – **Attack**	**10.75**
56 – **Training IDS**	**99.16**
57 – Attack	55.73
57 – Training IDS	99.14
58 – Attack	84.70
58 – Training IDS	99.24
59 – **Attack**	**11.05**
59 – **Training IDS**	**99.57**
60 – Attack	88.37
60 – Training IDS	98.73
61 – **Attack**	**11.20**
61 – **Training IDS**	**98.86**
62 – Attack	41.18
62 – Training IDS	98.91
63 – Attack	91.11
63 – Training IDS	98.97

In all the results, the *red-colored* columns suggest that the generator is producing polymorphic adversarial DDoS attacks, and *green-colored* columns depict that the IDS detects the polymorphic attacks. As seen in the results, the RL algorithm can launch 19 different feature profiles to generate polymorphic DDoS attack data. This experiment indicates that the polymorphic attacks evade the IDS for 63 cycles.

B.4 Automated Polymorphic DDoS Attack with 50 features

Table 10: Attack results with a total of 50 features

Cycles	IDS Detection Rate (%)
1 – **Attack**	**2.33**
1 – **Training IDS**	**99.12**
2 – Attack	55.24
2 – Training IDS	99.32
3 – Attack	91.33
3 – Training IDS	99.52
4 – **Attack**	**3.53**
4 – **Training IDS**	**99.23**
5 – Attack	59.12
5 – Training IDS	99.31
6 – Attack	90.57
6 – Training IDS	99.37
7 – **Attack**	**3.59**

7 – **Training IDS**	**99.09**
8 – Attack	42.73
8 – Training IDS	99.39
9 – Attack	89.14
9 – Training IDS	99.13
10 – **Attack**	**4.19**
10 – **Training IDS**	**99.14**
11 – Attack	46.23
11 – Training IDS	99.67
12 – Attack	90.51
12 – Training IDS	99.14
13 – **Attack**	**4.55**
13 – **Training IDS**	**99.24**
14 – Attack	45.23
14 – Training IDS	99.54
15 – Attack	74.19
15 – Training IDS	99.27
16 – Attack	88.17
16 – Training IDS	99.35
17 – **Attack**	**3.97**
17 – **Training IDS**	**99.07**
18 – Attack	66.21

18 – Training IDS	99.22
19 – Attack	90.62
19 – Training IDS	99.24
20 – **Attack**	**4.66**
20 – **Training IDS**	**99.58**
21 – Attack	63.18
21 – Training IDS	99.35
22 – Attack	88.37
22 – Training IDS	99.74
23 – **Attack**	**4.56**
23 – **Training IDS**	**99.03**
24 – Attack	57.14
24 – Training IDS	99.24
25 – Attack	75.87
25 – Training IDS	99.29
26 – Attack	90.71
26 – Training IDS	99.08
27 – **Attack**	**4.74**
27 – **Training IDS**	**99.21**
28 – Attack	34.32
28 – Training IDS	99.61
29 – Attack	66.40

29 – Training IDS	99.38
30 – Attack	87.96
30 – Training IDS	99.05
31 – **Attack**	**3.41**
31 – **Training IDS**	**99.15**
32 – Attack	57.19
32 – Training IDS	99.06
33 – Attack	85.14
33 – Training IDS	99.29
34 – **Attack**	**6.86**
34 – **Training IDS**	**99.46**
35 – Attack	37.36
35 – Training IDS	99.04
36 – Attack	62.26
36 – Training IDS	99.19
37 – Attack	84.37
37 – Training IDS	99.22
38 – **Attack**	**8.16**
38 – **Training IDS**	**99.36**
39 – Attack	53.19
39 – Training IDS	99.48
40 – Attack	89.45

40 – Training IDS	99.56
41 – **Attack**	**8.28**
41 – **Training IDS**	**99.30**
42 – Attack	66.40
42 – Training IDS	99.35
43 – Attack	88.16
43 – Training IDS	99.02
44 – **Attack**	**9.57**
44 – **Training IDS**	**99.31**
45 – Attack	59.23
45 – Training IDS	99.26
46 – Attack	87.79
46 – Training IDS	99.14
47 – **Attack**	**9.22**
47 – **Training IDS**	**99.09**
48 – Attack	45.20
48 – Training IDS	99.26
49 – Attack	87.79
49 – Training IDS	99.13
50 – **Attack**	**10.23**
50 – **Training IDS**	**99.77**
51 – Attack	69.32

51 – Training IDS	99.18
52 – Attack	88.21
52 – Training IDS	99.45
53 – **Attack**	**10.75**
53 – **Training IDS**	**99.16**
54 – Attack	55.73
54 – Training IDS	99.14
55 – Attack	84.70
55 – Training IDS	99.24
56 – **Attack**	**11.05**
56 – **Training IDS**	**99.57**
57 – Attack	52.53
57 – Training IDS	99.73
58 – Attack	88.37
58 – Training IDS	98.73
59 – **Attack**	**11.20**
59 – **Training IDS**	**98.86**
60 – Attack	41.18
60 – Training IDS	98.91
61 – Attack	91.11
61 – Training IDS	98.97
62 – **Attack**	**12.20**

62 – **Training IDS**	**99.26**
63 – Attack	61.25
63 – Training IDS	98.97
64 – Attack	90.32
64 – Training IDS	99.17
65 – **Attack**	**13.65**
65 – **Training IDS**	**98.97**
66 – Attack	60.87
66 – Training IDS	99.47
67 – Attack	92.41
67 – Training IDS	98.97
68 – **Attack**	**15.53**
68 – **Training IDS**	**98.97**
69 – Attack	52.55
69 – Training IDS	98.97
70 – Attack	90.42
70 – Training IDS	98.97
71 – **Attack**	**16.41**
71 – **Training IDS**	**98.97**
72 – Attack	61.88
72 – Training IDS	98.97
73 – Attack	89.36

73 – Training IDS	98.97

As seen in the results, the RL algorithm can launch 23 different feature profiles to generate polymorphic DDoS attack data. This experiment indicates that the polymorphic attacks evade the IDS for 73 cycles.

B.5 Automated Polymorphic DDoS Attack with 60 features

Table 11: Attack results with a total of 60 features

Cycles	IDS Detection Rate (%)
1 – **Attack**	**2.55**
1 – **Training IDS**	**99.12**
2 – Attack	52.64
2 – Training IDS	99.32
3 – Attack	91.33
3 – Training IDS	99.52
4 – **Attack**	**3.53**
4 – **Training IDS**	**99.23**
5 – Attack	59.12
5 – Training IDS	99.31
6 – Attack	92.12
6 – Training IDS	99.37
7 – **Attack**	**3.59**

7 – **Training IDS**	**99.09**
8 – Attack	42.73
8 – Training IDS	99.39
9 – Attack	89.14
9 – Training IDS	99.13
10 – **Attack**	**4.19**
10 – **Training IDS**	**99.14**
11 – Attack	46.23
11 – Training IDS	99.67
12 – Attack	90.51
12 – Training IDS	99.14
13 – **Attack**	**4.86**
13 – **Training IDS**	**99.24**
14 – Attack	45.23
14 – Training IDS	99.54
15 – Attack	74.19
15 – Training IDS	99.27
16 – Attack	88.17
16 – Training IDS	99.35
17 – **Attack**	**4.23**
17 – **Training IDS**	**99.07**
18 – Attack	66.21

18 – Training IDS	99.22
20 – Attack	90.62
20 – Training IDS	99.24
21 – **Attack**	**4.66**
21 – **Training IDS**	**99.58**
22 – Attack	64.18
22 – Training IDS	99.35
23 – Attack	87.37
23 – Training IDS	99.74
24 – **Attack**	**4.56**
24 – **Training IDS**	**99.03**
25 – Attack	47.14
25 – Training IDS	99.24
26 – Attack	71.87
26 – Training IDS	99.29
27 – Attack	91.71
27 – Training IDS	99.08
28 – **Attack**	**3.88**
28 – **Training IDS**	**99.21**
29 – Attack	34.32
29 – Training IDS	99.61
30 – Attack	66.40

30 – Training IDS	99.38
31 – Attack	87.96
31 – Training IDS	99.05
32 – **Attack**	**4.53**
32 – **Training IDS**	**99.15**
33 – Attack	46.54
33 – Training IDS	99.01
34 – Attack	88.47
34 – Training IDS	99.29
35 – **Attack**	**6.86**
35 – **Training IDS**	**99.46**
36 – Attack	37.36
36 – Training IDS	99.04
37 – Attack	62.26
37 – Training IDS	99.19
38 – Attack	84.37
38 – Training IDS	99.22
39 – **Attack**	**8.16**
39 – **Training IDS**	**99.36**
40 – Attack	53.19
40 – Training IDS	99.48
41 – Attack	89.45

41 – Training IDS	99.56
42 – **Attack**	**8.28**
42 – **Training IDS**	**99.30**
43 – Attack	66.40
43 – Training IDS	99.35
44 – Attack	88.16
44 – Training IDS	99.02
45 – **Attack**	**10.57**
45 – **Training IDS**	**99.31**
46 – Attack	59.23
46 – Training IDS	99.26
47 – Attack	87.79
47 – Training IDS	99.14
48 – **Attack**	**9.43**
48 – **Training IDS**	**99.09**
49 – Attack	45.20
49 – Training IDS	99.26
50 – Attack	65.18
50 – Training IDS	99.24
51 – Attack	87.79
51 – Training IDS	99.13
52 – **Attack**	**10.23**

52 – **Training IDS**	**99.77**
53 – Attack	69.32
53 – Training IDS	99.18
54 – Attack	88.21
54 – Training IDS	99.45
55 – **Attack**	**10.75**
55 – **Training IDS**	**99.16**
56 – Attack	55.73
56 – Training IDS	99.14
57 – Attack	84.70
57 – Training IDS	99.24
58 – **Attack**	**11.05**
58 – **Training IDS**	**99.57**
59 – Attack	65.81
59 – Training IDS	99.05
60 – Attack	88.37
60 – Training IDS	98.73
61 – **Attack**	**11.20**
61 – **Training IDS**	**98.86**
62 – Attack	41.18
62 – Training IDS	98.91
63 – Attack	91.11

63 – Training IDS	99.08
64 – **Attack**	**12.64**
64 – **Training IDS**	**99.18**
65 – Attack	53.69
65 – Training IDS	99.12
66 – Attack	91.59
66 – Training IDS	98.96
67 – **Attack**	**14.52**
67 – **Training IDS**	**99.12**
68 – Attack	55.93
68 – Training IDS	99.38
69 – Attack	88.65
69 – Training IDS	99.29
70 – **Attack**	**16.82**
70 – **Training IDS**	**98.97**
71 – Attack	59.49
71 – Training IDS	99.17
72 – Attack	88.27
72 – Training IDS	99.09
73 – **Attack**	**18.48**
73 – **Training IDS**	**99.45**
74 – Attack	48.17

74 – Training IDS	99.38
75 – Attack	87.31
75 – Training IDS	98.94
76 **– Attack**	**19.87**
76 **– Training IDS**	**99.38**
77 – Attack	68.77
77 – Training IDS	98.97
78 – Attack	92.02
78 – Training IDS	98.97
79 **– Attack**	**19.28**
79 **– Training IDS**	**98.97**
80 – Attack	64.21
80 – Training IDS	99.02
81 – Attack	91.11
81 – Training IDS	99.24

As seen in the results, the RL algorithm can launch 25 different feature profiles to generate polymorphic DDoS attack data. This experiment indicates that the polymorphic attacks evade the IDS for 81 cycles.

B.6 Automated Polymorphic DDoS Attack with 76 features

Table 12: Attack results with a total of 76 features

No of Cycles	IDS Detection Rate (%)
1 – **Attack**	**2.57**
1 – **Training IDS**	**99.54**
2 – Attack	43.65
2 – Training IDS	99.23
3 – Attack	72.72
3 – Training IDS	99.35
4 – Attack	89.43
4 – Training IDS	99.52
5 – **Attack**	**3.77**
5 – **Training IDS**	**99.13**
6 – Attack	65.21
6 – Training IDS	99.11
7 – Attack	88.66
7 – Training IDS	99.37
8 – **Attack**	**3.65**
8 – **Training IDS**	**99.49**
9 – Attack	62.63
9 – Training IDS	99.22

10 – Attack	91.34
10 – Training IDS	99.13
11 – **Attack**	**5.02**
11 – **Training IDS**	**99.41**
12 – Attack	46.23
12 – Training IDS	99.67
13 – Attack	90.51
13 – Training IDS	99.14
14 – **Attack**	**4.55**
14 – **Training IDS**	**99.24**
15 – Attack	45.23
15 – Training IDS	99.54
16 – Attack	74.19
16 – Training IDS	99.27
17 – Attack	88.17
17 – Training IDS	99.35
18 – **Attack**	**3.57**
18 – **Training IDS**	**99.07**
19 – Attack	63.21
19 – Training IDS	99.22
20 – Attack	90.62
20 – Training IDS	99.24

21 – **Attack**	**3.49**
21 – **Training IDS**	**99.58**
22 – Attack	61.18
22 – Training IDS	99.35
23 – Attack	86.63
23 – Training IDS	99.74
24 – **Attack**	**3.56**
24 – **Training IDS**	**99.03**
25 – Attack	42.14
25 – Training IDS	99.24
26 – Attack	71.87
26 – Training IDS	99.29
27 – Attack	91.71
28 – Training IDS	99.08
29 – **Attack**	**3.74**
29 – **Training IDS**	**99.21**
30 – Attack	34.32
30 – Training IDS	99.61
31 – Attack	66.40
31 – Training IDS	99.38
32 – Attack	87.96
32 – Training IDS	99.05

33 – **Attack**	**3.55**
33 – **Training IDS**	**99.15**
34 – Attack	26.88
34 – Training IDS	99.01
35 – Attack	57.19
35 – Training IDS	99.06
36 – Attack	85.14
36 – Training IDS	99.29
37 – **Attack**	**5.86**
37 – **Training IDS**	**99.46**
38 – Attack	37.36
38 – Training IDS	99.04
39 – Attack	62.26
39 – Training IDS	99.19
40 – Attack	84.37
40 – Training IDS	99.22
41 – **Attack**	**6.16**
41 – **Training IDS**	**99.36**
42 – Attack	53.19
42 – Training IDS	99.48
43 – Attack	89.45
43 – Training IDS	99.56

44 – **Attack**	**7.98**
44 – **Training IDS**	**99.30**
45 – Attack	66.40
45 – Training IDS	99.35
46 – Attack	88.16
46 – Training IDS	99.02
47 – **Attack**	**10.11**
47 – **Training IDS**	**99.31**
48 – Attack	59.23
48 – Training IDS	99.26
49 – Attack	87.79
49 – Training IDS	99.14
50 – **Attack**	**10.22**
50 – **Training IDS**	**99.09**
51 – Attack	45.20
51 – Training IDS	99.26
52 – Attack	65.18
52 – Training IDS	99.24
53 – Attack	87.79
53 – Training IDS	99.13
54 – **Attack**	**10.55**
54 – **Training IDS**	**99.77**

55 – Attack	69.32
55 – Training IDS	99.18
56 – Attack	88.21
56 – Training IDS	99.45
57 – **Attack**	**10.75**
57 – **Training IDS**	**99.16**
58 – Attack	55.73
58 – Training IDS	99.14
59 – Attack	84.70
59 – Training IDS	99.24
60 – **Attack**	**11.05**
60 – **Training IDS**	**99.57**
61 – Attack	88.37
61 – Training IDS	98.73
62 – **Attack**	**11.20**
62 – **Training IDS**	**98.86**
63 – Attack	41.18
63 – Training IDS	98.91
64 – Attack	91.11
64 – Training IDS	98.97
65 – **Attack**	**11.12**
65 – **Training IDS**	**98.67**

66 – Attack	54.78
66 – Training IDS	99.39
67 – Attack	82.31
67 – Training IDS	99.24
68 – **Attack**	**11.50**
68 – **Training IDS**	**99.13**
69 – Attack	45.24
69 – Training IDS	99.18
70 – Attack	83.82
70 – Training IDS	99.45
71 – **Attack**	**12.12**
71 – **Training IDS**	**99.51**
72 – Attack	41.05
72 – Training IDS	99.16
73 – Attack	70.35
73 – Training IDS	99.14
74 – Attack	90.55
74 – Training IDS	99.57
75 – **Attack**	**11.35**
75 – **Training IDS**	**99.77**
76 – Attack	77.66
76 – Training IDS	99.10

77 – Attack	89.85
77 – Training IDS	99.24
78 – **Attack**	**12.13**
78 – **Training IDS**	**99.18**
79 – Attack	43.61
79 – Training IDS	99.14
80 – Attack	90.62
80 – Training IDS	99.53
81 – **Attack**	**13.34**
81 – **Training IDS**	**99.62**
82 – Attack	74.59
82 – Training IDS	99.26
83 – Attack	90.19
83 – Training IDS	99.41
84 – **Attack**	**15.22**
84 – **Training IDS**	**99.26**
85 – Attack	89.38
85 – Training IDS	99.21
86 – **Attack**	**16.22**
86 – **Training IDS**	**99.23**
87 – Attack	88.59
87 – Training IDS	99.18

88 – **Attack**	**18.59**
88 – **Training IDS**	**99.08**
89 – Attack	90.43
89 – Training IDS	99.06
90 – **Attack**	**19.43**
90 – **Training IDS**	**99.02**
91 – Attack	63.67
91 – Training IDS	99.15
92 – Attack	90.43
92 – Training IDS	99.45
93 – **Attack**	**21.35**
93 – **Training IDS**	**99.47**
94 – Attack	84.55
94 – Training IDS	99.26
95 – **Attack**	**23.48**
95 – **Training IDS**	**99.56**
96 – Attack	87.12
96 – Training IDS	99.51
97 – **Attack**	**27.64**
97 – **Training IDS**	**99.61**
98 – Attack	90.12
98 – Training IDS	99.26

99 – Attack	92.12
99 – Training IDS	99.36

As seen in the results, the RL algorithm can launch 32 different feature profiles to generate polymorphic DDoS attack data. This experiment indicates that the polymorphic attacks evade the IDS for 99 cycles.

Appendix C. (Source Code)

C.1 Feature Selection using SHAP

```
import catboost
from catboost import CatBoostRegressor
import shap
import numpy as np
import pandas as pd

dataset = pd.read_csv('DDoS.csv', low_memory = False)
array = dataset.values
X = array[:, :-1]
Y = array[:, -1]

model = CatBoostRegressor(iterations=500, learning_rate=0.01, random_seed=123)
model.fit(X, Y, verbose=False, plot=False)

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X)

shap.summary_plot(shap_values, X, dataset.columns, plot_type="bar")
shap.summary_plot(shap_values, X, dataset.columns)
```

C.2 The Generator, The Discriminator, Black-Box IDS

```
import torch as th
from torch import nn
from torch.autograd import Variable as V
import numpy as np
import torch
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Model for the IDS
class BlackboxIDS(nn.Module):
    def __init__(self,input_dim, output_dim):
        super().__init__()
        self.layer = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.Dropout(0.6),
            nn.LeakyReLU(True),
            nn.Linear(64, 32),
            nn.Dropout(0.5),
            nn.LeakyReLU(True),
            nn.Linear(32, output_dim),
            nn.Dropout(0.5),
            nn.LeakyReLU(True),
        ).to(device)
        self.output = nn.Sigmoid().to(device)
    def forward(self,x):
        x = self.layer(x)
        return x

# Model for the Discriminator
class Discriminator(nn.Module):
    def __init__(self,input_dim, output_dim):
        super(Discriminator, self).__init__()
        self.layer = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.LeakyReLU(True),
            nn.Linear(64 , 32),
            nn.LeakyReLU(True),
            nn.Linear(32, output_dim)
        ).to(device)
    def forward(self,x):
        return self.layer(x)

# GAN - Produce Adversarial Attack
class Generator(nn.Module):
```

```
      def __init__(self,input_dim, output_dim):
            super(Generator, self).__init__()
            self.layer = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 32),
            nn.ReLU(True),
            nn.Linear(32, output_dim),
            nn.ReLU(True),
            nn.Tanh()
         ).to(device)
    def forward(self, noise_dim, raw_attack, attack_category, POS_NONFUNCTIONAL_F
EATURES):
        '''
        Generate Aversarial Attack Traffic while keeping functional features stat
ic
        '''
        if attack_category != 'DDoS':
            raise ValueError("Preprocess Data Fail: Invalid Attack Category")
        batch_size = len(raw_attack)
        pos_nonfunctional_feature = POS_NONFUNCTIONAL_FEATURES[attack_category]
        noise = V(th.Tensor(np.random.uniform(0,1,(batch_size, noise_dim)))).to(d
evice)
        generator_out = self.layer(noise)
        # Keep the functional features
        adversarial_attack = raw_attack.clone().type(torch.FloatTensor).to(device
)
        for idx in range(batch_size):
            adversarial_attack[idx][pos_nonfunctional_feature] = generator_out[id
x]
        return th.clamp(adversarial_attack,0.,1.).to(device)
```

C.3 WGAN

```
import numpy as np
import pandas as pd
import torch as th
from torch.autograd import Variable as V
import torch.autograd as autograd
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
import pickle

from keras.models import load_model

from models import *
from constants import *

import matplotlib.pyplot as plt
import math
import os
from datetime import date
import timeit

Dataset_Path = base_path + "Dataset/"
SavedModelPath = base_path + "Saved Model/"
Trainsets_Path = Dataset_Path + 'Trainset/'
g_trainset_path = Trainsets_Path + "WGAN_G.csv"
d_trainset_path = Trainsets_Path + "WGAN_D.csv"
testset_path = Dataset_Path + "Testset/" + "DDoS.csv"

GAN_Model_Path = SavedModelPath + 'WGANModel/'

IDS_Saved_Path = SavedModelPath + 'B_B_IDSModel/'


# Global Variables
N_FEATURES = 76

IDS_INPUT_DIM = N_FEATURES
IDS_OUTPUT_DIM = 2
attack_type = ['DDoS']

fun_feature = {'DDoS': DDoS_FEATURES}
non_fun_feature = {}
for atck_cat, pos_functional_feature in fun_feature.items():
```

```
non_fun_feature = {}
for atck_cat, pos_functional_feature in fun_feature.items():
    nonfuntional_feature = []
    for i in range(N_FEATURES):
        if i not in pos_functional_feature:
            nonfuntional_feature.append(i)
    non_fun_feature[atck_cat] = nonfuntional_feature

IDS_MODELS = {'RF'}


def create_batch(x,batch_size):
    a = list(range(len(x)))
    np.random.shuffle(a)
    x = x[a]
    batch_x = [x[batch_size * i : (i+1)*batch_size,:] for i in range(len(x)//batc
h_size)]
    return np.array(batch_x)

def prepro_attack_data(dataset, atck_cat):
    if atck_cat != 'DDoS':
      raise ValueError("Data Preprocessing failed: category not found")
    attack_data = dataset[dataset['class'] == atck_cat]
    del attack_data["class"]
    return np.array(attack_data)

def get_ids_path(model_name, atck_cat, created_date):
    if atck_cat != 'DOS':
        raise ValueError("Data Preprocessing failed: category not found")
    ids_path = str(f"{IDS_Saved_Path}{atck_cat}/ML/from_{cre-
ated_date}_{model_name}.pkl")
    if not os.path.exists(ids_path):
        raise ValueError(f"Invalid path: {ids_path}\nfile does not exist!")
    return ids_path


#IDS Models
def load_ids(model_name, atck_cat, created_date):
    ids_model_path = get_ids_path(model_name, atck_cat, created_date)
    with open(ids_model_path, 'rb') as file:
        pickle_model = pickle.load(file)
        print(f"{4*' '}IDS Loaded from: {ids_model_path}")
    return pickle_model
```

```
# Initialize the Generator
def init_generator(input_dim, output_dim):
    generator = Generator(input_dim, output_dim)
    return generator

def create_adversarial_DDoS_attack(generator, noise_dim, raw_atk, atck_cat):
    batch_size = len(raw_atk)
    noise = V(th.Tensor(np.random.uniform(0,1,(batch_size, noise_dim))))
    gen_out = generator(noise)
    adv_DDoS_attack = create_adversarial_DDoS_at-
tack(gen_out, raw_atk, atck_cat, non_fun_feature)
    return adv_DDoS_attack

def train_generator(generator, discriminator, opt_gen, noise_dim, attack_traf-
fic, atck_cat):
    for p in discriminator.parameters():
        p.requires_grad = False
    opt_gen.zero_grad()
    # Generator Generate Adversarial Attack
    adv_DDoS_attack = create_adversarial_DDoS_attack(generator, noise_dim, at-
tack_traffic, atck_cat)
    # GAN-D predict, Generator update parameter
    D_pred = discriminator(adv_DDoS_attack)
    g_loss = -th.mean(D_pred)
    g_loss.backward()
    opt_gen.step()
    return g_loss

def train_discriminator(discriminator, ids_model, generator, critic_iters,
opt_disc, normal_b, noise_dim, attack_traffic, atck_cat):
    run_d_loss = 0
    cnt = 0
    for p in discriminator.parameters():
      p.requires_grad = True

    for c in range(critic_iters):
        opt_disc.zero_grad()
```

```
adv_DDoS_attack = create_adversarial_DDoS_attack(generator, noise_dim, attack_tra
ffic, atck_cat)

        ids_input = th.cat((adv_DDoS_attack,normal_b))
        l = list(range(len(ids_input)))
        np.random.shuffle(l)
        ids_input = V(th.Tensor(ids_input[l]))

        ids_pred_label = V(th.Tensor(ids_model.predict(ids_input)))
        pred_normal = ids_input[ids_pred_label==0]
        pred_attack = ids_input[ids_pred_label==1]
        if len(pred_attack) == 0:
            cnt += 1
            break

        Disc_Normal = discriminator(V(th.Tensor(pred_normal)))
        Disc_Attack = discriminator(V(th.Tensor(pred_attack)))

        loss_normal = th.mean(Disc_Normal)
        loss_attack = th.mean(Disc_Attack)
        gradient_penalty = calculate_penalty(discriminator, normal_b.data, adv_DD
oS_attack.data)
        d_loss = loss_attack - loss_normal
        d_loss.backward()
        opt_disc.step()
        run_d_loss += d_loss.item()
    return run_d_loss, cnt

# Calculate Penalty
def calculate_penalty(D, normal_t, attack_t):
    alpha = th.Tensor(np.random.random((normal_t.shape[0], 1)))
    between_n_a = (alpha * normal_t + ((1 -
 alpha) * attack_t)).requires_grad_(True)
    d_between_n_a = D(between_n_a)
    adv = V(th.Tensor(normal_t.shape[0], 1).fill_(1.0), requires_grad=False)
    gradients = autograd.grad(
        outputs=d_between_n_a,
        inputs=between_n_a,
        grad_outputs=adv,
        create_graph=True,
        retain_graph=True,
        only_inputs=True,
    )[0]
```

```
gradients = gradients.view(gradients.size(0), -1)
    gradient_penalty = ((gradients.norm(2, dim=1) - 1) ** 2).mean()
    return gradient_penalty

def cal_dr(ids_model, normal, raw_atk, adv_DDoS_attack):
    # Make data to feed IDS contain: Attack & Normal
    o_ids_input = th.cat((raw_atk, normal))
    a_ids_input = th.cat((adv_DDoS_attack,normal))
    # Shuffle Input
    l = list(range(len(a_ids_input)))
    np.random.shuffle(l)
    o_ids_input = o_ids_input[l]
    a_ids_input = a_ids_input[l]
    # IDS Predict Label
    o_pred_label = th.Tensor(ids_model.predict(o_ids_input))
    a_pred_label = th.Tensor(ids_model.predict(a_ids_input))
    # True Label
    ids_true_label = np.r_[np.ones(BATCH_SIZE),np.zeros(BATCH_SIZE)][l]
    # Calc DR
    tn1, fn1, fp1, tp1 = confusion_matrix(ids_true_label,o_pred_label).ravel()
    tn2, fn2, fp2, tp2 = confusion_matrix(ids_true_label,a_pred_label).ravel()
    origin_dr = tp1/(tp1 + fp1)
    adversarial_dr = tp2/(tp2 + fp2)
    return origin_dr, adversarial_dr

ids_ml_model_name = "RF"
ids_created_date = 'Auto'
if ids_created_date == 'Auto':
    ids_created_date = IDS_Model_Created_Auto[ids_ml_model_name]
    print(f"IDS: {ids_ml_model_name} - created on: \t{ids_created_date}")
GAN_variant = 'WGAN'

# Hyper-parameters
BATCH_SIZE = 256
learning_rate = 0.0001
LAMBDA = 10
CRITIC_ITERS = 5


# GAN-D
D_INPUT_DIM = N_FEATURES
D_OUTPUT_DIM = 1
discriminator = Discriminator(D_INPUT_DIM,D_OUTPUT_DIM)
opt_disc = optim.Adam(discriminator.parameters(), lr=learning_rate)
```

```
g_train_data = pd.read_csv(g_trainset_path)
d_train_data = pd.read_csv(d_trainset_path)

del d_train_data["class"]
normal = np.array(d_train_data)

print("Amout of Generator Trainset:", g_train_data.shape[0])
print("Amout of Discriminator Trainset:", d_train_data.shape[0])

for atck_cat in attack_type:
    total_time_start = timeit.default_timer()

    # Load IDS
    ids_model = load_ids(ids_ml_model_name, atck_cat, ids_cr    eated_date)

    # Initialize the Generator
    G_OUTPUT_DIM = len(non_fun_feature[atck_cat]) # Generator o utput is number o
f nonfunctional feature
    print(f"nf : {G_OUTPUT_DIM} (num. of nonfunctional features)")
    G_INPUT_DIM = NOISE_DIM
    print(f"Generator noise_vector_dim : {NOISE_DIM}")
    print(f"Generator input_feat_dim : {G_INPUT_DIM}")
    print(f"Generator out_dim: {G_OUTPUT_DIM}")
    generator = init_generator(G_INPUT_DIM,G_OUTPUT_DIM,)
    opt_gen = optim.Adam(generator.parameters(), lr=learning_rate)
    # Load Raw Attack Dataset
    raw_atk = prepro_attack_data(g_train_data, atck_cat)
    # Prepare Save Folder
    ids_path = str(f"{GAN_Model_Path}ML/{ids_ml_model_name}/")
    if not os.path.exists(ids_path):
        os.makedirs(ids_path)
          GAN_Save_Path = str(f"{ids_path}{atck_cat}")
    if not os.path.exists(GAN_Save_Path):
        os.makedirs(GAN_Save_Path)


# Create batch of attack traffic
batch_attack = create_batch(raw_atk,BATCH_SIZE)
# Declare Loss, DR List and Train Generator, GAN-D
d_losses,g_losses = [],[]
o_dr, a_dr = [],[]
generator.train()
discriminator.train()
```

```
# IDS Training Started
print(f"-->IDSGAN start training")
for epoch in range(MAX_EPOCH):
    batch_normal = create_batch(normal,BATCH_SIZE)
      epoch_time_start = timeit.default_timer()
      cnt = 0
      run_g_loss = 0.
      run_d_loss = 0.
      epoch_o_drs, epoch_a_drs = [], []

      for idx, bn in enumerate(batch_normal):
            normal_b = th.Tensor(bn.astype("float64"))
            attack_traffic  = V(th.Tensor(batch_attack[idx % len(batch_attack)]))
            #  Train Generator
            g_loss = train_generator(generator, discriminator, opt_gen, NOISE_DIM
, attack_traffic, atck_cat)
            run_g_loss += g_loss.item()

            # Train Discriminator
            d_loss, current_cnt = train_discriminator(discriminator, ids_model, g
enerator, CRITIC_ITERS, opt_disc, normal_b, NOISE_DIM, attack_traffic, atck_cat)
            run_d_loss += d_loss
            cnt += current_cnt

            # CALC Epoch DR
            adv_DDoS_attack = create_adversarial_DDoS_attack(generator, NOISE_DIM
, attack_traffic, atck_cat).detach()
            origin_dr, adversarial_dr = cal_dr(ids_model, normal_b, attack_traffi
c, adv_DDoS_attack)
            epoch_o_drs.append(origin_dr)
            epoch_a_drs.append(adversarial_dr)


if cnt >= (len(normal)/BATCH_SIZE):
            print("predicted attack does not exist")
            break
            d_losses.append(run_d_loss/CRITIC_ITERS)
        g_losses.append(run_g_loss)
        epoch_o_dr = np.mean(epoch_o_drs)
        epoch_a_dr = np.mean(epoch_a_drs)
        o_dr.append(epoch_o_dr)
        a_dr.append(epoch_a_dr)
        runtime = timeit.default_timer() - epoch_time_start
```

```
for val in print_vals:
            if isinstance(val, float):
                print_string.append(str(f"{val:.2f}"))
            else:
                print_string.append(str(val))

        if (epoch == 0 or (epoch + 1) % 10 == 0):
            model_g_save_name = f"time_created_{today}_GAN_G_{epoch+1}epoch.pth"
            path = GAN_Save_Path + model_g_save_name
            th.save(generator.state_dict(), path)

overall_running_time = timeit.default_timer() - total_time_start
# Save Model
model_d_save_name = f"time_created_{today}_GAN_D_{MAX_EPOCH}epoch.pth"
path = GAN_Save_Path + model_d_save_name
th.save(discriminator.state_dict(), path)
print(f"Total model runtime: {overall_running_time:.2f}")

plt.plot(d_losses,label = "Discriminator_Loss")
plt.plot(g_losses, label = "Generator_Loss")
plt.legend()
plt.show()
# DR-Graph
plt.plot(o_dr,label = "Origin DR")
plt.plot(a_dr, label = "Adversarial DR")
plt.legend()
plt.show()


# Evaluate adversarial DDoS data
for atck_cat in attack_type:
    # Load sklearn IDS Model
    ids_model = load_ids(ids_ml_model_name, atck_cat, ids_cr    eated_date)
    # Init Generator model
    G_OUTPUT_DIM = len(non_fun_feature[atck_cat]) # Generator I nput dimension is
 dimention of noise
    print(f"nf: {G_OUTPUT_DIM} (num. of nonfunctional features)")
    G_INPUT_DIM = NOISE_DIM
    print(f"Generator noise_vector_dim : {NOISE_DIM}")
    print(f"Generator input_feat_dim : {G_INPUT_DIM}")
    print(f"Generator out_dim: {G_OUTPUT_DIM}")
    generator = init_generator(G_INPUT_DIM,G_OUTPUT_DIM)
```

```
# Load Attack Dataset
    test_raw_atk = prepro_attack_data(testset, atck_cat)
    # Create batch of attack traffic
    batch_attack = create_batch(test_raw_atk, BATCH_SIZE)
    n_batch_attack = len(batch_attack)
    print(f"{4*' '}Amout of {atck_cat}:\t{len(test_raw_atk)} ({n_batch_attack} ba
tchs - {BATCH_SIZE} records/batch)")

    # Calculate Detection Rate for each epoch
    gan_g_folder_path = str(f"{GAN_Model_Path}ML/{ids_ml_model_name}/{atck_cat}")
    print(f"{4*' '}GAN Models Folder: {gan_g_folder_path}")
    for epoch in range(0, MAX_EPOCH + 1, 10):
        # Load Generator Model
        model_g_save_name = f"time_created_{gan_model_time_created}_GAN_G_{1 if e
poch == 0 else epoch}epoch.pth"
        gan_g_model_path = gan_g_folder_path + model_g_save_name
        param = th.load(gan_g_model_path,map_location=lambda x,y:x)
        generator.load_state_dict(param)
        generator.eval()

        o_dr,a_dr =[],[]
        with th.no_grad():
            for idx, bn in enumerate(test_batch_normal):
                normal_b = th.Tensor(bn)
                attack_b = th.Tensor(batch_attack[idx % n_batch_attack])
                # Generate Adversarial Traffic
                adv_DDoS_attack_b = create_adversarial_DDoS_attack(generator, NOI
SE_DIM, attack_b, atck_cat).detach()


      # Calculate Detection Rate

      origin_dr, adversarial_dr = cal_dr(ids_model, normal_b, attack_b, adversary
al_attack_b)
      o_dr.append(origin_dr)
      a_dr.append(adversarial_dr)
      eir = 1 - (np.mean(a_dr)/np.mean(o_dr))
      print(f"\t {epoch:3d} epochs:\tOrigin DR : {np.mean(o_dr)*100:.2f}% \t Adve
rsarial DR : {np.mean(a_dr)*100:.2f}% \t EIR : {eir*100:.2f}%")
```