Northumbria University NEWCASTLE

EVASION-AWARE BOTNET DETECTION USING ARTIFICIAL INTELLIGENCE

RIZWAN HAMID RANDHAWA

PhD

2023

EVASION-AWARE BOTNET DETECTION USING ARTIFICIAL INTELLIGENCE

RIZWAN HAMID RANDHAWA

A thesis submitted in partial fulfilment of the requirements of the University of Northumbria at Newcastle for the degree of Doctor of Philosophy

Department of Computer & Information Sciences

April 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original. I also declare that this work has not been submitted in whole or in part for consideration for any degree or qualification, to this or any other u niversity. This dissertation is my work and to the best of my knowledge, does not contain content which breaks any law of copyrights, except as specified in the text and acknowledgments.

Name: Rizwan Hamid Randhawa

Date: 17 April 2023

Dedications

I want to dedicate this thesis to my country Pakistan and its people, my family members and sincere friends who always supported me during this journey.

Acknowledgements

I want to thank Prof Dr Nauman Aslam and Muhamamd Alauthman for their valuable guidance, support and encouragement at every step. Without their help, this research work would not have been possible. I also want to thank Dr Hubert, who supervised and guided me for the initial year of my PhD. I also want to thank Northumbria University for providing me RDF studentship to support my PhD studies.

I want to extend my gratitude to Dr Khalid, Dr Suleman Khan, Dr Biju and Dr Xiaomin for their support during the paper submission procedures. I would also like to thank my colleagues, Husnain Rafiq, Mohammad AlJaidi and Rana Faisal Shahzad, for their constant support and valuable discussions.

Abstract

Adversarial evasions are modern threats to Machine Learning (ML) based applications. Due to the vulnerabilities in the classic ML inference systems, botnet detectors are equally likely to be attacked using adversarial examples. The evasions can be generated using sophisticated attack strategies using complex AI models. Generative AI models are one of the candidates to spawn evasion attacks. The other significant concern is the data scarcity due to which ML classifiers become biased toward the majority class samples during training. This research proposes novel techniques to improve the detection accuracy of botnet classifiers to mitigate the effects of adversarial evasion and data scarcity. The ultimate goal is to design a sophisticated botnet detector that is adversarially aware in low data regimes. First, the technical background of the research is mentioned to help understand the problem and the potential solutions. Second, a Generative Adversarial Network (GAN) based model called *Botshot* is proposed to address the dataset imbalance issues using Adversarial Training (AT). *Botshot* gives promising results in contrast to the most voguish ML classifiers that can be fooled by adversarial samples by 100%. *Botshot* improves the detection accuracy up to 99.74% after AT as the best case scenario in one of the botnet datasets used. Third, an evasion-aware model called Evasion Generative Adversarial Network (EVAGAN) for botnet detection in low data regimes is presented. EVAGAN outperforms the state-of-the-art Auxiliary Classifier Generative Adversarial Network (ACGAN) in detection performance, stability, and time complexity. Last, an improved version of EVAGAN called deep Reinforcement Learning based Generative Adversarial Network (RELEVAGAN) is bid to make the EVAGAN further hardened against evasion attacks and preserve the attack sample semantics. The rationale is proactively attacking the detection model with a Deep Reinforcement Learning (DRL) agent to know the possible unseen functionality-preserving adversarial evasions. The attacks are conducted during the EVAGAN training to adjust the network weights for learning perturbations crafted by the agent. RELEVAGAN, like its parent model, does not require AT for the ML classifiers since it can act as an adversarial-aware botnet detection model. The experimental results show the RELEVAGAN's supremacy over EVAGAN in terms of early convergence. RELEVAGAN is one step further toward designing an evasion-aware and functionality-preserving botnet detection model that tends to be sustainable with evolving botnets with the help of DRL.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview

The ever-increasing sophistication in the design of botnets is alarming for digital economies. With the increase in online transactions, mobile payments, and cryptocurrencies, botnets could be in greater demand by cybercriminals on the dark web. Soon botnets will be used to mine Bitcoins on a gigantic scale and rallied to perform more seriously detrimental DDoS attacks than ever on traditional as well as Internet of Things [1, 2]. These attacks exploit the inherent weaknesses in existing Intrusion Detection Systems (IDSs) and analysis frameworks. Machine learning (ML) has recently emerged as a powerful tool to design sophisticated IDSs for botnet detection. With the modern enhancements in hardware capabilities of computer systems, deep learning started getting tremendous attention within the research community after giving remarkable results in computer vision [3, 4]. Deep Neural Networks (DNNs) have been successfully used to solve classification problems with high accuracy. To facilitate online learning, researchers use DRL, which does not require classifiers to be trained exclusively offline for new updates [5]. However, due to vulnerabilities to adversarial attacks, the training model may learn malicious samples as benign and train itself on false negatives. This malicious effect is poisoning caused by adversarial evasion attacks [6–8]. An adversarial evasion attack can be defined as crafting the input sample to evade the ML-based detectors due to probing an IDS [9]. In this way, the behaviour of the decision boundary of an IDS can be learned and replicated to a local machine. As a result, example inputs that evade the copied learning model are generated. These evading examples are fed to the target IDS that most likely cannot classify those as malign. This crucial concern about the security of ML-based systems has also gained significant attention in botnet detection [10]. The state-of-the-art IDS for botnet detection would ideally learn such crafting by an attacker beforehand to minimise the effect of an evasion attack.

Another significant aspect discussed in this research is addressing the botnet dataset scarcity and imbalance issues in low data regimes without incurring the additional cost of generating actual attacks with multiple computers. In low data regimes, the ML classifiers over-fit the majority class and fall short in generalising the test set [11]. Various synthetic oversampling techniques have been proposed, like SMOTE_IPF, ProWSyn, and polynom_fit_SMOTE, to address the data imbalance [12, 13]; however, these techniques depend on algorithms like nearest neighbours and linear interpolation, which make them unsuitable for high-dimensional and complex probability distributions of data [14]. Deep learning models have been rigorously researched for data oversampling [14–19] only recently. There are two types of deep learning models: generative and discriminative. Generative models produce joint probability distribution $p(l|o)$, given observable examples of data o and labels l, while discriminative models work around conditional probability for classification tasks. Deep belief networks [20], Restricted Boltzmann Machines (RBM) [21], and GANs [22] are some of the examples of generative models [23].

In this research, GANs are used to generate the botnet samples to address the problems of data scarcity, imbalance and adversarial evasions. A GAN combines two different AI models competitively learning to generate realistic samples. The use of GANs in a particular way makes this research novel compared to some previous works in [15, 19, 24]. The advantage of using GANs here is that they follow the probability distribution of a minority class (botnet) to generate sister samples. When training a GAN with a binary class, these samples can be learned along with the benign class. $\mathcal{D}$ of a GAN can be used as a powerful evasion-aware botnet detector since the discriminative model learns with the generative part of a GAN. This research proposes a GAN model that could be used as an oversampler and a sophisticated evasion-aware detector for botnets. However, making a GAN-based semantic-aware model is a non-trivial problem. DRL-based models could generate semantic-aware evasion samples [25–27]; however, the similarity with the real distribution of the benign samples can not be guaranteed if the perturbation size goes beyond a threshold. Moreover, unlike GANs, a reinforcement learning-based model does not follow a particular data distribution. Hence, a combination of GAN and reinforcement learning models can provide both distribution and semantic awareness that can help address the issues of adversarial evasion attacks in an efficient manner. This research discusses a solution to the problem with concrete experimentation and reasoning around GANs and DRL to design a sophisticated botnet detection model. Two out of three contributions of this research are based on GANs to generate evasion samples to improve ML classifiers' detection and to devise an automatic adversarial learning GAN model. The third contribution of the research leverages DRL-based GAN for self-learning the evasion attacks generated using DRL.

1.2 Research Problems

Due to the imbalance in cybersecurity botnet datasets, ML-based botnet detectors suffer from detection inaccuracies and evasion attacks. The improvement in detection accuracy demands the design of sophisticated AI-based models robust against adversarial evasion attacks. There is a considerable gap between the current state-of-the-art botnet detection models and an ideal botnet detector. The solution towards achieving the goal of devising an accurate and evasion-aware botnet detection model demands more data and researching novel schemes. This thesis proposes a solution around modern AI techniques like GANs to increase the robustness with improved detection accuracies for botnet detection. GANs can also be used as oversamplers to generate more data for addressing the problem of data imbalance. At the same time, the detection models can be devised based on GANs, addressing adversarial evasion attacks. There is a broad scope of using GANs in cybersecurity due to their versatile forms, so exploring the evasion-hardened schemes for botnet detection is still evolving. However, more specifically, this research thesis highlights and addresses the following research problems:

- The scarcity of data in cybersecurity datasets is due to their sensitive nature. Usually, the malicious class data, as compared to the normal class data, are low in number. Due to this problem, the data generalisation in ML classifiers is affected. Data can be generated using attack emulation in labs. However, the emulation can be costly due to the involvement of various hardware resources. Moreover, the emulated attacks generated in a lab environment may not represent the real attack generalisation. Since the semantics of one type of generated attack may differ from another. Will it always require expensive hardware resources to generate individual attack samples to create datasets for research and development? The answer to this question is ”No”, as the attacks can be generated using synthetic oversampling. However, the oversampling methods use nearest neighbours and linear interpolation, which can be unsuitable for high-dimensional and complex probability distributions like Cybersecurity data. So which oversampling would be more suitable for such scenarios?
- Modern studies propose using GANs that have proved to be better than the traditional oversampling methods that use linear interpolation. However, how would GANs be employed as oversamplers to improve the detection accuracy of ML-based botnet classifiers?
- ML-based botnet classifiers are prone to adversarial evasion attacks as well. This problem can be due to insufficient data or model vulnerabilities. ML models have inherent vulnerabilities that can be addressed using novel data samples. This method is called adversarial training. How could the GANs be exploited to generate adversarial examples that could be used in adversarial training to improve the ML classifiers’ detection accuracy?

- The problem with adversarial training is that the data may not have a linear relationship with detection accuracy. Second, adversarial training can not guarantee a robust defence as it can be bypassed. So, can a sophisticated, evasion-aware model be devised that could simultaneously act as an intelligent botnet detector?
- In contrast to interpolation-based methods, GANs follow the probability distribution of input training data. So similar samples as given as input data are generated by GANs. However, GANs are not good at generating categorical features like IP addresses or port numbers in Cybersecurity datasets. Moreover, the synthetic samples generated by GANs may not necessarily preserve the original attack semantics of the botnet. So DRL can be employed to confine the Generator ($\mathcal{G}$)'s output so that the GAN can be made both adversarial and semantic-aware?

1.3 Research Contributions

The novel research contributions of this research are as follows:

- To address the problem of data scarcity, a GAN-based technique called *Botshot* has been proposed in Chapter 3. The unique selling point of *Botshot* is the use of *recall* in C2ST as the metric in sample evaluation during GAN training. The empirical results show that *Botshot* based oversampling outperforms the traditional oversampling methods, especially in minimising false negative rates for most of the datasets used.
- To render adversarial training needless, especially in low data regimes, a novel GAN design called Evasion Generative Adversarial Network (*EVAGAN*) is proposed. EVAGAN acts as an oversampler and a powerful evasion-aware botnet detection model. $\mathcal{D}$ of EVAGAN acts as a botnet detector and simultaneously trains itself on adversarial evasions generated by $\mathcal{G}$ of GAN. The details of the work have been explained in Chapter 4.
- Semantic preservation is ensured by using DRL-based evasion generation techniques.

However, many previous research works used adversarial training, which may be undesirable in cybersecurity due to intricate probability distributions. A novel technique called Reinforcement Learning based EVAGAN (*RELEVAGAN*) using DRL in conjunction with EVAGAN is proposed in Chapter 5 to preserve the semantics without the need for adversarial training. To the best of the knowledge, no previous research has used EVAGAN with DRL in the proposed way.

1.4 Research Publications

Journal Papers:

[1] R. H. Randhawa, N. Aslam, M. Alauthman, H. Rafiq and F. Comeau, ”Security Hardening of Botnet Detectors Using Generative Adversarial Networks,” in IEEE Access, vol. 9, pp. 78276-78292, 2021, DOI: 10.1109/ACCESS.2021.3083421.

[2] R. H. Randhawa, N. Aslam, M. Alauthman and H. Rafiq, ”Evasion Generative Adversarial Network for Low Data Regimes,” in IEEE Transactions on Artificial Intelligence, 2022, DOI: 10.1109/TAI.2022.3196283.

[3] R. H. Randhawa, N. Aslam, M. Alauthman, M. Khalid and H. Rafiq. ”Deep Reinforcement Learning based Evasion Generative Adversarial Network for Botnet Detection,” in ArXiv, 2022, abs/2210.02840.

1.5 Thesis Structure

The thesis is organised into six chapters as follows:

- **Chapter 2** discusses the problem's background in the related works. It also explains the technical details of the AI models used in this research.
- **Chapter 3** proposes *Botshot*. Section 3.2 illustrates the proposed methodology, Section

3.3 explains the implementation details. Section 3.4 shows results, Section 3.5 gives the reasoning for the inferences from the experimental outcomes and Section 3.6 concludes this chapter.

- **Chapter 4** proposes EVAGAN. Section 4.2 presents the details of the proposed model, section 4.3 gives a description of implementation details, section 4.4 demonstrates the results, section 4.5 provides an analysis of the results and section 4.7 concludes the chapter.
- **Chapter 5** proposes RELEVAGAN. Section 5.2 presents the details of the proposed model, section 5.3 gives a description of implementation details, section 5.4 demonstrates the results, section 5.4 provides an analysis of the results and section 5.6 summarises the chapter.
- **Chapter 6** concludes this thesis by discussing potential future work and open challenges.

Chapter 2

Background

This chapter discusses background literature to help the readers understand the vulnerabilities in AI-based systems in general and botnet detectors in particular. Multiple problems concerning the botnet datasets, evasion attacks, sophisticated adversarial-aware detection model and functionality preservation have been discussed.

2.1 Problems with Botnet Datasets

The importance of ML in IDS, especially botnet detection, has gained significant attention in the last decade [28–33]. The primary concern raised by the authors in [28] is that most of the research works had addressed only 25% of the modern attacks because they were based on outdated datasets. With the continuous evolution in botnet evasion techniques, it is necessary to update benchmark data sets [34]. To fulfil this objective, many researchers have proposed the generation of new data sets emulating known attacks [35–37]. However, these proposed methods of generating data sets have two drawbacks:

- Data set generation can be costly due to the involvement of multiple machines.
- The generated traffic may not depict the actual scenario of attacks [38]

The problems mentioned above can be addressed in two ways. One is to merge all the benchmark data sets publicly available into a new data set. The authors of [38] used this approach to generate a new data set, considering the common updated attacks with eleven different evaluation characteristics of IDS data sets. However, the problem of obsolescence [28] remains. There is a need to continuously update the data sets to maintain deterrence against evolving botnets [38]. Hence, for the generation of botnet datasets, unseen attack data is needed in large amounts to address the issues of dataset imbalance and obsolescence. The data generation can be opted using synthetic oversampling methods that can solve both problems.

In network-based anomaly detection for botnets, mainly two schemes are employed: deep packet-based and network flow-based inspection. Since botnets have evolved, modern types have used packet encryption. Another issue with deep packet inspection (DPI) is the severe concern about user privacy [38]. As a solution to these problems, most research is now performed using network flow-based features to detect anomalies. For botnet detection, researchers have extensively studied different flow-based features based on their effectiveness for the desired outcome [39–41]. The researchers in [42] have devised new time and flow-based traffic features for detecting Tor traffic. This extended set of features can improve the detection of the network anomaly in the case of botnet detectors as well because most botnet traffic types have underlying similarities in spacio-temporal communication patterns [27].

So not only are there scarcity, unavailability and data imbalance issues with botnet detectors, but the appropriate feature set selection can also affect the performance of detection models.

2.2 Adversarial Attacks on Botnet Detectors

Artificial Intelligence (AI) is an open-source tool that cybercriminals can compromise for the incarnation of intelligent evasions. AI-based intrusion detection systems (IDSs) need continuous online learning to keep them up to date against zero-day attacks. To address the problem of online learning, researchers use DRL to avoid training the system exclusively offline for new updates [43, 44]. However, DRL can train the model on malicious samples recognised as benign ones [45]. This type of evasion of malign samples can be caused by adversarial attacks such as model extraction or black-box attack [9]. In this type of attack, the behaviour of the decision boundary of a machine learning (ML) model can be learned and replicated to a local machine. After this step, the example inputs that evade the copied learning model are generated. These evading examples are tried on the target system that most likely is unable to classify those as malign. In [19], the authors proposed an attack and a countermeasure to demonstrate that AI-based IDSs are prone to adversarial attacks.

2.2.1 Adversarial Attacks

The adversarial ML deals with ruggedising the existing models by learning the adversarial examples [46]. This term was coined for computer vision problems but is also valid in network and computer security since both these fields are highly dependent on ML models [47–49]. The adversarial evasion can be defined as a perturbed version of an input sample x as x* such that the x* = x + η, where η is a carefully crafted perturbation. When making an adversarial attack, η could be sought and selected so that the classifier cannot discriminate the x* from x.

2.2.2 Attack Types

In general, the adversary can make two types of adversarial attacks, white-box and black-box attacks [50].

2.2.2.1 White-box Attacks

In white-box attacks, the attacker knows not only the gradient information for the model's loss function but also the source of the training data, hyper-parameters, model architectures, numbers of layers, activation functions and model weights. In other words, the attacker has complete knowledge of the model and even the direction of the gradient. The attacker can create a perturbation that most likely could increase the loss value.

2.2.2.2 Black-box Attacks

In black-box attacks, the adversary has no inside information except probing the system on a particular input. The attacker does not know the model and can only know the system's response. One of the objectives of this research work is to make the classifiers adept at proactively knowing the adversarial examples so that the effect of black-box attacks can be mitigated, especially on ML-based botnet detectors.

2.2.3 Transferability

The white-box attacks can be transferable to attack a black-box service [51]. Hence most of the attacks generated are white-box attacks due to the ease of generation, but the transferability property can be effectively exploited for black-box attacks [50, 52].

It means that if an adversarial attack is valid for one type of ML model, then it is highly likely that it will be effective against another model without added effort by the adversary [52]. To address the transferability of adversarial examples, researchers use GANs to train the existing deep learning models, with synthetic adversarial examples [19, 53]. This way, an IDS may become aware of a zero-day attack and avoid it.

Given the above discussion, GANs can be suitable for generating new botnet data to address the dataset imbalance and scarcity issues and mitigate the effects of adversarial evasion attacks.

2.3 Generative Adversarial Networks (GANs)

The GAN is a combination of two neural networks, among which the one that generates samples is called $\mathcal{G}$ and the other that evaluates the generated samples is called Discriminator ($\mathcal{D}$). The loss of $\mathcal{D}$ over generated data is fed back to $\mathcal{G}$ while $\mathcal{D}$'s weights are not updated so that $\mathcal{G}$ can try to mimic the real data probability density function more efficiently and fool $\mathcal{D}$.

2.3.1 Working of a GAN

Figure 2.1 shows the internal architecture of a classical/vanilla GAN. There are two consecutive steps in which a GAN is trained. In the first step, $\mathcal{D}$ is trained on real data labelled as REAL, and the data generated by an untrained $\mathcal{G}$ is labelled as FAKE. In the next step, now that $\mathcal{D}$ has trained already, it is tested on the fake data from $\mathcal{G}$, but this time intentionally labelled as REAL. The loss of $\mathcal{D}$ on this falsely labelled data is fed back to $\mathcal{G}$, which adjusts its weights in one complete batch training. There can be several batch iterations, after which one complete traversal of the dataset is complete, also known as an epoch. In the classical GAN, the generator model can be represented as $\mathcal{G}$: z$\longrightarrow$ $\mathcal{X}$ where z is the normal distribution from noise space and $\mathcal{X}$ is the real data distribution.

The discriminator $\mathcal{D}$: $\mathcal{X}$ $\longrightarrow$[0,1] model is a classifier that outputs an estimate of probability between 0 and 1, to mark whether the data coming from $\mathcal{G}$ is real or fake. The objective function of the combined model can be represented by Equation 2.1.

$$\begin{aligned}\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{x \sim p_{data}(x)}[\log \mathcal{D}(x)] + \\ \mathbb{E}_{z \sim p_z(z)}[\log(1-\mathcal{D}(\mathcal{G}(z)))]\end{aligned} \tag{2.1}$$

Here, $\mathbb{E}$ represents the expected value of the loss and x and z denote the real and noise samples, respectively. At the same time, p_{data} and p_z are the probability distributions of real and noise data, respectively. The objective of a min-max game is to minimise the generator's loss in creating data resembling real data. Since the generator can not control the loss of $\mathcal{D}$ on real data, still, it can maximize the loss of $\mathcal{D}$ on generated data $\mathcal{G}(z)$. The objective function $J_{\mathcal{G}}$ of $\mathcal{G}$ is given by Equation 2.2. The goal is to minimise the log-likelihood of the fake samples being classified as fake by $\mathcal{D}$.

$$J^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}_{z \sim p_z(z)}[\log(\mathcal{D}(\mathcal{G}(z)))] \tag{2.2}$$

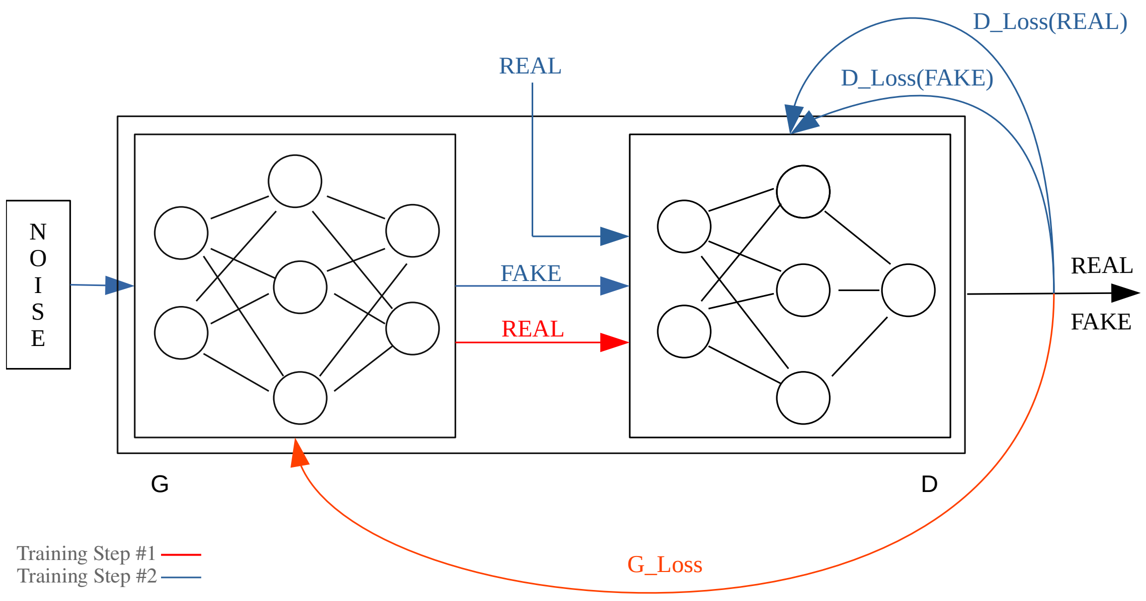


Figure 2.1: Internal Architecture of a classical GAN

As demonstrated in the Figure 2.1, the losses of $\mathcal{D}$ on real $D\_Loss(REAL)$ and generated data $D\_Loss(FAKE)$ respectively, are fed to $\mathcal{D}$ using back-propagation. In the next step, in forward propagation, given label as REAL to the input generated samples (coming from $\mathcal{G}$), the evaluation is done by $\mathcal{D}$ and $G\_Loss(REAL)$, is fed back to $\mathcal{G}$ to update its weights. This step can be called combined model training. The combined model takes noise as input and the output of $\mathcal{D}$ as the feedback to update the weights of $\mathcal{G}$. This process keeps iterating till the number of epochs reaches a set value. Upon achieving the Nash equilibrium, the generator and discriminator do not learn further.

2.4 Other GAN Models used in this Research

In the first part of this research, two different GAN architectures, vanilla and conditional GAN have been used to generate botnet traffic to keep the experiments simple for estimating their potential for botnet data generation. The second part proposed a unique GAN model based on Auxiliary Classifier GAN (ACGAN) called EVAGAN. The last part of this research proposes another enhanced GAN model called RELEVAGAN based on EVAGAN. The following is a brief detail of existing GAN architectures used in this research.

2.4.1 Conditional GAN (CGAN)

The conditional version of vanilla GAN adds another feature as a class label to help GAN learn the probability distribution quicker than vanilla GAN. This is equivalent to feeding additional information so that the convergence of the $\mathcal{G}$ can be quicker. With the class label incorporation, the CGAN generates higher-quality samples than the vanilla GAN. The loss functions for $\mathcal{D}$ and G, in this case, can be represented in Equations 2.3 and 2.4 as $J_{\mathcal{D}}$ and $J_{\mathcal{G}}$ respectively.

$$\begin{aligned} J_{\mathcal{D}} = -\frac{1}{2m}\sum_{i=1}^{m}\log \mathcal{D}(x_i, y_i) + \\ \sum_{i=1}^{m}\log(1 - \mathcal{D}(\mathcal{G}(z_i, y_i), y_i))] \end{aligned} \tag{2.3}$$

$$J_{\mathcal{G}} = -\frac{1}{m}\sum_{i=1}^{m}\log \mathcal{D}(\mathcal{G}(z_i, y_i), y_i) \tag{2.4}$$

Here, m is the total number of examples in one batch selected for training the model, x is real data, and y are the additional labels as a condition due to which this GAN is called conditional GAN. The batch size is the same for noise and real data inputs to $\mathcal{G}$ and $\mathcal{D}$, respectively.

2.4.2 Auxiliary Classifier GAN

Auxiliary Classifier GAN (ACGAN) extends a classical GAN exploiting class labels in the training process [54]. Like a classical GAN, ACGAN includes two neural networks: a $\mathcal{G}$ and a $\mathcal{D}$. In addition to random noise samples z, the input of $\mathcal{G}$ includes class labels c. Therefore, the synthesized sample from $\mathcal{G}$ in ACGAN is $\mathcal{X}_{fake} = \mathcal{G}(c, z)$, instead of $\mathcal{X}_{fake} = \mathcal{G}(z)$. In other terms, desired class label data can be generated by an ACGAN. So $\mathcal{D}$ of ACGAN works as a dual classifier for differentiating between the real/fake data and different classes of the input samples whether coming from the real source or $\mathcal{G}$.

The objective function of ACGAN consists of two parts: The first is the log-likelihood L_S of the correct source data, and the second is the log-likelihood L_C of the correct class labels. $\mathcal{D}$ is trained to maximize $L_C + L_S$ and $\mathcal{G}$ learns to maximize $L_C - L_S$. In other words, the objective of $\mathcal{D}$ is to improve the two likelihoods while of $\mathcal{G}$ is to improve only one likelihood, i.e. to improve the performance of $\mathcal{D}$ on classifying the samples as different class labels. $\mathcal{G}$ will also try to suppress the log-likelihood by $\mathcal{D}$ on a fake sample correctly classified as fake, i.e. it will try to fool $\mathcal{D}$. $\mathcal{D}$ gives both a probability distribution over sources and the class labels respectively $[P(S|\mathcal{X}), P(C|\mathcal{X})] = \mathcal{D}(\mathcal{X})$ where S are the sources (real/fake) and C are the class labels. Equations 2.5 and 2.6 denote the L_s and L_c respectively.

$$\begin{aligned} L_S = \mathbb{E}[logP(S = real|\mathcal{X}_{real})] + \\ \mathbb{E}[logP(S = fake|\mathcal{X}_{fake})] \end{aligned} \tag{2.5}$$

$$\begin{aligned} L_C = \mathbb{E}[logP(C = c|\mathcal{X}_{real})] + \\ \mathbb{E}[logP(C = c|\mathcal{X}_{fake})] \end{aligned} \tag{2.6}$$

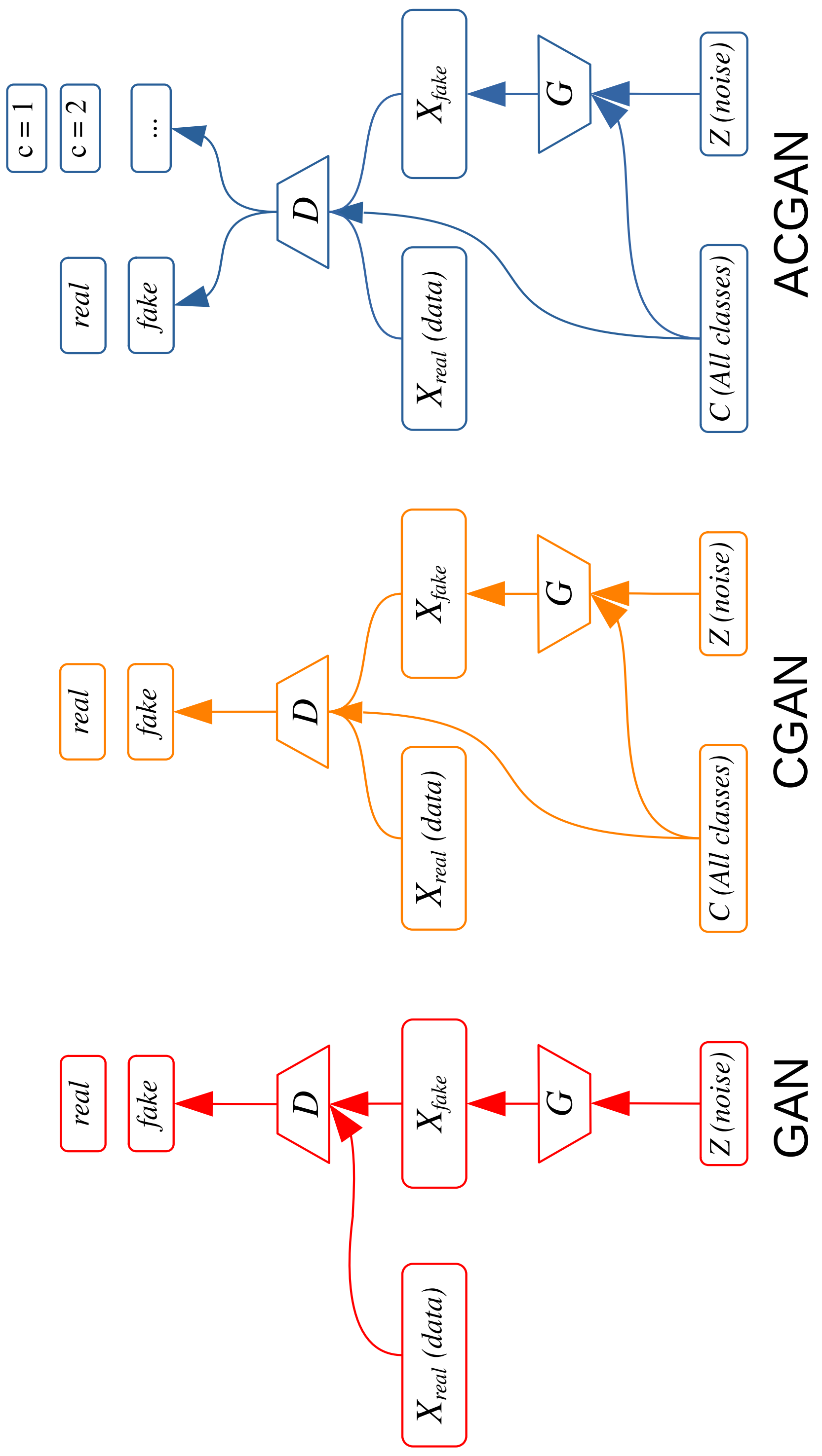


Figure 2.2: GAN vs CGAN vs ACGAN

Figure 2.2 shows a structural comparison of a classical GAN, CGAN and ACGAN.

2.5 Botnet Detectors & GANs

Currently, adversarial attacks on AI-based systems are of considerable attention in botnet detectors [10]. Adversarial examples from black-box attacks can be detrimental to an IDS because malign inputs can become part of benign network applications. The research community has highlighted the gravity of this issue [10, 55, 56]. In [57], the authors have proposed ensemble techniques to address the problem of transferability for adversarial attacks in computer vision. However, efficient and reliable IDS design techniques for most botnet behaviours are missing. For example, in [58], authors deal only with the domain generation aspect of the botnet evasion using GANs; however, not all modern botnets use DNS-based communication because these techniques are more prone to detection than peer-to-peer botnets [59]. In [15], the authors propose another GAN-based technique for botnet detection named Bot-GAN. The authors have extended the discriminator output from two to three traffic types, i.e. normal, fake botnet and real botnet. By using GANs, they have achieved better accuracy and lower false positives than previous studies. However, the feature set selection is limited in their work. In addition, the authors need to balance their data set with increasing generated examples as the accuracy drops after adding a certain number of samples. In another work, authors have proposed the avoidance of model theft or extraction [60]. The same concept can be applied in conjunction with GAN for botnet detection.

2.6 Data Oversampling & GANs

In low data regimes, oversampling or undersampling can help balance the datasets. However, undersampling might result in the loss of diversity. For oversampling, methods like Synthetic Minority Oversampling TEchnique (SMOTE) use nearest neighbours, and linear interpolation, which can be unsuitable for high-dimensional and complex probability distributions

[13, 14]. Recent research works proposed algorithms for data oversampling. Authors in [12] compared 85 different oversampling techniques and suggested the three best-performing variants as SMOTE_IPF, ProWSyn and polynom_fit_SMOTE. In [53], authors have compared the performance of these three SMOTE variants with GANs. Through empirical results, they found that GANs outperform the three mentioned oversamplers in most ML classifiers' adversarial training.

2.7 Adversarial Evasion & GANs

The decision bias in ML classifiers can lead to misclassifying malicious samples as normal. The attackers can exploit this intrinsic nature of ML classifiers to incarnate evasion samples, particularly in low data regimes. The adversarial evasion is a perturbed version of an input sample j as j* such that the j* = j + η, where η is a carefully crafted perturbation. When making an adversarial attack, η could be sought and selected so that the classifier can not discriminate the j* from j [9, 25]. The researchers usually employ adversarial training to make the classifiers proactively aware of the evasion samples. However, this is not needed if $\mathcal{D}$ of a GAN is used as a classifier to differentiate not only between the fake and real samples but also between normal and anomaly samples. The fake samples generated by $\mathcal{G}$ are also learnt simultaneously, so it is better to consider the power of $\mathcal{D}$ as an evasion-aware classifier. The use of extra ML classifiers is not needed, which is a common practice in various literary works to design such a classifier [15, 24].

To this end, EVAGAN is proposed that provides such type of $\mathcal{D}$ and compares its performance with $\mathcal{D}$ of ACGAN and other ML classifiers, xgboost (XGB), decision tree (DT), naive bayes (NB), random forests (RF), logistic regression (LR) and k-nearest neighbours (KNN). Following rigorous experimentation, it has been explored that EVAGAN's $\mathcal{D}$ not only outperforms the ML classifiers in black box testing but also gives 100% accuracy in normal and evasion samples estimation. The details of the experimental results will be discussed in section 4.4.

2.7.1 Adversarial Attacks & DRL

A plethora of seminal works have been created to deal with adversarial attacks like poisoning, evasion and transferability [7, 8, 53]. The counter defence strategies can be based on preprocessing, adversarial training, architecture, detection, defensive testing, multiclassifiers, and game theory [7]. Adversarial training seems to be a simplistic strategy to provide robustness against adversarial evasion attacks; however, it has some cons. First, increasing the number of samples in the auxiliary data may not have a linear relationship with detection accuracy [53]. Second, the adversarial training can not guarantee a robust defence as it can be bypassed [57]. Although adopted as an immediate remedy against some adversarial attacks, it can not be considered an ultimate cure for the grave problem in AI. Third, it consumes additional time for retraining. Several GAN-based research works preserve the functionality of the generated samples by manipulating only non-functional features [19, 61]. However, GANs do not play a role in generating a complete feature vector in those works. It is also quite challenging to generate categorical features using a GAN without manual engineering except using a sequence GAN [62]. Researchers have also used DRL to generate functionality-preserving evasion attacks [25–27]. The main goal of employing DRL is to explore functionality-preserving samples as it can guarantee semantic awareness in contrast to GANs; however, these works consider adversarial training to make the detection models adept at evasion awareness.

In [27], the authors proposed a DRL-based model to evade the botnet classifiers based on the output from the ML classifiers under black-box attacks. According to the authors, this work was the first on evasion attacks using a DRL agent. Following similar research, the authors in [25] proposed a technique to generate evasion attacks on the botnet detector as a black box. In [63], authors presented a new dataset generated using DRL evasion attacks on botnet detectors. The adversarial sample generation using DRL can be used with GANs to oversample the adversarial examples, which will be presented in Chapter 5, named RELEVAGAN.

2.8 Semantics/Functionality Preservation

Semantics or functionality preservation means that the synthetic samples can still execute the original malicious function intended by the attacker. Various researchers have employed GANs to synthesise the network traffic data to address the issue of data imbalance and adversarial evasion attacks [53, 64, 65]. However, GANs are not good at generating categorical features [62]. Researchers have used one-hot representation [19], or IP2Vec techniques [17, 66] to transform IP addresses into integer values to input to GANs for data generation. Since GANs follow the probability distribution of the input data so there is a high chance that the synthetic samples may lie outside the semantic limits. The perturbations could be easily perceived as anomalies by the ML detectors. To preserve the functionality, the perturbation j* in Equation 2.3 should be small enough not to be perceived by the detectors as malicious and distant enough to be considered a normal sample. At the same time, the malicious activity must not be compromised. For this reason, several researchers have proposed functionality preservation by modifying only the non-functionality-preserving features using GANs [19, 61, 62]. However, various research works claim to preserve the malicious functionality of the generated samples using DRL.

2.9 DRL-based Evasion Generation

A reinforcement learning model comprises an agent and an environment that engage for a defined number of iterations/steps. For a turn t the agent chooses an action $a_t \in \mathcal{A}$ using some policy $\pi(a|s_t)$ and an observable state s_t. The environment returns a reward $r_t \in \mathbb{R}$ against an action and a new state s_{t+1}. The reward r_t and observed state of the environment s_{t+1} are fed back to the agent for defining a new action based on policy $\pi(a|s_{t+1})$. Keeping the current state in view, the agent tries to learn through the trade-off between exploration and exploitation. The maximum reward collection is the main goal for exploration, which can be possible if the agent employs a certain policy to boost the expected value given by the Q-value function in Equation 2.7.

$$V^{\pi}(s_t) = \mathbb{E}_{a_t}[Q^{\pi}(s_t, a_t)|s_t] \tag{2.7}$$

Here Q is given by Equation 2.8.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}}[R_t|s_t, a_t] \tag{2.8}$$

and R is denoted by Equation 2.9

$$R_t = \Sigma_{i \geq 0}\gamma^i r_{t+i} \tag{2.9}$$

In Equation 2.9, $\gamma \in [0, 1]$ is a discounting factor for rewards from future actions. For catering to the large intractable storage of tabular representation of the Q-function, DNNs come to the rescue to act as an approximator in the deep Q-Network (DQN) to represent the state-action value function [67]. In RELEVAGAN, double DQN is the core technique for DRL-based attack generation.

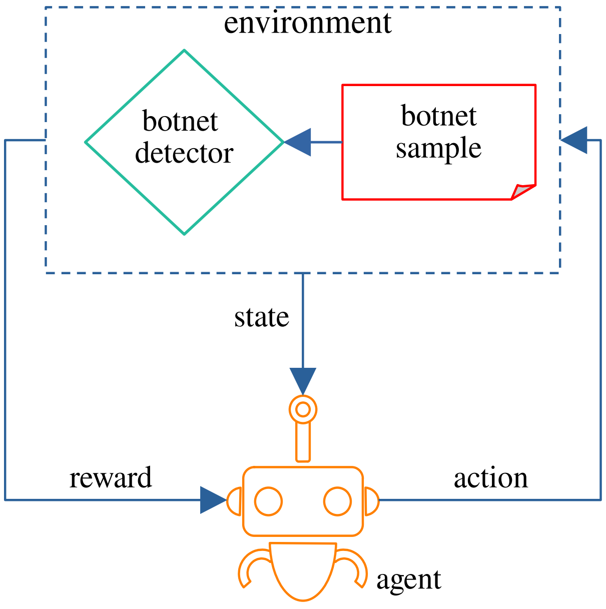


Figure 2.3: Markov Decision Process for Botnet evasion generation using Reinforcement Learning

Figure 2.3 shows the Markov Decision Process (MDP) [68] where an agent is interacting with a botnet detector environment by providing a new sample which can act as a potential evasion. The Q-function and the policy help in taking a particular action. The botnet sample is a manipulated feature vector defined within certain boundaries to ensure semantics preservation. The reward is the classification output of the botnet detector. After trying the botnet detector, the reward and the new state are fed back to the agent.

2.10 DRL-based Functionality Preservation

To preserve the malicious functionality, researchers generate the evasion samples within semantic limits [25–27, 69, 70] by attacking the trained classifiers using a DRL agent. The evaded samples were collected to be used for adversarial training to make the model adept at adversarial awareness. Similarly, in [69], authors used DRL to attack PDF (Portable Document Format) malware detectors to generate evasive malware samples while preserving the attacks' functionality. The rationale for using the DRL is that it can be bound to explore the evasion attacks within a specific range defined by ϵ in Equation 2.3. This research uses the DRL and the EVAGAN model to continuously generate the evasion attack samples using a DRL agent. The process becomes part of the EVAGAN training. The marriage of GAN and DRL culminates in RELEVAGAN, which uses the power of EVAGAN as a robust evasion-aware detection model and DRL as the semantics check on the samples generated by $\mathcal{G}$ of EVAGAN. The details of the model will be further discussed in Section 5.2.

2.11 Summary

This chapter includes the prerequisite knowledge to understand the rest of the thesis. The chapter started with current problems with botnet datasets, like scarcity and obsolescence. The problems associated with dataset generation using lab emulation were also discussed. Another significant issue with ML-based botnet detectors is their vulnerability to adversarial evasion attacks. Several previous research works were referred to highlight the problems and solutions proposed by peers. The use of GANs for improving ML-based classifiers was extensively researched. The working of a classical GAN was also explained along with two other GANs (CGAN and ACGAN) that are the basis for this research. More specifically, the use of GANs in botnet detection was discussed in some papers from which the ideas were conceived to set the foundation of this research. The use of DRL for semantic preservation of the generated data was also mentioned. DRL confines the samples based on a specific policy which can help generate semantic-aware samples. The feasibility of using DRL in conjunction with a GAN was also elaborated.

The rest of the thesis is based on the realisation of the ideas discussed in this chapter; in particular, the use of GANs to oversample the attacks data, devising a GAN-based evasion-aware detection model and use of DRL to help the detection model become semantic-aware.

Chapter 3

Security Hardening of Botnet Detectors using Generative Adversarial Networks

3.1 Introduction

Machine learning (ML) based botnet detectors are no exception to traditional ML models for adversarial evasion attacks. The datasets used to train these models have also scarcity and imbalance issues. In this chapter, a technique named *Botshot* has been proposed, based on generative adversarial networks (GANs), for addressing these issues and proactively making botnet detectors aware of adversarial evasions. In contrast to the network emulation for botnet traffic data generation, *Botshot* is cost-effective because it does not involve dedicated hardware resources. In this chapter, first, the extended set of network flow and time-based features have been used for three publicly available botnet datasets. Second, two different types of GANs (vanilla and conditional) have been utilised for generating realistic botnet traffic. The generator performance is evaluated using the classifier two-sample test (C2ST) with a 10-fold 70-30 train-test split. The use of 'recall' has been proposed in contrast to 'accuracy' for the proactive learning of adversarial evasions. The train set is augmented with the generated data and tested using the unchanged test set. Last, the results were compared with benchmark oversampling methods to augment additional botnet traffic data in terms of average accuracy, precision, recall and F1 score over six different ML classifiers. The empirical results demonstrate the effectiveness of the GAN-based oversampling for learning in advance the adversarial evasion attacks on botnet detectors.

The classifier two-sample test (C2ST) was used to assess the quality of generated traffic data. The use of 'recall' is proposed as the evaluation metric of the generator performance in C2ST instead of 'accuracy' for the purpose of learning evasions. Since the false negatives are possible evasions, the objective function should minimise the value of 'recall' during GAN training for the exploration of generator weights that generate samples with maximum similarity with the real samples, The detail of this proposition will be discussed in Section 3.2.

To maintain the coherency of the experiments, An extended flow, and time-based feature set of network traffic [10] has been utilised for three botnet datasets, i.e. ISCX-2014, CIC-IDS2017 and CIC-IDS2018, from the Canadian Institute of Cybersecurity. The traffic was generated using two different GANs. Six different classifiers (extreme gradient boosting or xgboost (XGB), random forests (RF), decision trees (DT), linear regression (LR), k-nearest neighbours (KNN) and naive bayes (NB) with default parameters) were used for generator evaluation and post augmentation detection. The main purpose of using two different GANs was to compare their performance over all three botnet datasets using the six classifiers and choose the best pair (GAN & classifier) to be used in continued research work. The six different classifiers were selected based on a related work that utilised those for comparative analysis using black-box testing [19]. The support vector machine (SVM) was not included due to its expensive training time. Instead of gradient boosting (GB), XGB was used which has a better timer complexity [71]. The neural network (NN) was not used because the discriminator in GAN is also an NN. It was desired to test the performance of GAN only on non-neural network-based ML classifiers.

The botnet traffic data generated after the GANs' training was used to augment the original train set to improve the performance of the classifiers. The tests were performed using a 10-fold, 70-30 train-test split for the aforementioned datasets for both generator performance in C2ST and post-augmentation testing. The post-augmentation performance was evaluated based on accuracy, recall, precision and F1 scores. It was demonstrated by experiments that using the proposed approach; the GAN-generated data could be better evaluated with improved botnet traffic detection .

The main essence of this chapter is the proposition of a novel technique named *Botshot*, which has the following contributions:

- *Botshot* proactively learns adversarial evasions in terms of recall score/false negatives for quantitative evaluation of GANs.
- To the best of the knowledge, this is the first work to use an extended set of features [38] for botnet data generation using GANs.
- To the best of the knowledge, no previous work has provided the comparison of GAN based oversampling with state-of-the-art synthetic oversamplers for improvement in botnet detection.

3.2 Proposed Methodology

a GAN-based technique is proposed that could mimic and generate the botnet traffic and then augment the original dataset with the generated traffic to improve the detection performance of the botnet classifiers. The purpose of training the botnet classifiers on GAN-generated data is to proactively learn the unseen samples with similar but slightly different probability distributions. *Botshot* is novel in a way that recall is used as the evaluation metric for C2ST instead of accuracy. The intuition is that the false negatives need to be reduced in the classifier performance in post-augmentation testing. The false negatives are the possible evasions in the test set that the classifiers are not aware of. Since the primary purpose of the *Botshot* is to suppress the false negatives; the C2ST has been tweaked so that the objective function becomes as given in the Equation 3.1

$$\hat{r}_{argmin} = \frac{1}{n_{test}} \sum_{z_i, l_i \in D_{test}} \mathbb{I}[\mathbb{I}(f(z_i) > \frac{1}{2}) = l_i] \tag{3.1}$$

In the above Equation, $\hat{r}$ is the recall score on D_{test} which is test set, n_{test} is the total number of samples in test set, z_i are the samples in test set, l_i are the labels, $f(z_i)$ is the conditional probability distribution $p(l_i = 1|z_i)$. If a GAN-bot is very close in probability distribution with a real bot, the recall in Equation 3.1 should remain close to 50%, which is also called a chance level, which means that the classifier was evaded, or the sample was misclassified as a real bot. So if recall is used as the metric instead of accuracy, the false negatives can be better minimised because accuracy includes the value for false positives (FP) and true negatives (TN) given by Equation 3.2.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

However, in the recall, there are only true positives (TP) and false negatives (FN) as given by Equation 3.3.

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

3.2.1 C2ST for GAN Evaluation

The classifier two-sample test (C2ST) is a quantitative metric to evaluate whether two different data samples have been taken from the same distribution. In other words, if there are samples real_bots (X_b) and GAN_bots ($G(z)$), then it can be assessed if both samples have similar or identical probability distributions. The more the distributions overlap, the greater the chance that GAN_bots are realistic. The C2ST method has been shown in Algorithm 1.

Algorithm 1 C2ST Algorithm

```
Require: $\mathcal{X}_b$ (real_bots), $G(z)$ (GAN_bots), botnet_classifier
Ensure: $\mathcal{A}$(accuracy)
1: {Evaluation of Generated Data ($G(z)$)}
2: $t_r \leftarrow \mathcal{X}_b[0 : m(7/10)] \cup G(z)[0 : m(7/10)]$ {Create a train set from 70% of examples in
   the unified set where m = total number of examples in $\mathcal{X}_b$} and $G(z)$
3: $t_s \leftarrow \mathcal{X}_b[m(7/10) : m] \cup G(z)[m(7/10) : m]$ {Create a test set from 30% of examples in
   $\mathcal{X}_b$ and $G(z)$}
4: train botnet_classifier on $t_r$
5: test botnet_classifier on $t_s$
6: compute $\mathcal{A}$ = (TP + TN)/(TP + TN + FP + FN){Compute accuracy using confusion matrix}
```

Since the objective function is to minimise false negatives in generator evaluation, those epochs must be chosen in which the recall was the lowest instead of the epochs where accuracy was the lowest.

The complete functional block diagram of *Botshot* is illustrated in Figure 3.1. First, the datasets are preprocessed as mentioned in Section 3.3.6. From the cleaned train set, the real bot samples of the set size for each dataset are extracted as mentioned in Table 3.1 and train the selected GAN. The GAN is trained for a certain number of epochs as given in Table 3.3. Once an epoch is completed, the trained $\mathcal{G}$ is used to generate data of size equal to the total number of real_bots given as an input to the GAN (Algorithm 2 line 16). The generated bot samples, also called GAN_bots, along with real_bots, are combined into a unified set (Algorithm 2 line 17). This unified set is reshuffled and used to perform 10-fold train-test splitting using the selected classifier (Algorithm 2 lines 18-24).

The results are compiled based on accuracy for C2ST and recall for *Botshot* (Algorithm 2 line 25-26). The epoch numbers for each classifier's minimum values for recall and accuracy are saved. After generating the GAN-bots, using the epoch number with minimum accuracy value for C2ST and minimum recall value for *Botshot*, these are augmented into two sets $\mathcal{Y}_{C2ST}$ and

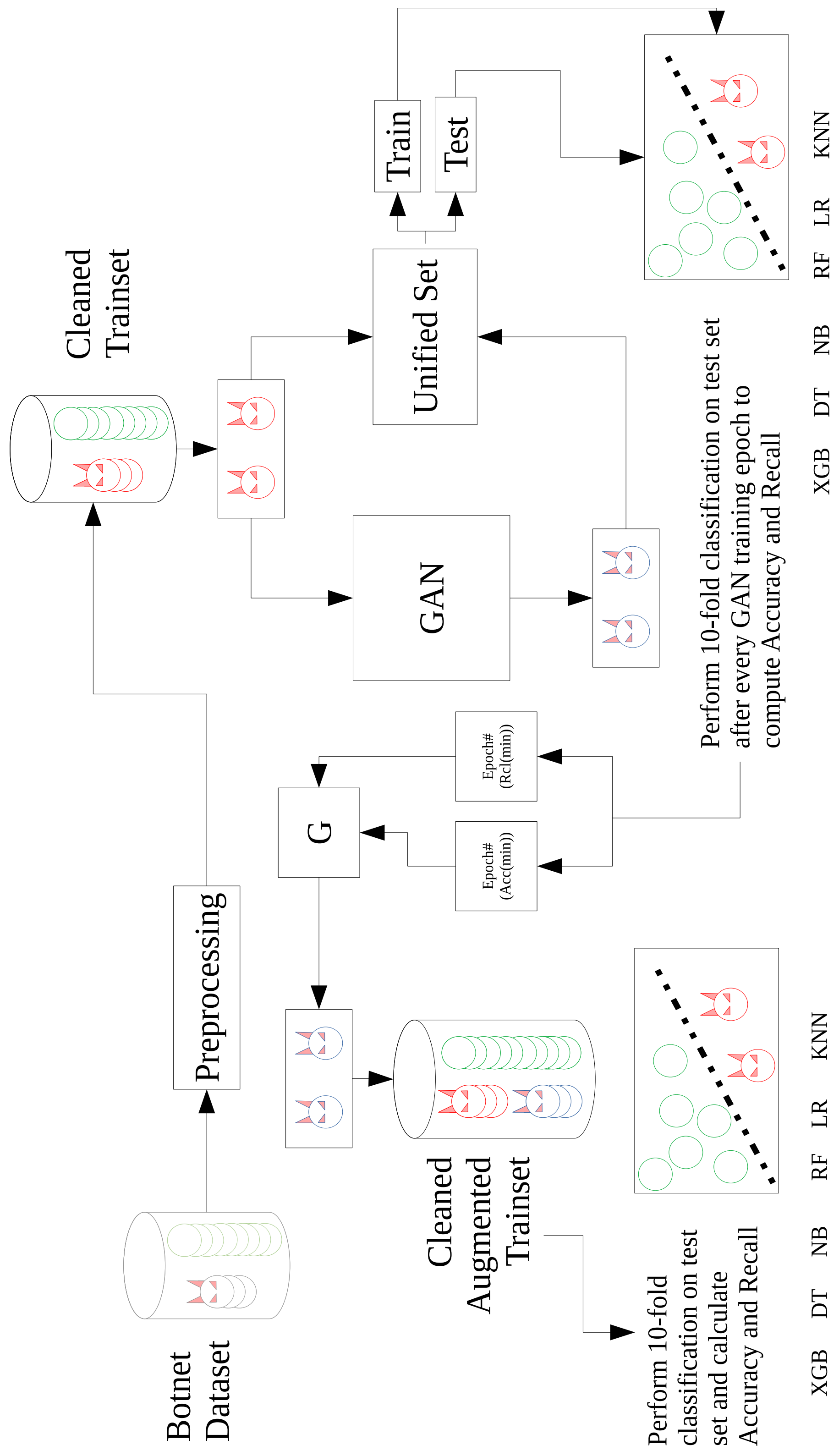

Botnet Dataset
Preprocessing
Cleaned Trainset
Unified Set
Train
Test
GAN
G
Epoch# (Acc(min))
Epoch# (Rcl(min))
Cleaned Augmented Trainset
Perform 10-fold classification on test set and calculate Accuracy and Recall
Perform 10-fold classification on test set after every GAN training epoch to compute Accuracy and Recall
XGB
DT
NB
RF
LR
KNN

Figure 3.1: Functional diagram of *Botshot*

Algorithm 2 Bothshot Algorithm

```
Require: $\mathcal{T}$ (preprocessed train set in csv format), $GAN_{type}$, batch_size, botnet_classifier
Ensure: $\mathcal{A}$ (average accuracy), $\mathcal{R}$ (average recall), $\mathcal{P}$ (average precision), $\mathcal{F}$ (average F1)
 1: $X_b \sim \mathcal{T}$ {Extract real_bots from the data set $\mathcal{T}$}
 2: Create $\mathcal{D}$, $\mathcal{G}$ and $C$ models{Define and compile discriminator, generator and combined models for the GAN}
 3: if $GAN_{type}$ is equal to CGAN then
 4:     $\mathcal{X}_b[Label] \leftarrow$ K-Means labels {Replace the Label column with k-means class labels}
 5: end if
 6: {Adversarial Training GAN}
 7: for i = 1, 2, 3, ..., epochs do
 8:     for j = 1, 2, 3, ..., batches do
 9:         $x_i \sim \mathcal{X}_b$ {Extract botnet data of batch_size}
10:         $z_j \leftarrow \mathcal{N}_{\{mean=0, std=1, size=batch\_size\}}$ {Extract noise from normal distribution of batch_size}
11:         $g_{z_j} \leftarrow \mathbb{E}_{z_i \sim p(z_j)}$ {Probability Estimation from $\mathcal{G}$ using $z_j$}
12:         $\theta_{\mathcal{D}j} \leftarrow \theta_{\mathcal{D}j} - \eta \nabla \theta_{\mathcal{D}j} \mathcal{L}(x_j)$ {Train $\mathcal{D}$ on $x_j$ with Adam optimizer}
13:         $\theta_{\mathcal{D}j} \leftarrow \theta_{\mathcal{D}j} - \eta \nabla \theta_{\mathcal{D}j} \mathcal{L}(g_{z_j})$ {Train $\mathcal{D}$ on $g_{z_j}$ with Adam optimizer}
14:         $\theta_{\mathcal{G}j} \leftarrow \theta_{\mathcal{G}j} - \eta \nabla \theta_{\mathcal{G}j} \mathcal{L}(z_j)$ {Train $C$ (combined model) on $z_j$ with Adam optimizer}
15:     end for
16:     $\theta_{\mathcal{G}i} \leftarrow \theta_{\mathcal{G}j}$ {Save the weights of $C$ after every epoch}
17:     $z_i \leftarrow \mathcal{N}_{\{mean=0, std=1, size=sizeof(\mathcal{X}_b)\}}$ {Extract noise from normal distribution with botnet data size}
18:     $G_{z_i} \leftarrow \mathbb{E}_{z_i \sim p(z)}$ {Probability Estimation from $\mathcal{G}$ using $z_i$}
19:     {Evaluation of Generated Data ($g_z$)}
20:     U = $\mathcal{X}_b \cup G_{z_i}$
21:     for k= 1, 2, 3, ..., 10 do
22:         split_pointer = k
23:         $t_r \leftarrow$ 70% of U {Create a train set from 70% of examples in unified set }
24:         $t_s \leftarrow$ 30% of U {Create a test set from 30% of examples in unified set }
25:         train botnet_classifier on $t_r$ test botnet_classifier on $t_s$
26:         compute $\mathcal{A}_i$, $\mathcal{R}_i$ {Compute accuracy and recall using confusion matrix}
27:     end for
28:     $L_{\mathcal{A}} \leftarrow$compute average $\mathcal{A}$ {Compute average accuracy and save into a list}
29:     $L_{\mathcal{R}} \leftarrow$compute average $\mathcal{R}$ {Compute average recall and save into a list}
30: end for
31: $z \leftarrow \mathcal{N}_{\{mean=0, std=1, size=Normal-real\_bots\}}$ {Extract noise from normal distribution of size = (Normal - real_bots)}
32: $Gz_{\mathcal{I}_{argmin}\mathcal{A})} \leftarrow \mathbb{E}_{z \sim p(z)}$ {Probability Estimation from $\mathcal{G}$ using $z_j$}
33: $\mathcal{Y}_{C2ST} \leftarrow \mathcal{T} \cup Gz_{\mathcal{I}_{argmin}(\mathcal{A})}$ {Augment the original dataset $\mathcal{T}$ with GAN_bots generated based on the epoch of GAN training with minimum value of accuracy }
```

```
34: $z \leftarrow \mathcal{N}_{\{mean=0,std=1,size=Normal-real\_bots\}}$ {Extract noise from normal distribution of size
    = (Normal - real_bots)}
35: $Gz_{\mathcal{I}_{argmin}(\mathcal{R})} \leftarrow \mathbb{E}_{z \sim p(z)}$ {Probability Estimation from $\mathcal{G}$ using $z_j$}
36: $\mathcal{Y}_{Botshot} \leftarrow \mathcal{T} \cup Gz_{\mathcal{I}_{argmin}(\mathcal{R})}$ {Augment the original dataset $\mathcal{T}$ with GAN_bots generated
    based on the epoch of GAN training with minimum value of recall }
37: for K= 1, 2, 3, ..., 10 do
38:     split_pointer = K
39:     $T_r \leftarrow$ 70% of $\mathcal{Y}_{Botshot}$ {Create a train set from 70% of examples in unified set }
40:     $T_s \leftarrow$ 30% of $\mathcal{Y}_{Botshot}$ {Create a test set from 30% of examples in unified set }
41:     train botnet_classifier on $T_r$ test botnet_classifier on $T_s$
42:     compute $\mathcal{A}_K, \mathcal{R}_K, \mathcal{P}_K$ and
43:     $\mathcal{F}_K$ {Compute accuracy, recall, precision and F1 score using confusion matrix}
44: end for
45: compute average $\mathcal{A}$, $\mathcal{R}$, $\mathcal{P}$ and $\mathcal{F}$ {Compute average accuracy, recall, precision and F1
    score}
```

$\mathcal{Y}_{Botshot}$ respectively (Algorithm 2 lines 27-32). Finally, the average accuracy, recall, precision, and F1 scores are computed after the 10-fold train-test split of the two augmented datasets. The values have been recorded in Table 3.4 for the case of no augmentation, interpolation-based synthetic oversamplers, GAN and CGAN-based C2ST and *Botshot* evaluation for oversampling the botnet data.

3.3 Implementation

3.3.1 Datasets

Three different datasets were used from the same source, i.e. Canadian Institute of Cybersecurity (CIC), to keep the coherency of the experiments. The choice of the dataset is based on the rigorous use in existing literature, public availability and the support for the open-source feature extraction tool called CICFlowmeter-v4 provided by CIC. Table 3.2 shows the features used in this work (in bold text). The features can be categorised into different groups based on the number of packets, time, size, flow and flags. The categorical features like IP addresses or port numbers were not used, while a previous work [15] did use protocol. Although previously used for Tor traffic characterisation [42], and data set generation for CIC-IDS2017 and CIC-IDS2018

[38], these extended sets of features can improve the detection of the network anomaly in case of botnet detection as well [27]. To the best of the knowledge, this is the first work to use these extended features for botnet datasets for data generation using GANs. The number of samples of benign vs botnet is mentioned in Table 3.1. The following is the detail of each dataset and the botnet samples used in this work.

3.3.2 ISCX-2014 Dataset

The ISCX-2014 data set [72] is an amalgamation of three publicly available datasets ISOT [39], ISCX 2012 IDS [40] and CTU-13 [73]. The composition of the dataset has been explained in terms of generality, realism, and representativeness. The generality can be defined as the richness of diversity of botnet behaviour. Realism is the closeness with the actual traffic captured, and representativeness determines the ability to reflect the real environment a detector would face in deployment. Only the VIRUT botnet was used for this work among the seven different botnets. The VIRUT was chosen because it had very few samples compared to other botnets and was not insufficiently scarce as the Zeus bot. SMTP or NSIS could be used, but their labels were not available on the website[1]. As a result, a subset including all the normal traffic flows with VIRUT samples was used. In this way, this dataset could be used as a good example of an unbalanced set.

3.3.3 CIC-IDS2017 Dataset

The CIC-IDS2017 is a relatively recent dataset by CIC with the traffic of the ARES botnet. On Friday, July 7, 2017, the traffic was collected from 10:02 AM – 11:02 AM in the CIC facility. The dataset is available on the CIC website[2]. a subset of this dataset was made using all the normal flows with the botnet. The ratio of the number of samples has been mentioned in Table 3.1. As the previously mentioned dataset, the subset is also a good kind of unbalanced dataset for the particular research problem of this work.

[1]https://www.unb.ca/cic/datasets/botnet.html
[2]https://www.unb.ca/cic/datasets/ids-2017.html

3.3.4 CIC-IDS2018 Dataset

To create another subset of the unbalanced dataset, CIC-IDS2018 was used. This dataset included samples for ARES and ZEUS botnets. A subset of all the benign traffic samples was created with just 2560 samples of botnet flows to mimic another unbalanced dataset.

Table 3.1: Distribution of Normal and Bot samples in three Engineered Datasets

Dataset	Normal	Real_bots	Total Samples
ISCX-2014	246929	VIRUT: 1748	248677
CIC-IDS2017	70374	ARES: 1956	72330
CIC-IDS2018	390961	ARES/ZEUS:2560	393521

3.3.5 Feature Selection

The performance of the botnet detectors can be highly dependent on the quality of the dataset in general and the number of distinct features in particular. Using a limited feature set may not necessarily give a stronger classification than an enhanced set of non-redundant features. Previously, [72] summarised the most important features from network flows that could be helpful in botnet detection. Almost all these features except a few are included in [10] that are also used in this thesis. A utility used by the authors was also employed, available on GitHub named CICFlowMeter-v4, to extract more than 80 flow & time-based features from the three different datasets' .pcap files. This utility can extract the said features for any input .pcap files. Table 3.2 shows the feature set. The detail of the features is available on the source website[3].

3.3.6 Preprocessing

For preprocessing the ISCX-2014 dataset, one may need to label it. For this purpose, the information provided on the CIC website for IPs associated with the particular botnets was used. After labelling, the preprocessing was performed as mentioned in Algorithm 3. All the high and low skewed values were removed to suppress outliers, cleaned the NaN and Inf values, removed

[3]https://www.unb.ca/cic/datasets/ids-2018.html

Table 3.2: Existing vs Extended Feature Set: The features can be categorised into five different groups. Features in the bold text were used in the current work, and the rest were dropped in preprocessing using Algorithm 2.

Group	**Features [15]** (used in Existing Work)	**Extended Features [38]** (used in current work)
Categorical	Protocol	-
Number of packets	PX, NNP, PSP, IOPR, Reconnect	**fin_cnt**, **syn_cnt**, **rst_cnt**, **psh_cnt**, **ack_cnt**, urg_cnt, **cwe_cnt**, **ece_cnt**, **down_up_ratio**, fw_byt_blk_avg, fw_pkt_blk_avg, fw_blk_rate_avg, **bw_byt_blk_avg**, **bw_pkt_blk_avg**, **bw_blk_rate_avg**, **subfl_fw_pk**, **subfl_fw_byt**, subfl_bw_pkt, **subfl_bw_byt**, **fw_win_byt**, **bw_win_byt**, **fw_act_pkt**, **fw_seg_min**
Time	Duration	**fl_dur**, **fw_iat_tot**, **fw_iat_avg**, **fw_iat_std**, **fw_iat_max**, **fw_iat_min**, **bw_iat_tot**, **bw_iat_avg**, **bw_iat_std**, **bw_iat_max**, **bw_iat_min**, atv_avg, atv_std, atv_max, atv_min, , **idl_avg**, **idl_std**, **idl_max**, **idl_min**, **idl_min**
Size	FPS, TBT, APL, DPL, PV	**tot_fw_pk**, **tot_bw_pk**, **tot_l_fw_pkt**, **tot_l_bw_pkt**, **fw_pkt_l_max**, **fw_pkt_l_min**, **fw_pkt_l_avg**, **fw_pkt_l_std**, **Bw_pkt_l_max**, **Bw_pkt_l_min**, **Bw_pkt_l_avg**, **Bw_pkt_l_std**, **fw_hdr_len**, **bw_hdr_len**, **pkt_size_avg**, **fw_seg_avg**, **bw_seg_avg**
Flow	BS, PS, AIT, PPS	**fl_byt_s**, **fl_pkt_s**, **fl_iat_avg**, **fl_iat_std**, **fl_iat_max**, **fl_iat_min**, **fw_pkt_s**, **bw_pkt_s**, **pkt_len_min**, **pkt_len_max**, **pkt_len_avg**, **pkt_len_std**, **pkt_len_va**,
Flags	-	**fw_psh_flag**, bw_psh_flag, fw_urg_flag, bw_urg_flag

the features with zero standard deviation, and scaled those in the [0,1] range. The [0,1] scaling is necessary to apply GANs with rectified linear unit (ReLU) activation function in its layers because the range of the output with the ReLU function lies within this range. The CIC-IDS2017 and CIC-IDS2018 were already labelled. So the preprocessing was performed only for these two data sets after extracting the unbalanced subsets.

Algorithm 3 Preprocessing

```
Require: $\mathcal{X}$ (original train set in csv format)
Ensure: $\mathcal{T}$ (preprocessed train set in csv format)
 1: $\mathcal{X}_n \leftarrow \mathcal{X}[Label = 0], \mathcal{X}_b \leftarrow \mathcal{X}[Label = 1]$ {label flows based on malicious IPs}
 2: Preprocess Data {remove: outliers, rows with NaN and Inf values, columns with std=0 and
    scale the features in range(0,1)}
```

3.3.7 GAN Hyperparameters Tuning

The GAN tuning can be challenging, primarily working on tabular data, because, unlike images, the GAN performance needs to be assessed based on some quantitative parameters (one of which is C2ST). However, if a classical GAN is tuned carefully [74], it can work to generate realistic samples as it was done in the particular case. After a random search of batch size and the multiple for the number of neurons in hidden layers of $\mathcal{G}$ and $\mathcal{D}$, the optimum values could be explored as highlighted in Figure 3.2. The choice is based on the stability of losses of $\mathcal{D}$ and $\mathcal{G}$. The rest of the hyperparameter details are in Table 3.3. The densely connected feed-forward neural networks were used as the architecture in both $\mathcal{G}$ and $\mathcal{D}$. The activation functions are ReLU in all the hidden layers and sigmoid in the output layer of $\mathcal{D}$. The batch normalisation was used in all the hidden layers. This technique regularises the output of each hidden layer and stabilises the GAN loss. The optimiser type used was Adam, with binary cross-entropy (BCE) as the loss function. The learning rate was set as 5e-1 because classifiers tend to get fooled more at this learning rate than at lower values. So it can also be considered an exploratory value in the case.

Figure 3.2 shows that the values for the three losses are stable in case of a batch size of 256 and multiplier (n) equal to 64 for both GAN and CGAN. In all the other cases, the values are overshot or not converging. These results were taken in the ISCX-2014 dataset and considered for the other two datasets. However, the individual loss values for the three datasets are manifested for each GAN in Figures 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11. The GAN losses for each GAN are different for every data set used. Red-coloured peaks are shown where $\mathcal{G}$ has a high loss L[G(z)]; then damps down in the next few epochs and remains close to zero. This result means that $\mathcal{G}$ has achieved Nash equilibrium, but it can be observed that more peaks arrive in which $\mathcal{D}$'s loss L[D(G(z))] also is increased along with L[G(z)] and sometimes even higher. This result happened when $\mathcal{G}$ generated some sample which fooled $\mathcal{D}$ pretending to be real, but it was fake. Most of the time, during the training of the desired epochs, the GAN losses remain stable. The result could be very different if the batch normalisation is not used, which regularises values in each hidden layer.

3.3.8 Data Augmentation

An unlimited amount of data could be generated from GAN's generator virtually. However, an equal number of the GAN-bot samples were generated to the benign traffic samples. The generated samples were augmented into the 70% train set only while 30% of the test set remains

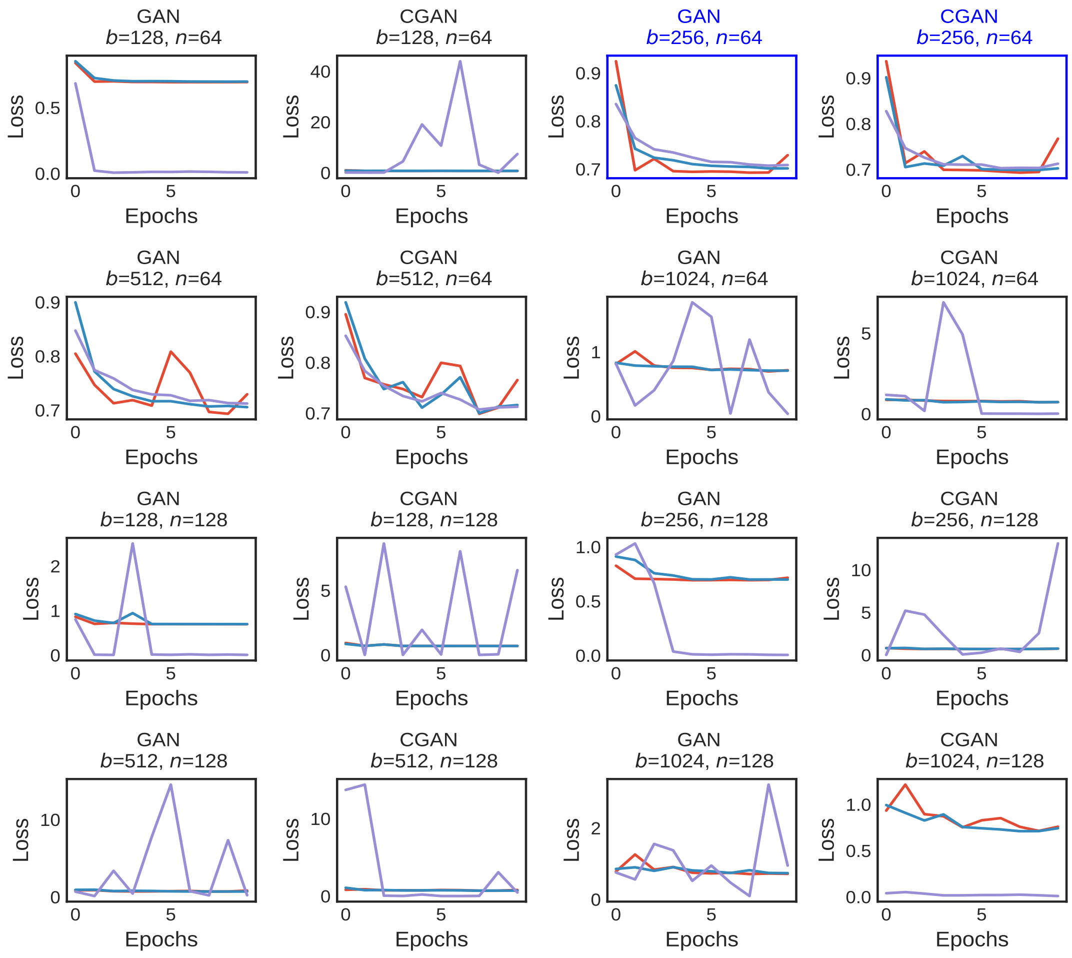


Figure 3.2: Hyperparameters Exploration for batch size (b) and hidden layer neurons count multiple (n)

Table 3.3: GAN Models

Parameter	GAN	CGAN
Network Type	Densely Connected Feed Forward	
Number of Layers	$\mathcal{G}$: 6, $\mathcal{D}/\mathcal{C}$: 5	
Activations	$\mathcal{G}$: ReLU (All Layers), $\mathcal{D}$: ReLU (All Layers except output)	
Batch Size (b)	256	
Multiplier (n)	64	
Neurons in the input layer	64	65
Neurons in layer 1	$\mathcal{G} : n \times 1 = 64$, $\mathcal{D} : n \times 4 = 256$	
Neurons in layer 2	$\mathcal{G} : n \times 2 = 128$, $\mathcal{D} : n \times 2 = 128$	
Neurons in layer 3	$\mathcal{G} : n \times 4 = 256$, $\mathcal{D} : n \times 1 = 64$	
Neurons in layer 4	$\mathcal{G} : n \times 8 = 512$	
Neurons in output layer	$\mathcal{G} : 64$, $\mathcal{D} : 1$	$\mathcal{G} : 65$, $\mathcal{D} : 1$
Layer Regularization	$\mathcal{G}, \mathcal{D}$: $BatchNorm$	
Optimizer	Adam (beta_1=0.5, beta_2=0.9)	
Loss Function	binary cross entropy	
Learning Rate	5e-1	

unchanged for each k-fold split where k is equal to 10 in the proposed case.

3.3.9 Experimental Setup

The experiments were performed on a GPU workstation AMD Ryzen threadripper 1950x 16-core processor with three GeForce GTC 1070 Ti units running ubuntu 20.04. The platform used was the Jupyter notebook with libraries mainly Keras, TensorFlow, Sklearn and NumPy.

3.4 Results

The results are demonstrated in Table 3.4 for the performance evaluation of six classifiers after the augmentation of data generated from different data generation techniques. The case of no augmentation was used as the benchmark to compare with the oversampling-based techniques. For making a performance comparison, the three variants of the best performing SMOTEs among 85 as suggested by the author in [12] were employed. The results of these oversamplers have been summarised in Table 3.4 along with the GAN and CGAN-based C2ST and *Botshot* oversampling. The post-augmentation results for the performance evaluation metrics used were accuracy, recall, precision and F1 score. Figures 3.3, 3.4 and 3.5 show the average performance of all the classifiers after augmenting the train set with GAN-bots in a 10-fold train-test split for the ISCX-2014, CIC-IDS2017 and CIC-IDS2018 datasets and testing on the fixed 30% test set. The following sections will discuss the results in detail in terms of the performance metrics of the classifiers used:

3.4.1 Accuracy

The accuracy values for both GAN and CGAN based oversampling are almost always greater than or equal to the no-augmentation values and, in some cases, more than SMOTEs. Specifically, the value of accuracy in the case of ISCX-2014 for GAN-based *Botshot* is higher than GAN-based C2ST but equal in the case of CIC-IDS2017 and CIC-IDS2018. As per equation 3.2, the accuracy is the ratio of the sum of TP and TN to the total number of samples (TP+TN+FP+FN) in the confusion matrix, so due to the large number of TN (benign samples), the improvement in post augmentation results may not be significantly more accurate. So the decisive metrics could be recall and precision.

3.4.2 Recall

The recall improvement is the highest value proposition of this research work because the effects of adversarial evasions need to be mitigated. However, the recall could not be improved significantly for all classifiers for the three datasets. In fact, for NB and KNN, the recall deteriorated adversely, especially in the case of the ISCX-2014 dataset. This behaviour of the classifiers is specific to their inherent properties, the explanation of which is beyond the scope of this research. In the case of CIC-IDS2017, the recall was improved compared to the benchmark for all the GAN and CGAN-based oversampling methods. This proves the effectiveness of the *Botshot* as compared to C2ST as well. The values for recall in almost all cases for *Botshot*-based GAN/CGAN oversampling is greater than or equal to C2ST-based GAN/CGAN oversampling except in the case of CGAN(Botshot) with recall equal to 98.65% as compared to 98.61% of CGAN(C2ST) for CIC-IDS2017 dataset. It can be confidently inferred that *Botshot* is competitive not only with the SMOTE based oversamplers in the case of CIC-IDS2017 and CIC-IDS2018 datasets but, in some instances, better for particular classifiers like XGB, which almost always shows improvement in recall for almost all the oversampling techniques as compared to the benchmark. This information can significantly help us choose the suitable classifier to be used in *Botshot* in future works.

3.4.3 Precision

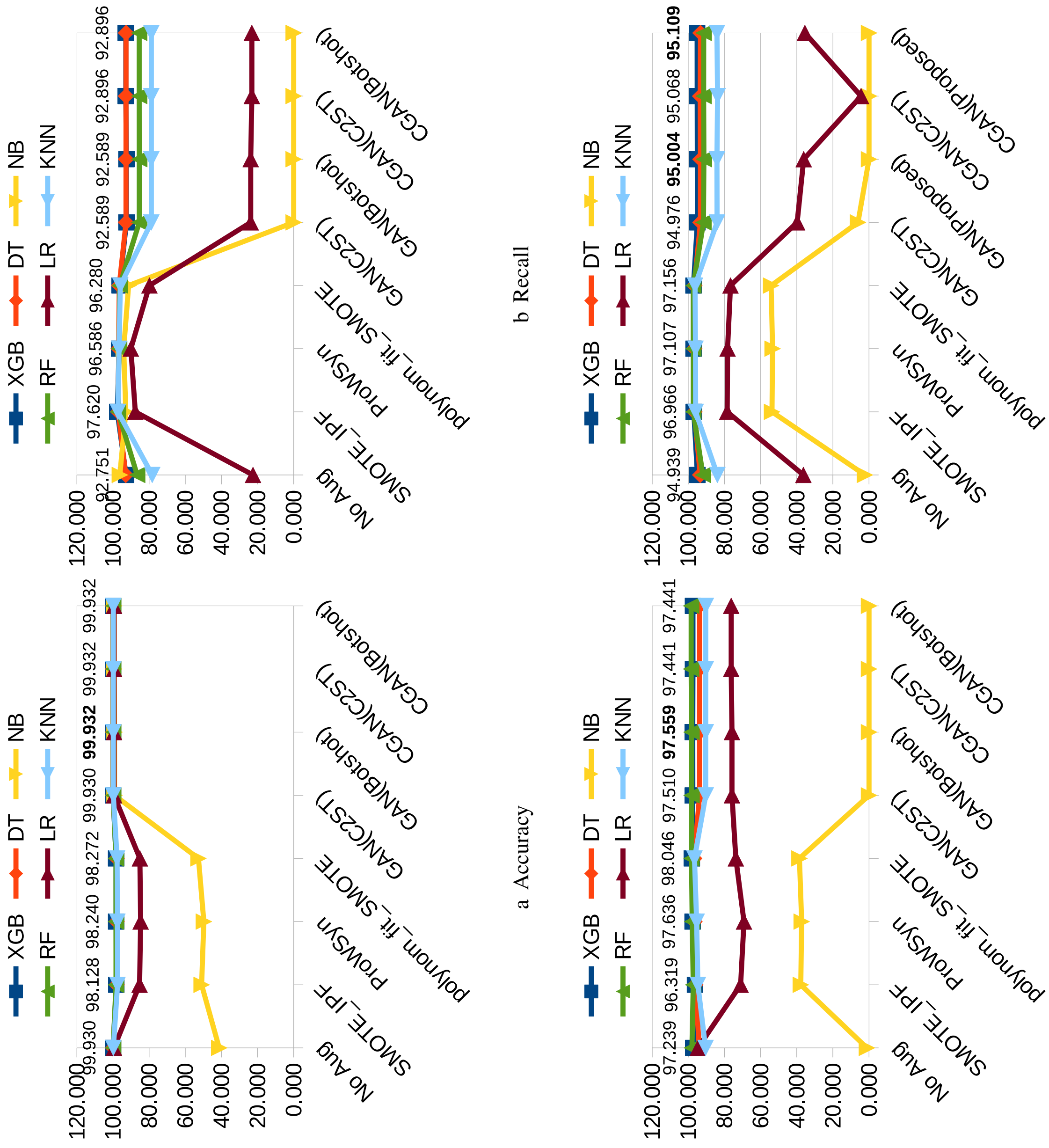
The minimisation of false positives is specified as the property of a classifier being precise in its classification decisions. The precision values for almost all the classifiers have significantly higher values as compared to the SMOTE based techniques, which gives us strong confidence in estimating *Botshot* for reducing the FP except for NB, which has the worst performance in the case of all the three datasets. However, the precision value for GAN(Botshot) is higher than GAN(C2ST) and equal in the case of CGAN for the ISCX-2014 dataset. Similarly, the value for CGAN(Botshot) is higher than CGAN(C2ST) but lower than GAN(C2ST) for GAN(Botshot) for the CIC-IDS2017 dataset. One higher precision vote also goes to GAN(Botshot) than GAN(C2ST), while the values remain the same for CGAN. The improvement in precision was unexpected since the main objective was to reduce FN and not FP. GAN-based oversampling has helped the classifiers make better decisions about the benign traffic samples, which is an added benefit of the proposed technique.

3.4.4 F1

The F1 score is the harmonic mean of the precision and recall, reflecting the combined effect of both these parameters. It can be observed from the results that F1 in the case of CIC-IDS2017 and CIC-IDS2018 is better for *Botshot* in the majority of cases as compared to the C2ST-based GAN/CGAN oversampling and even better than the benchmark in some instances.

Table 3.4: Results of Data Augmentation for Different Oversampling Techniques: The original train set was augmented with botnet traffic samples generated from each of the different synthetic oversampling techniques, and the testing was done on the original test set.

Results on ISCX-2014 Dataset																								
	Accuracy						**Recall**						**Precision**						**F1**					
	XGB	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**
No. Aug	99.930	99.907	41.231	**99.895**	**99.443**	99.788	92.751	92.920	**96.222**	86.634	22.500	78.288	97.239	93.935	1.144	97.995	**94.676**	90.456	94.939	93.418	2.261	91.960	36.341	83.926
SMOTE_IPF	98.128	97.920	50.940	98.305	85.285	97.490	**97.620**	**97.102**	92.902	**97.200**	87.648	**97.271**	96.319	96.147	37.763	97.262	71.057	94.676	96.966	96.621	53.699	97.232	**78.487**	95.958
ProWSyn	98.240	97.978	49.711	98.319	84.683	97.552	96.586	96.825	94.066	96.636	**90.106**	96.662	97.636	96.578	37.277	97.849	69.187	95.402	97.107	**96.701**	53.393	97.242	78.271	96.028
polynom_fit_SMOTE	98.272	97.959	52.708	98.316	85.110	97.723	96.280	96.542	91.351	96.229	79.837	95.927	**98.046**	**96.787**	**38.522**	98.237	73.715	**96.612**	**97.156**	96.662	**54.191**	**97.223**	76.653	**96.268**
GAN(C2ST)	99.930	99.904	99.132	99.887	99.409	99.791	92.589	92.783	0.000	85.470	23.751	78.785	97.510	93.737	0.000	98.232	75.859	90.345	94.976	93.288	6.078	91.625	39.796	84.096
GAN(Botshot)	99.932	**99.908**	99.132	99.887	99.409	**99.791**	92.589	92.783	0.000	85.470	23.751	78.785	97.559	93.737	0.000	98.232	75.859	90.345	95.004	93.272	0.000	91.398	36.168	84.163
CGAN(C2ST)	99.932	99.904	99.292	99.887	99.405	99.790	92.896	92.783	0.000	85.506	23.139	78.780	97.441	93.737	0.000	98.422	76.282	90.341	95.068	93.425	0.000	91.540	4.387	83.827
CGAN(Botshot)	**99.932**	99.904	**99.292**	99.887	99.405	99.790	92.896	92.783	0.000	85.506	23.139	78.780	97.441	93.737	0.000	**98.422**	76.282	90.341	95.109	93.253	0.000	91.505	35.500	84.154
Results on CIC-2017 Dataset																								
	Accuracy						**Recall**						**Precision**						**F1**					
	XGB	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**
No. Aug	99.94	99.855	70.853	99.799	**98.037**	99.558	98.54	97.492	99.793	92.816	33.89	93.984	99.168	97.104	8.407	99.609	**82.598**	89.989	98.851	97.294	15.509	96.089	48.036	91.936
SMOTE_IPF	99.922	99.862	70.894	98.709	87.734	99.331	99.159	98.698	99.793	**99.073**	98.81	98.243	97.889	96.306	8.421	67.861	17.795	80.906	98.52	97.487	15.529	80.5	30.157	88.73
ProWSyn	99.875	99.767	72.214	98.316	87.329	98.923	**99.314**	**99.175**	99.793	98.782	**99.07**	**98.421**	96.132	92.684	8.785	61.634	17.352	71.825	97.692	95.816	16.148	75.887	29.53	83.04
polynom_fit_SMOTE	99.942	**99.865**	75.396	99.787	89.327	99.547	98.714	97.681	99.52	92.377	94.719	94.052	99.135	**97.326**	**9.782**	99.665	19.417	89.588	**98.922**	**97.501**	**17.813**	95.874	32.228	91.759
GAN(C2ST)	99.942	99.856	73.184	99.79	98.035	99.555	98.629	97.489	**99.898**	92.475	34.275	93.931	**99.186**	97.19	9.075	99.685	81.764	89.954	98.905	97.338	16.643	95.937	**48.285**	91.894
GAN(Botshot)	99.942	99.853	94.511	**99.808**	98.026	99.558	98.645	97.473	0	93.228	34.241	93.984	99.134	97.103	0	99.614	81.305	89.989	98.886	97.287	0	**96.307**	48.171	91.936
CGAN(C2ST)	99.942	99.854	94.547	99.802	97.666	99.558	98.647	97.473	0	92.968	20.238	93.984	99.169	97.137	0	99.68	64.088	89.989	98.853	97.311	0.149	96.009	47.774	91.921
CGAN(Botshot)	**99.942**	99.854	**94.547**	99.802	97.666	**99.558**	98.612	97.421	0	92.968	20.238	93.984	99.185	97.104	0	**99.687**	64.088	**89.989**	98.896	97.262	0	96.196	29.881	**91.936**
Results on CIC-2018 Dataset																								
	Accuracy						**Recall**						**Precision**						**F1**					
	XGB	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**	**XGB**	**DT**	**NB**	**RF**	**LR**	**KNN**
No. Aug	100	**100**	99.99	100	**99.955**	100	99.961	99.885	99	**100**	100	**100**	99.959	99.866	**100**	100	**93.536**	99.921	99.961	99.876	99.5	**100**	**96.658**	**99.96**
SMOTE_IPF	100	100	99.99	100	99.842	100	**99.987**	**99.935**	98.94	99.973	100	100	99.959	**99.974**	100	100	80.64	99.805	99. 973	**99.955**	99.47	99.987	89.28	99.902
ProWSyn	100	100	**100**	100	99.839	100	99.987	99.92	**100**	99.933	100	100	**100**	99.961	99.24	100	80.381	99.678	**99.993**	99.941	**99.62**	99.967	89.121	99.937
polynom_fit_SMOTE	100	100	99.99	100	99.842	100	99.933	99.923	98.61	99.987	100	100	99.973	99.855	100	100	80.597	99.729	99.953	99.89	99.3	99.994	89.252	99.864
GAN(C2ST)	100	99.999	99.34	100	99.944	100	99.884	99.769	0	99.974	100	99.974	99.961	99.856	0	100	92.172	99.922	99.921	99.811	0	99.986	89.926	99.947
GAN(Botshot)	100	99.99	99.34	100	99.944	100	99.884	99.769	0	99.974	100	99.974	99.987	99.856	0	100	92.172	99.922	99.935	99.811	0	99.987	95.926	99.947
CGAN(C2ST)	100	99.998	99.35	100	99.944	100	99.884	99.742	0	99.987	100	99.974	99.935	99.83	0	100	92.172	99.922	99.908	99.785	0	99.993	95.926	99.947
CGAN(Botshot)	**100**	99.998	99.35	**100**	99.944	**100**	99.884	99.742	0	99.987	**100**	99.974	99.935	99.83	0	**100**	92.172	**99.922**	99.908	99.785	0	99.993	95.926	99.947



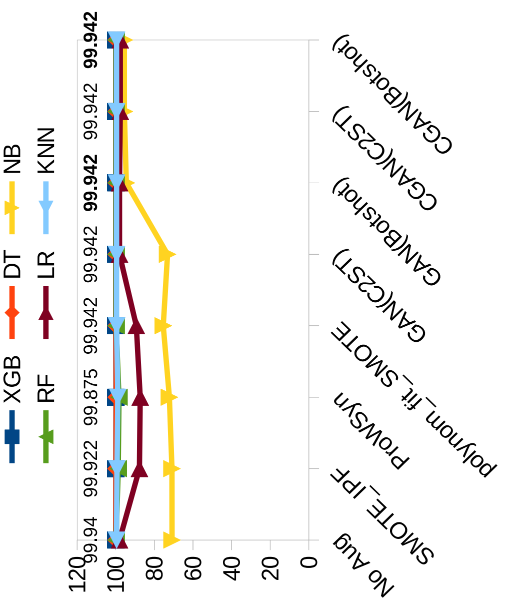
a Accuracy

b Recall

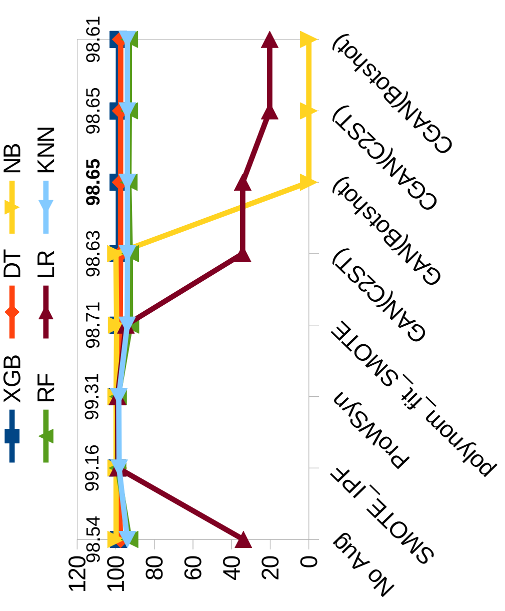
c Precision

d F1

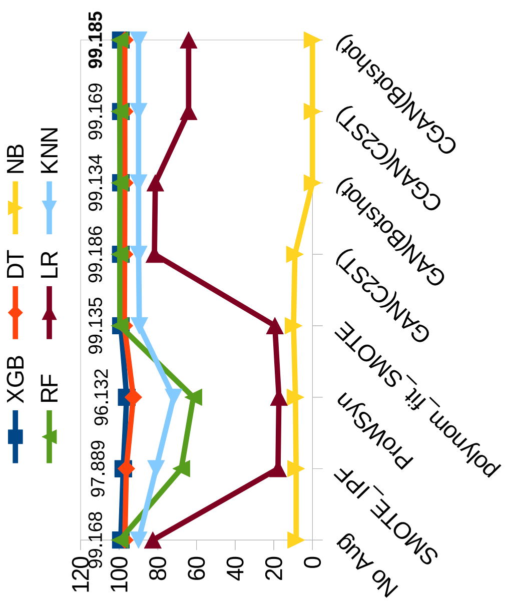
Figure 3.3: Results on ISCX-2014 Dataset



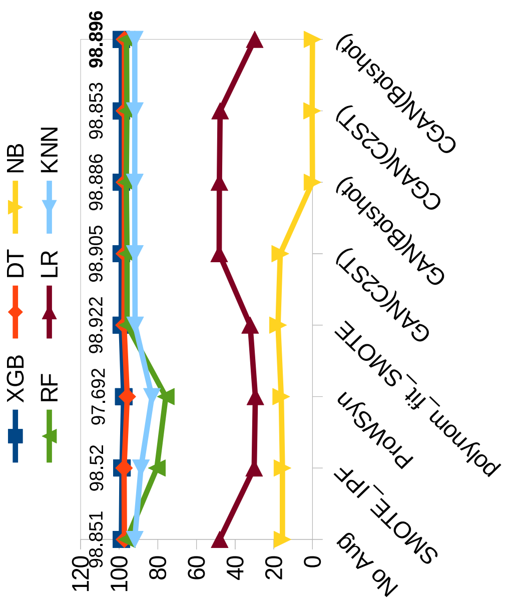
a Accuracy



b Recall

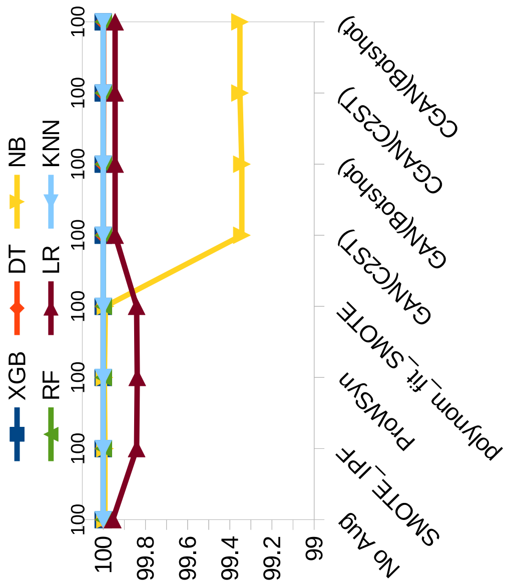


c Precision

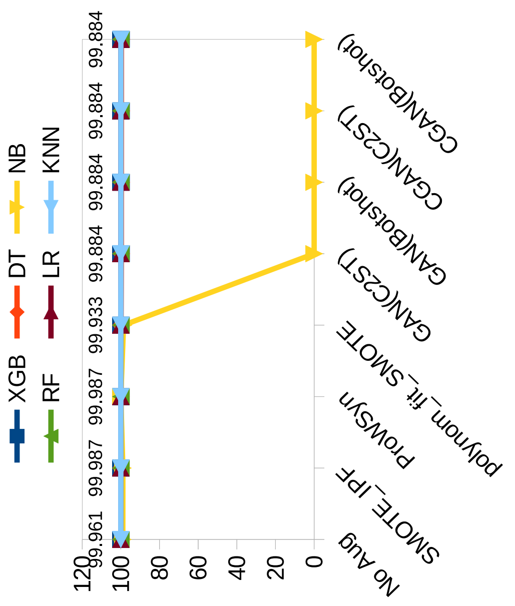


d F1

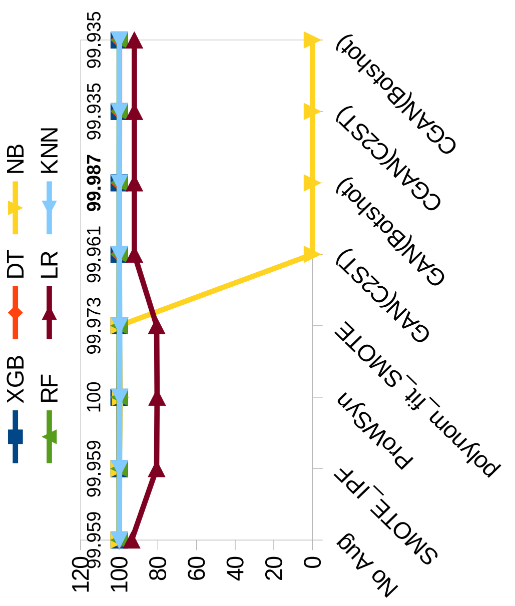
Figure 3.4: Results on CIC-IDS2017 Dataset



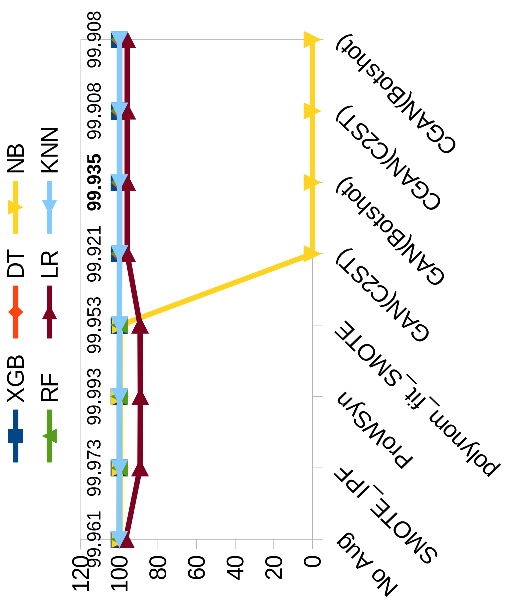
a Accuracy



b Recall



c Precision



d F1

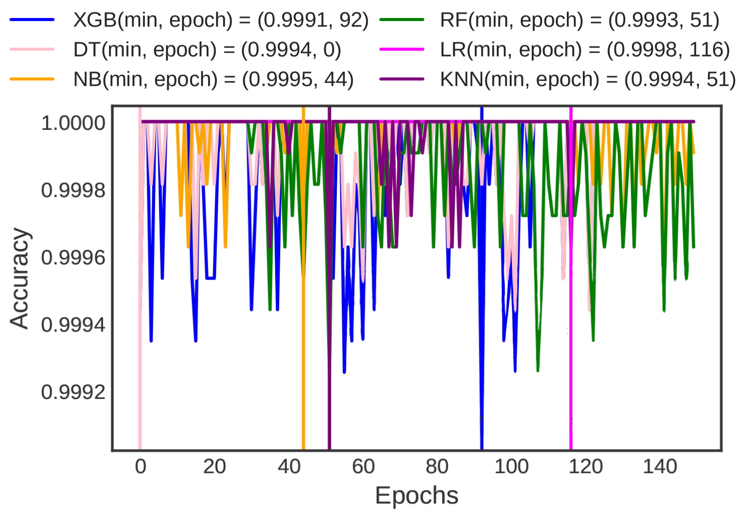
Figure 3.5: Results on CIC-IDS2018 Dataset

3.5 Discussion

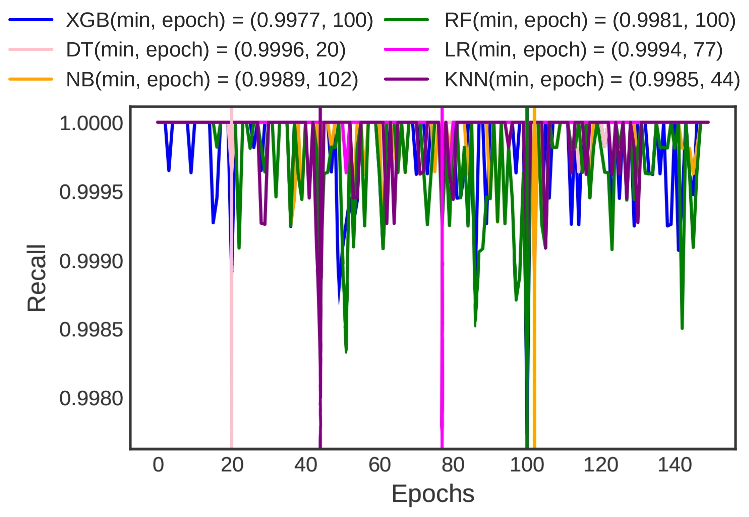
With the help of the extensive results, the effectiveness of the proposed technique can be empirically estimated. Although the existing works show the effectiveness of GANs, one step further is proposed by showing that GANs can be used effectively for the security hardening of botnet detectors against evasion attacks. In light of the results, the discussion can be summarised in subsections as follows:

3.5.1 Relationship of GAN Loss with C2ST and *Botshot*

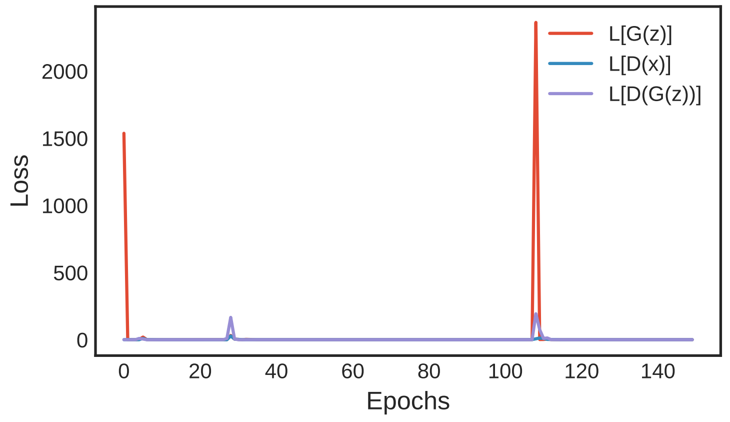
The minimum values of accuracy (for C2ST) and recall (for *Botshot*) for each classifier during the GAN training are depicted in Figures 3.6, 3.7, 3.8, 3.9, 3.10 and 3.11 for all the three datasets. The lines that touch the base (x-axis) represent the minimum values of the accuracy or recall for the particular classifier on a specific epoch. There are two points worth noticing in these diagrams. First, the accuracy or recall may not decrease gradually with each training epoch. The values are stochastic and unpredictable in each epoch for every dataset for both GANs. This result shows that considering only the GAN loss is not a strong metric for evaluating $\mathcal{G}$ performance. Although, as mentioned in Subsection 3.3.7, the losses converge and remain close to zero, the adopted evaluation, using C2ST and *Botshot*, behaves differently. That said, a GAN may not generate realistic samples even it has converged to a Nash equilibrium. Hence, some other GAN evaluation techniques should be used as C2ST or *Botshot* [53, 75]. Second, there will not always be an epoch that will have both accuracy and recall minimum at the same time. In fact in some cases the epochs for both these metrics are different as can be seen in case of XGB, DT and RF in Figure 3.6, DT in Figure 3.7, KNN in Figure 3.8, XGB and DT in Figure 3.9 and XGB in Figure 3.11. Moreover, in only one case where accuracy and recall were both minima in the same epoch for all the classifiers in Figure 3.10 reinforces the notion that recall could be a more effective metric for assessing the similarity between the generated and real samples.



a C2ST

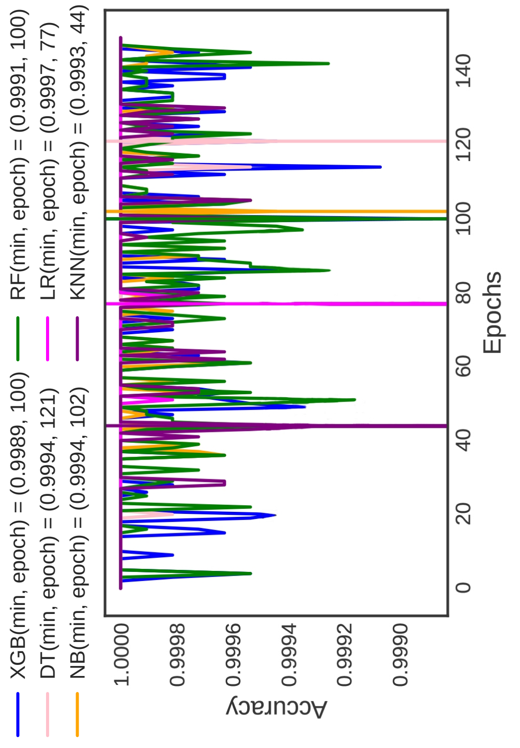


b Botshot

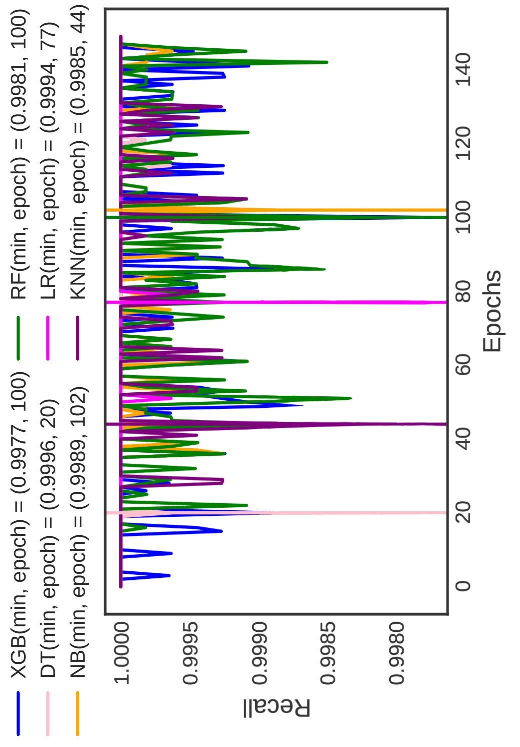


c Losses

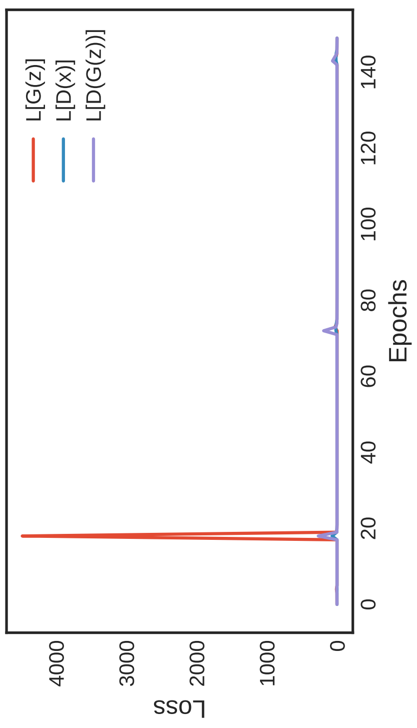
Figure 3.6: GAN Training & Evaluation on ISCX-2104 Dataset



a C2ST

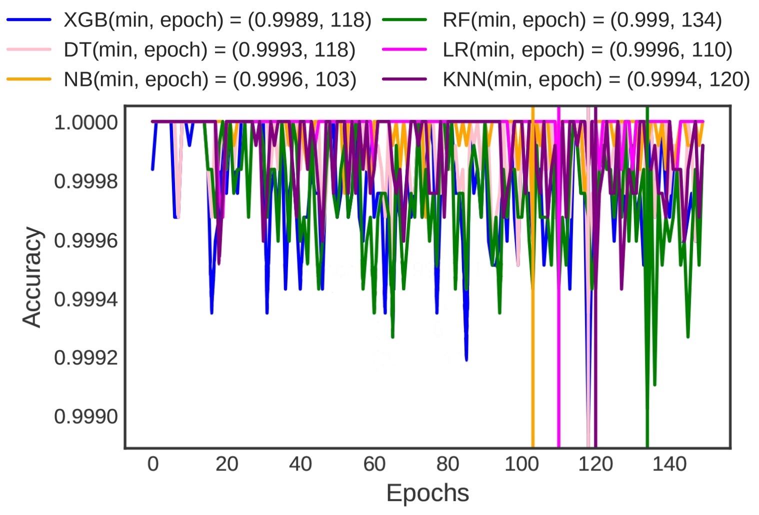


b Botshot

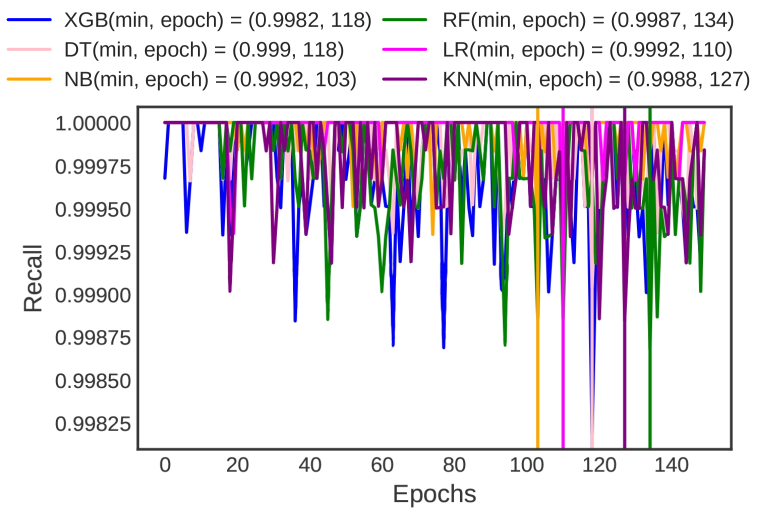


c Losses

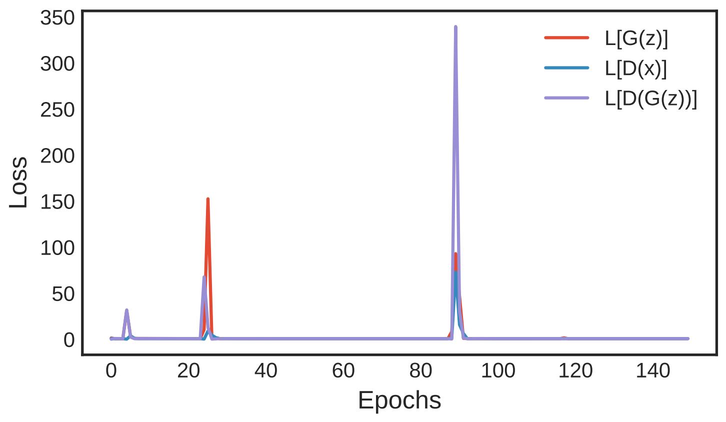
Figure 3.7: CGAN Training & Evaluation on ISCX-2104 Dataset



a C2ST

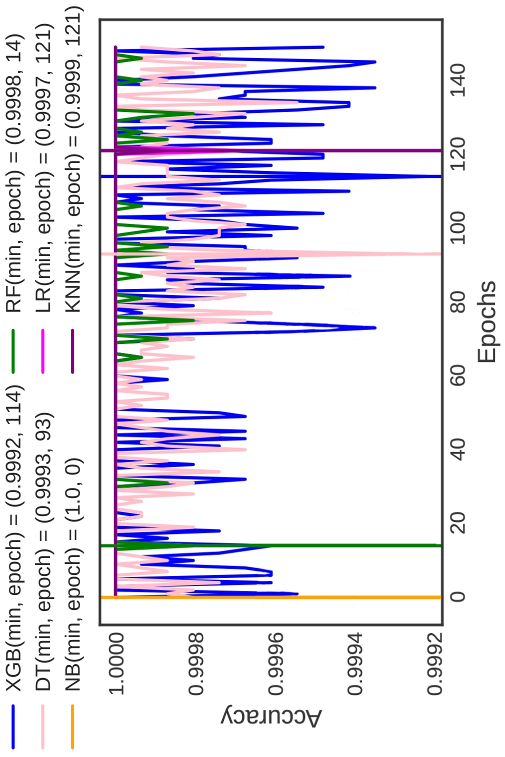


b Botshot

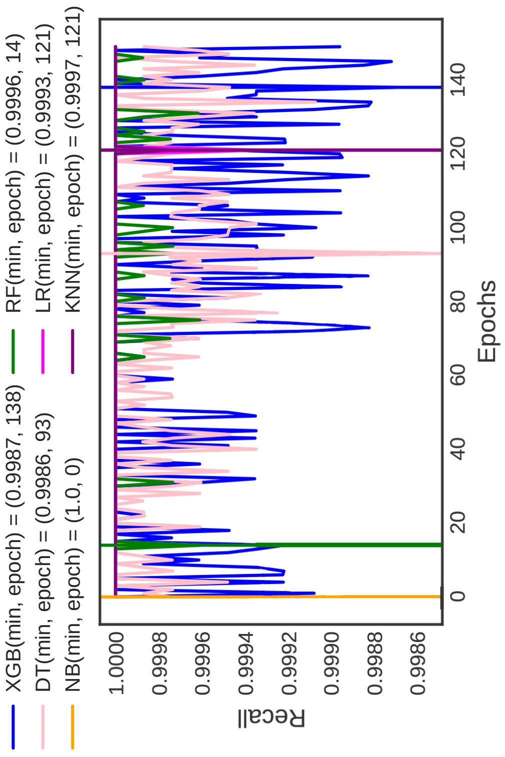


c Losses

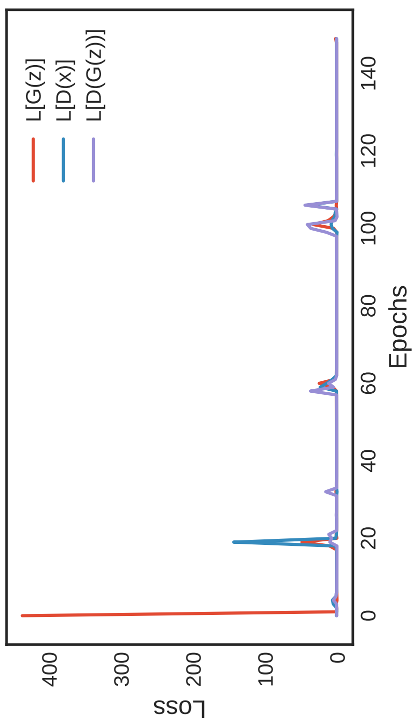
Figure 3.8: GAN Training & Evaluation on CIC-IDS2017 Dataset



a C2ST

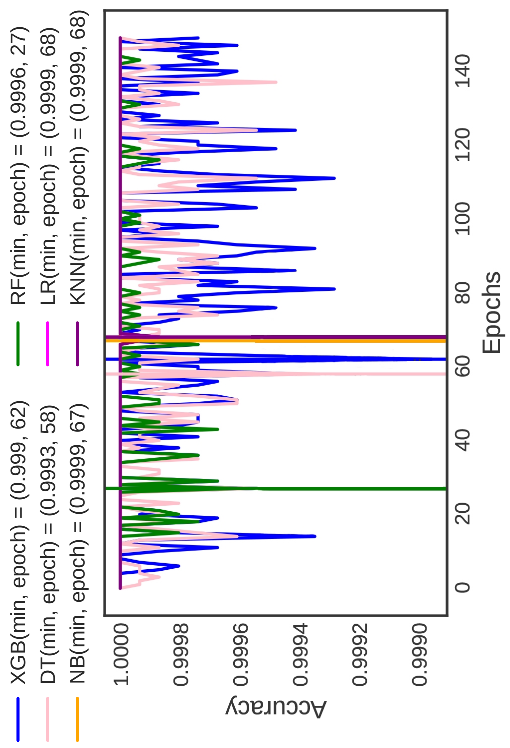


b Botshot

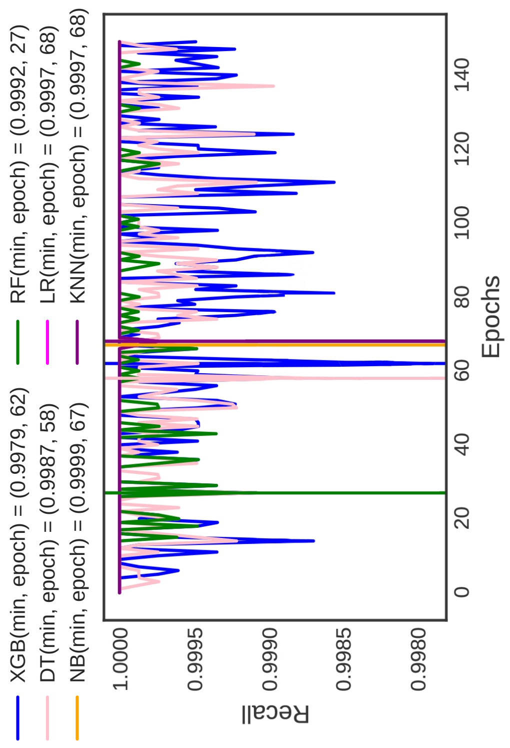


c Losses

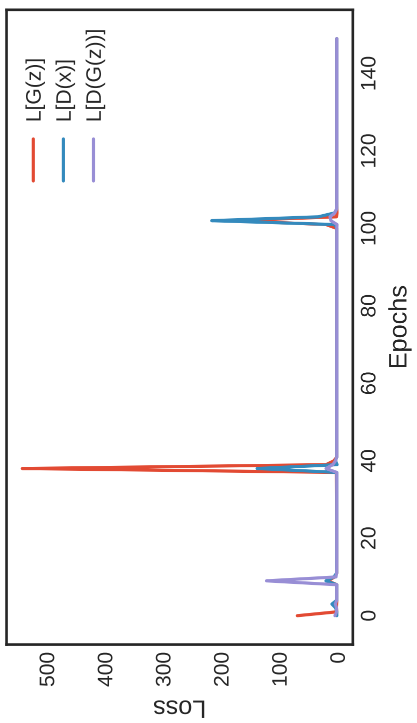
Figure 3.9: CGAN Training & Evaluation on CIC-IDS2017 Dataset



a C2ST

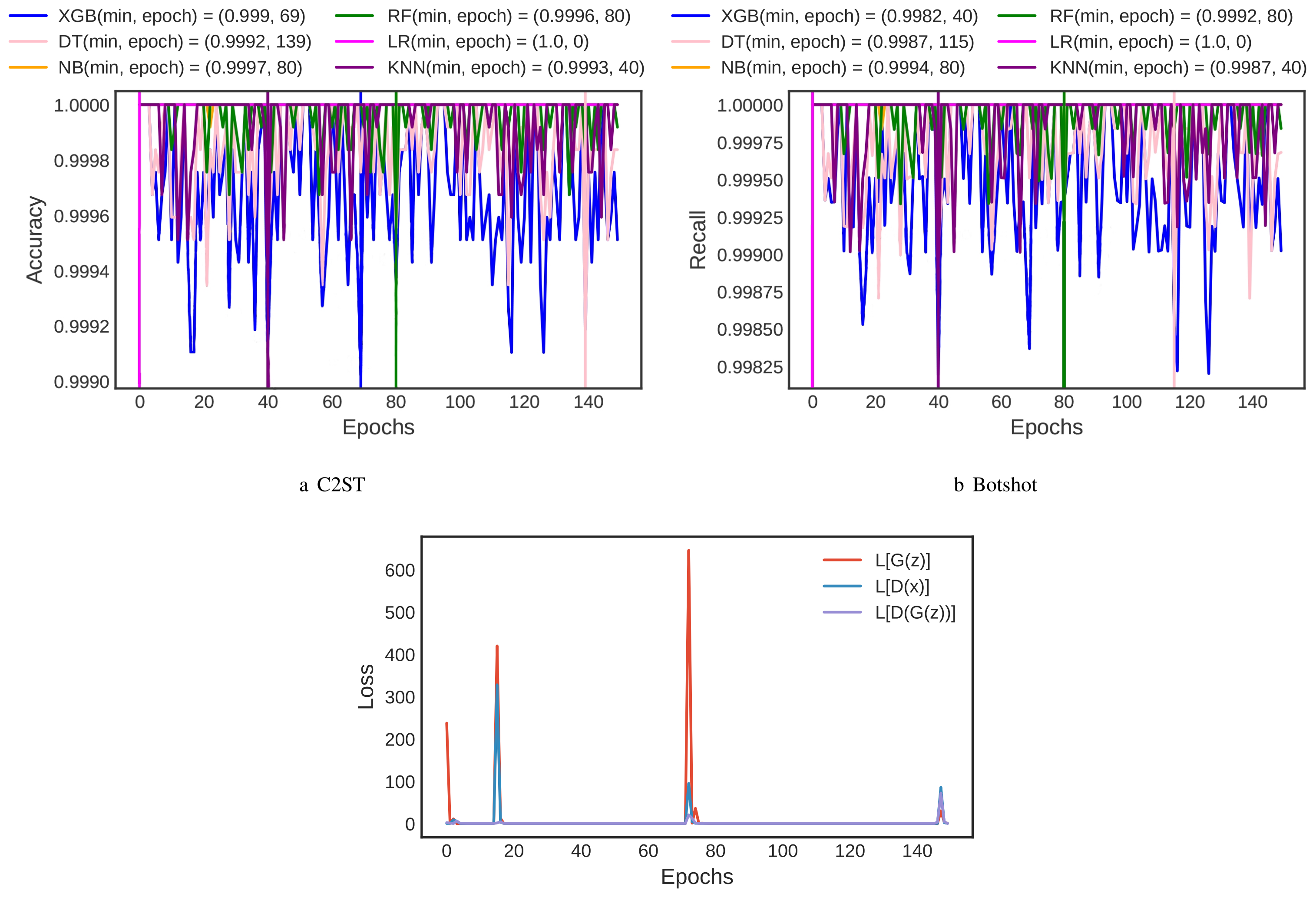


b Botshot



c Losses

Figure 3.10: GAN Training & Evaluation on CIC-IDS2018 Dataset



a C2ST

b Botshot

c Losses

Figure 3.11: CGAN Training & Evaluation on CIC-IDS2018 Dataset

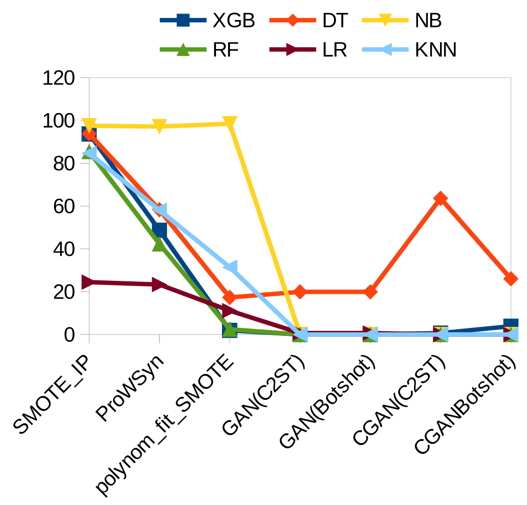
3.5.2 Comparison of *Botshot* with Peer Techniques/Works

3.5.2.1 Oversampling

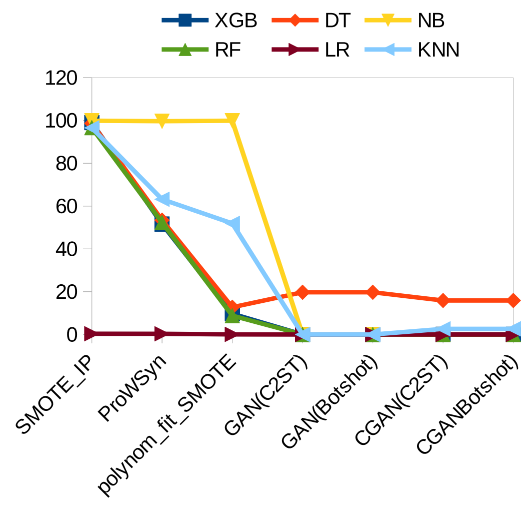
While comparing GAN based oversampling with SMOTEs, a question may arise whether it is better than its interpolation-based counterpart. The answer can be a definite 'yes' if a GAN model is explored with appropriate hyperparameters that can generate more realistic samples than those proposed in the current work. The accuracy and recall both have values during GAN evaluation no less than 99% (ideal case is 50%), which means that the GANs generate few realistic samples of botnet traffic. However, a significant improvement in the final results can be observed in Table 3.4. Since the SMOTE oversamplers is better in many cases; it can not be said 'yes' to the question. For that reason, the research of exploring a suitable GAN for this purpose is left to future work. For example, the ISCX-2014 dataset could not give us good results in recall for the *Botshot*, so one can start with a set of an appropriate classifier, a GAN and this dataset to improve the detection performance by making necessary exploration of a suitable GAN model.

3.5.2.2 Blackbox Test and Improvements in Recall

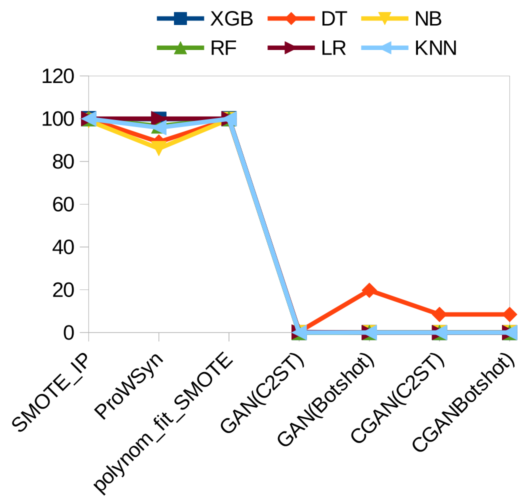
The black-box test could evaluate the aptness of the generated botnet data to be fooled as benign by the classifiers. The trained classifiers were tested with generated samples added in the test set in each 70-30 split. The results for the three datasets for recall are illustrated in Figure 3.12. The results are significantly remarkable if the GAN-based oversampling is compared with the other synthetic techniques for black-box testing. The recall score drops excessively for GAN-based techniques inferring that the GAN-generated samples can evade the classifiers more than peer oversampling techniques.



a ISCX-2014

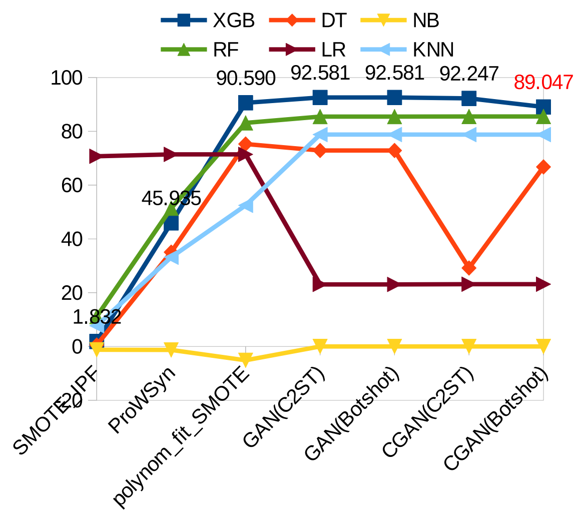


b CIC-IDS2017

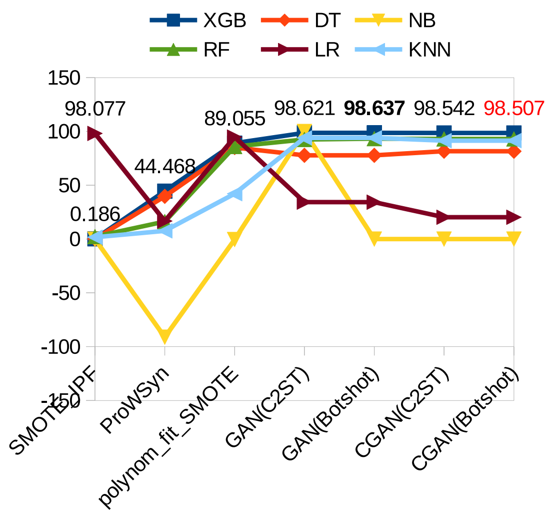


c CIC-IDS2018

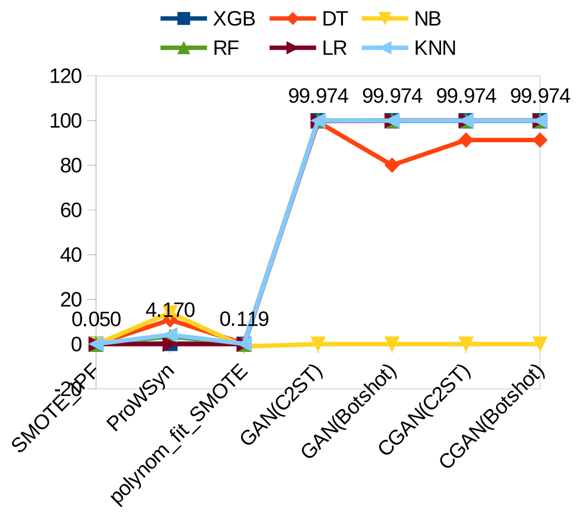
Figure 3.12: Recall score (y-axis) after black box test for each oversampling technique for all the three datasets



a ISCX-2014



b CIC-IDS2017



c CIC-IDS2018

Figure 3.13: Improvement in recall score (y-axis) after adversarial training for all the three datasets

Figure 3.13 shows the improvement in the value of recall after adversarial training of generated data for each oversampling technique. In the case of the best performing classifier (XGB), the improvement is 98.58%, 98.64% and 99.97% for *Botshot* being the highest among the peer techniques. The improvement in recall as compared to the C2ST was lower for ISCX-2014 and CIC-IDS2017 (values marked in red in Figure 3.13) for CGAN(Botshot), which opens further research horizons towards exploring more GANs to validate *Botshot*. It is left as future work.

The works like [15, 19] are the closest for comparing to BotShot Although, [15] has used the ISCX dataset used in this thesis but they did not employ the enhanced feature set and there was no comparison made with the synthetic oversamplers. [19] are using the KDD99 dataset that is different to used in this thesis although they have used GANs as the oversamplers to improve the detection performance of the ML classifiers.

3.5.3 Limitation of *Botshot*

In Figure 3.14, the GAN mimicry of the two distinct features from the ISCX-2014 dataset for all six classifiers has been demonstrated in the form of contour plots. The graphs show that the GAN Bots generated by both C2ST and *Botshot* are trying to follow the distribution of the Real Bots for the two features 'Fwd Segment Size Avg' and 'Bwd Segment Size Avg'. These two features were chosen based on the fact that in most of the training iterations, their distribution is unchanged. The generator generates these two features after a certain number of initial training iterations. The same principle is valid for the CIC-2017 and CIC-2018 datasets, so their results were not included. However, it is imperative to know if the generated samples are as valid as real-life traffic samples since not all the network traffic samples generated by the GANs might be valid. The encoding/decoding of generated attack samples to relay to the internet to validate the semantics was not performed . Since this work aims to devise a defence strategy against adversarial attacks, improving the detection performance with the addition of data generated by GAN in the ML domain is adequate. Hence, the realistic samples were generated to improve detection rather than real traffic samples to fail botnet detectors. This is left as future work to filter valid samples out of the generated data from GANs.

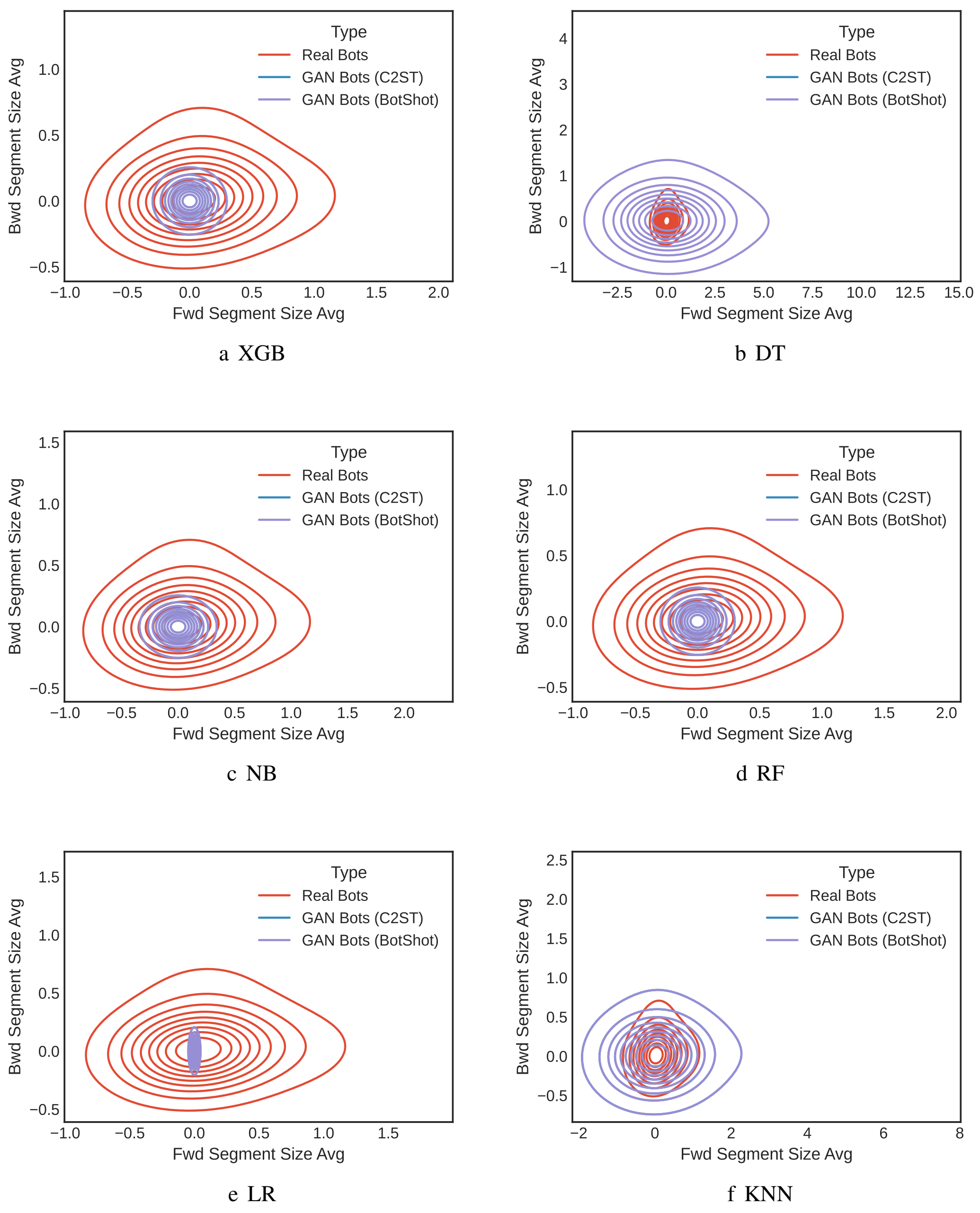


Figure 3.14: Contour plot of two random features from the ISCX-2014 dataset: Both GAN Bots (C2ST) and GAN Bots (*Botshot*) try to follow the Real Bots distribution.

3.6 Summary

GANs have proved to be very effective in computer vision-based applications. However, it can be inferred from this work that GANs are suitable candidates to address multiple challenges within the cybersecurity domain as well. Significantly, the effects of adversarial evasion attacks can be mitigated proactively using GANs' generated traffic augmentation to the original train sets. The general method to utilise GANs to generate quality samples is based on the loss functions of generator and discriminator networks. However, the quality of generated data could be further evaluated using the classifiers under test.

- A technique was proposed to generate improved quality samples to fulfil this objective in the case of botnet detection.
- The results show that GANs can provide a better alternative to the traditional traffic generation methods for all the classifiers used. GANs can also balance the datasets, further enhance the botnet detectors' security hardening, especially against adversarial evasion attacks, and decrease false positives.

The behaviours and landscapes of modern botnets need to be explored further with the introduction of novel traffic features to differentiate botnets from benign traffic. Modern GANs could be harnessed to enhance the quality of adversarial examples further. The validity of GANs' generated traffic in terms of semantics could be another extension of this research work. A further envisaged research direction could be making the IDS autonomous to be proactively trained against novel evasion samples using deep reinforcement learning.

Chapter 4

Evasion Generative Adversarial Network for Low Data Regimes

4.1 Introduction

Many recent literary works have leveraged generative adversarial networks (GANs) to spawn unseen evasion samples. The purpose is to annex the generated data with the original train set for adversarial training to improve the detection performance of machine learning (ML) classifiers. The quality of generated adversarial samples relies on the adequacy of training data samples. However, in low data regimes like medical diagnostic imaging and cybersecurity, the anomaly samples are scarce in number. This chapter proposes a novel GAN design called Evasion Generative Adversarial Network (EVAGAN) that is more suitable for low data regime problems that use oversampling for the detection improvement of ML classifiers. EVAGAN not only can generate evasion samples, but its discriminator can act as an evasion-aware classifier. The Auxiliary Classifier GAN (ACGAN) was considered as a benchmark to evaluate the performance of EVAGAN on cybersecurity (ISCX-2014, CIC-2017 and CIC2018) botnet and computer vision (MNIST) datasets. It is demonstrated that EVAGAN outperforms ACGAN for unbalanced datasets concerning detection performance, training stability and time complexity. EVAGAN's generator quickly learns to generate the low sample class and hardens its discriminator simultaneously. In contrast to ML classifiers that require security hardening after being adversarially trained by GAN-generated data, EVAGAN renders it needless. The experimental analysis proves that EVAGAN is an efficient evasion-hardened model for low data regimes for the selected cybersecurity and computer vision datasets. The implementation of EVAGAN in the form of python code is available at https: //github.com/rhr407/EVAGAN.

Low data regimes are found in many real-life applications where researchers face data scarcity problems [76]. The data scarcity pertains to the situation where one class is abundant in data samples (especially normal behaviour) while the anomaly samples are rare and challenging to gather [77]. The data scarcity can also be described as a data imbalance problem potentially resulting in decision bias in machine learning (ML) classifiers. The network traffic datasets are one of the prime examples of data imbalance problems. Since the ML intrusion detection systems are data-hungry probabilistic models, having more data can improve their performance [65]. The real attacks can be emulated with dedicated machines in a lab environment using open-source operating systems like Kali Linux [38, 42]. However, there can be two main disadvantages of emulating real attacks: First, real data gathering can be expensive, involving multiple hardware resources like multiple computers and network switches [78]. Second, the emulated attacks may not accurately represent a real attack scenario. A cost-effective way of gathering the attacks' data is synthetic generation using AI generative models [53].

As discussed in the previous chapter, generative adversarial networks (GANs) as synthetic oversamplers have been a voguish research endeavour for low data regimes [65, 79]. Various researchers have demonstrated that GANs are more effective than other synthetic oversamplers like SMOTE [11, 14, 53, 77]. It is found in numerous studies that due to the adversarial factor; GANs can better estimate the target probability distribution [14, 77, 80]. In a simple/vanilla GAN, two different neural networks(generator ($\mathcal{G}$) and discriminator ($\mathcal{D}$)), work antagonistically to learn from each other's experience to converge to Nash equilibrium [22]. As an oversampler, after being trained to a certain number of epochs, $\mathcal{G}$ is used to generate additional data. Depending on how well a GAN learned the input data probability distribution, the close resembling data is annexed to the original train set. This process is called data augmentation (DA), which many researchers have demonstrated to be effective in improving the detection performance of ML classifiers [19, 81–84].

Since AI-based systems are prone to adversarial evasion attacks, it is imperative to harden the ML classifiers against adversarial evasions. Black box attackers can use GANs to generate evasion samples [19, 83, 85]. Therefore, employing GANs can be an effective technique for proactively designing an evasion-aware classifier resulting from DA. Although DA is effective in helping the ML classifiers recognise the perturbed data samples, $\mathcal{D}$ of a GAN can be extended to act as a multiclass classifier so that it can be used as an anomaly detector [15, 24, 83]. In this way, there is no need to use DA as $\mathcal{D}$ is trained simultaneously with $\mathcal{G}$. ACGAN is an example of such a GAN in which $\mathcal{D}$ not only differentiates between fake and real samples but also can be used as a multiclass classifier [54, 77]. The advantage of extending $\mathcal{D}$ in ACGAN is to improve training stability and quality of generated samples [54]. In this work, with the help of experimentation, it has been demonstrated that ACGAN does not perform well in highly unbalanced datasets. So a novel GAN based on ACGAN called EVAGAN is proposed that outperforms ACGAN in terms of detection performance, stability in training and time complexity.

The main contributions of this chapter are summarised in the following aspects:

1. A novel GAN model is proposed to design an evasion-aware discriminator as a sophisticated botnet detector.
2. The experiments demonstrate that the existing use of ACGAN to design a sophisticated classifier can fail in highly unbalanced datasets.
3. It was determined that the EVAGAN outperforms ACGAN in terms of performance detection, stability and time complexity for cybersecurity (CC) botnet and computer vision

Table 4.1: Main notations

Notation	Definition
$\mathcal{G}$	Generator
$\mathcal{D}$	Discriminator
z	Normal distribution from noise space
z	Noise samples
p_{data}	Probability distribution of real samples
p_z	Probability distribution of noise samples
$\mathcal{X}$	Real data distribution
$\mathbb{E}$	Expected value
c_m	Minority class
c_M	Majority class
y_{x_i}	Label of a sample x_i in $\mathcal{X}$ dataset
$y_{x_i}^{real}$	Label of a real sample
$y_{x_i}^{fake}$	Label of a fake sample
$y_{x_i}^{c_M}$	Label of a majority class sample
$y_{x_i}^{c_m}$	Label of a minority class sample
$\mathcal{X}_{M_{real}}$	Set of real majority class samples
$\mathcal{X}_{m_{fake}}$	Set of fake/generated samples
$\mathcal{X}_{m_{real}}$	Set of real minority class samples
$y_{x_i}^{c_{m_{real}}}$	Label of real minority class samples

(CV) datasets.

4.2 EVAGAN

In this section, the motivation behind the design of EVAGAN is discussed, the structural explanation of its generator and discriminator, and the objective and loss functions. Table 4.1 shows the main notations used in this chapter.

4.2.1 Motivation

Considering the generator ($\mathcal{G}$) of ACGAN, $\mathcal{X}_{fake} = \mathcal{G}(c, z)$ where c is the class label, $\mathcal{G}$ has to generate the samples of all classes. Hence the number of the samples generated by $\mathcal{G}$ may include $C = \{c_1, c_2, c_3, ..., c_n\}$ which may not be a requirement in low data regimes. Since there is only need to generate a low sample class with labels c_m instead of all the classes, the generator does not need to be aware of the classification performance of $\mathcal{D}$ on majority class samples. In this way, the training time of $\mathcal{G}$ is reduced as the diversity seen by $\mathcal{G}$ is less complex to generate a single class sample. Due to this reason, the performance of $\mathcal{G}$ can be improved and the $\mathcal{D}$ can be hardened simultaneously with fewer c_m samples.

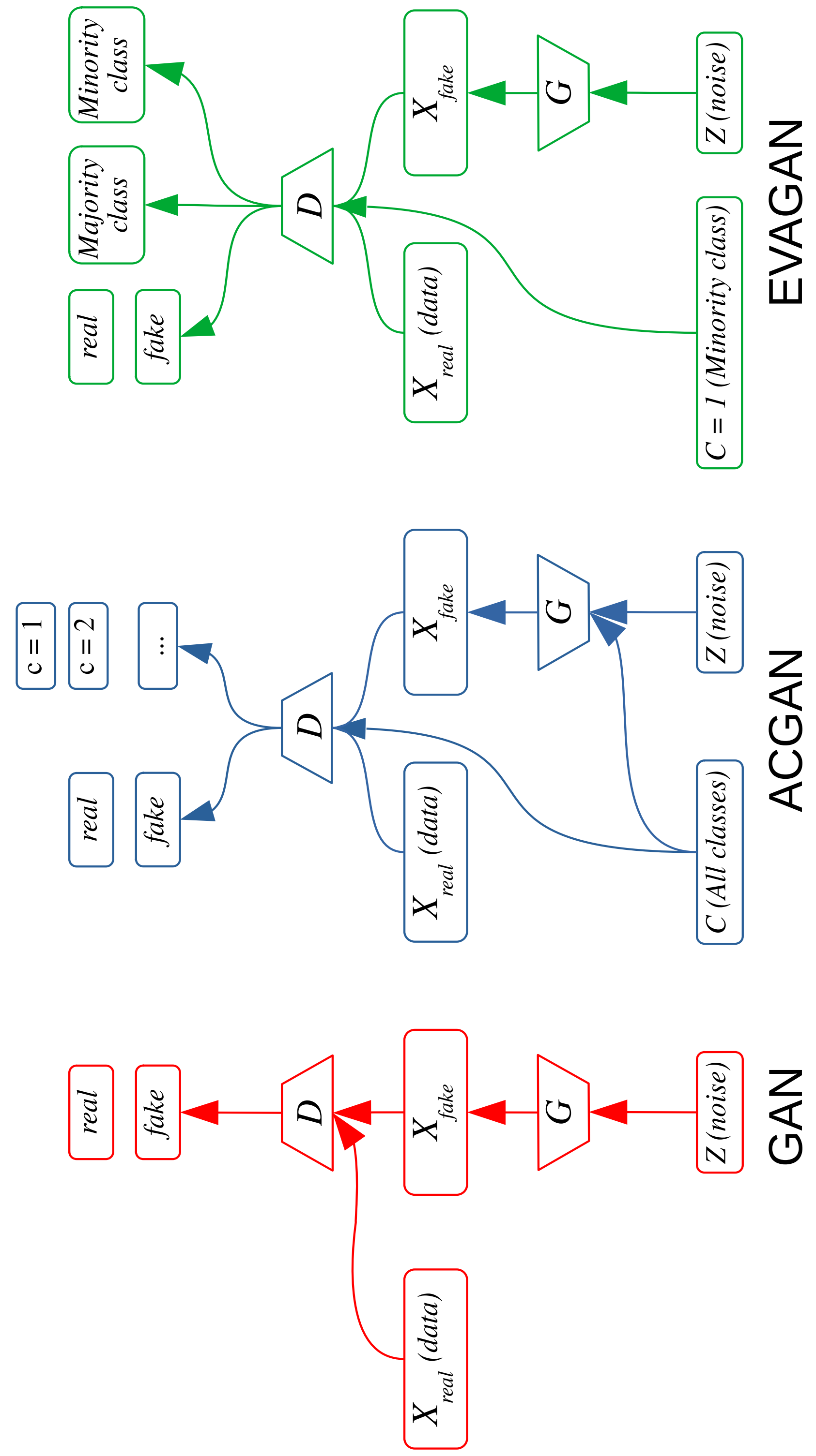


Figure 4.1: Comparison of EVAGAN model with vanilla GAN and ACGAN

The ratio of the different class labels can vary the performance of $\mathcal{G}$ as this is a stochastic process. However, in most cases, the normal class samples will be more than the anomaly samples. Note that EVAGAN design is dedicated to binary class problems where the samples of a minority class are scanty. For using EVAGAN for multiclass cases, each anomaly class should be considered separately from the normal class to make it a binary classification problem. However, the concept can be extended to multiclass, which is left to future work.

Figure 4.1 shows the structural difference between ACGAN and EVAGAN. Considering the challenges mentioned above, the design of ACGAN based on a few simple modifications is extended. The significance of the modifications in the generator input, discriminator output and loss functions for EVAGAN is discussed in more detail in section 4.2.

4.2.2 Architecture

The design of EVAGAN is inspired by ACGAN as the desire is to develop a classifier model that hardens itself on the GAN-generated evasion samples. The main structure of EVAGAN consists of two neural networks; the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$. In contrast to ACGAN, EVAGAN's model is limited to labels from a single class embedded with noise as the input to the generator ($\mathcal{G}$). The details of $\mathcal{G}$ have been explained in subsection 4.2.3. Figure 4.2 shows the detailed architecture of EVAGAN. There are three types of colour-graded arrows shown in the figure. The red coloured arrows demonstrate the first training step in which only $\mathcal{D}$ is trained. The green arrows depict the second training step for $\mathcal{G}$. The orange arrows show the involvement of common inputs (minority class labels and noise) and outputs (real/fake and minority class estimations) for both training steps mentioned previously. These two steps of a typical GAN training were expressed earlier in subsection 2.3. The discriminator $\mathcal{D}$ of EVAGAN has three different outputs for the estimation of majority, minority and fake/real classes. Sigmoid functions have been used for the three outputs, each with binary cross-entropy (BCE) loss. The details of $\mathcal{D}$ are further expressed in subsection 4.2.4. The loss functions have also been mentioned in respective subsections of $\mathcal{G}$ and $\mathcal{D}$.

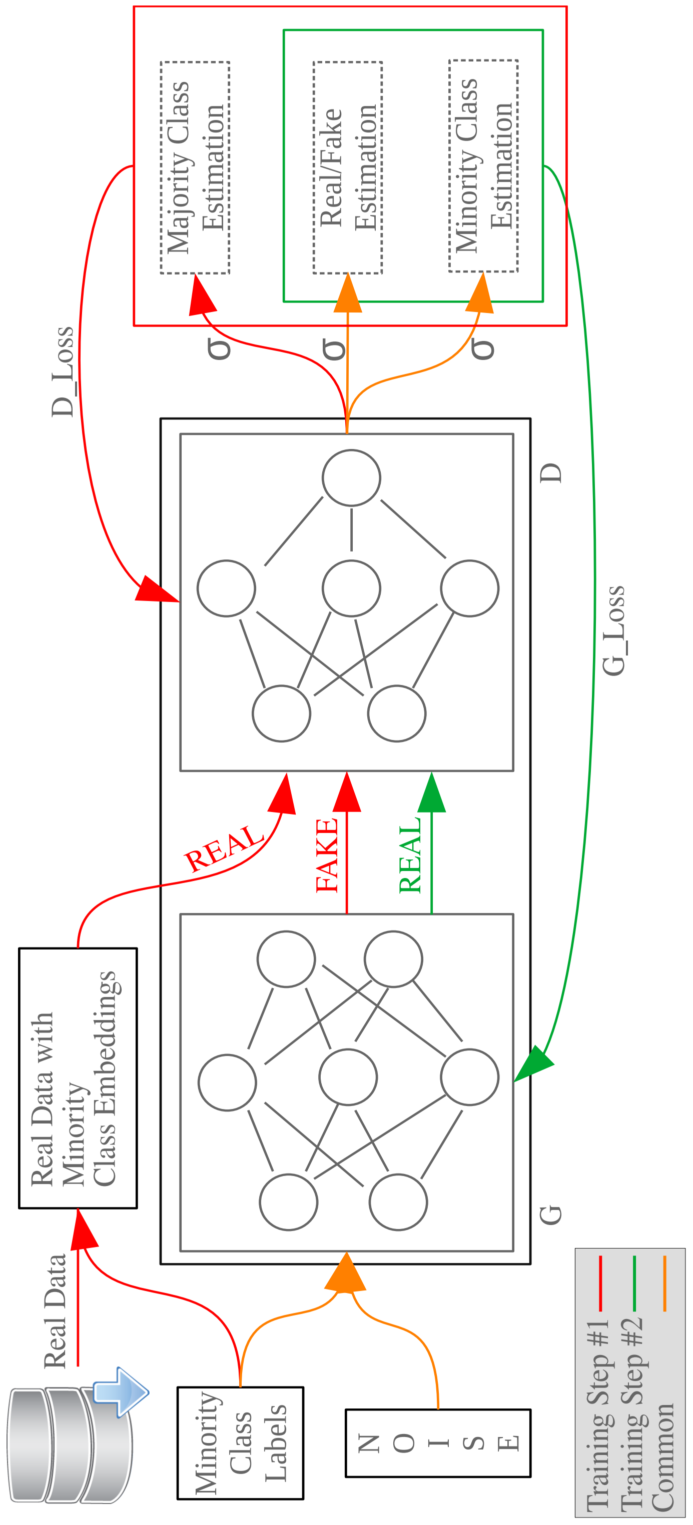


Figure 4.2: EVAGAN Architecture

Figure 4.2 shows a red outlined box on the right side to illustrate the three different probability estimations as outputs from $\mathcal{D}$. These three estimations are used to compute the loss of $\mathcal{D}$ in the first step of EVAGAN training. A green outlined box including the real/fake estimation and minority class estimation computes the G_Loss to be fed back to $\mathcal{G}$ in the back-propagation of the combined model training (second step of EVAGAN training). Note that the output of $\mathcal{D}$ is distributed using three different sigmoid units to separate the probabilities of each class, i.e. majority, sources (real/fake) and minority. The majority and minority class estimations could be combined using a single sigmoid function. However, keeping them separate has three advantages. The first is to avoid the loss of the majority class being fed back to $\mathcal{G}$. Second, it simplifies the model with no extra training cost. Third, the predictions for the test set samples can be conveniently separated, which will be discussed in Section 4.4.

4.2.3 Generator

The generator ($\mathcal{G}$) of EVAGAN only takes noise z and the single class labels $c_m = 1$ as the only binary classification (normal and anomaly samples) is considered. The labels are embedded in the input layer of $\mathcal{G}$. The objective function of $\mathcal{G}$ has two parts as shown in Equations 4.1 and 4.2.

$$I^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}[\log(\mathcal{D}(\mathcal{G}(z)))] \tag{4.1}$$

$$J^{\mathcal{G}}(\mathcal{G}) = \mathbb{E}[\log P(C = c_m | \mathcal{X}_{m_{fake}})] \tag{4.2}$$

Equation 4.1 is the objective function of $\mathcal{G}$ similar to Equation 2.2. The goal is to minimise the log-likelihood of the fake samples being classified as fake by $\mathcal{D}$. In Equation 4.2, $J^{\mathcal{G}}(\mathcal{G})$ is the objective function of $\mathcal{G}$ for improving the log-likelihood of minority class samples coming from $\mathcal{G}$ into $\mathcal{D}$, where P is the output probability from $\mathcal{D}$. Since $\mathcal{G}$ only needs to generate c_m samples so it should only receive the loss of $\mathcal{D}$ on the estimation of minority class and the sources, i.e. the samples being real or fake. The objective function of $\mathcal{G}$ is to maximise $\mathcal{D}$ loss on the fake source. At the same time, it will assist in minimising $\mathcal{D}$ loss on c_m samples. Equation 4.3 shows the objective function of $\mathcal{G}$.

$$L^{\mathcal{G}}(\mathcal{G}) = J^{\mathcal{G}}(\mathcal{G}) - I^{\mathcal{G}}(\mathcal{G}) \tag{4.3}$$

The cross-entropy (CE) loss of real and predicted probability distributions $p(x)$ and $q(x)$ respectively, can be denoted using Equation 4.4, where x denotes the samples belonging to $\mathcal{X}$ dataset.

$$CE(p,q) = -\sum_{x \epsilon X} p(x) \log q(x) \tag{4.4}$$

Let y_{x_i} be the actual label of sample x_i in dataset $\mathcal{X}$, $P(S = fake|\mathcal{X}_{m_{fake}})$ be the predicted probability distribution of generated samples being fake and $P(C = c_m|\mathcal{X}_{m_{fake}})$ be the predicted probability distribution for minority class c_m, both coming from $\mathcal{D}$, then the loss function of $\mathcal{G}$ for N samples will be given by the Equation 4.5.

$$\begin{aligned} G\_Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_{x_i}^{fake}(\log P(S = fake|\mathcal{X}_{m_{fake}}))+ \\ y_{x_i}^{c_m}(1 - \log P(C = c_m|\mathcal{X}_{m_{fake}}))] \end{aligned} \tag{4.5}$$

In Equation 4.5, $y_{x_i}^{fake}$ and $y_{x_i}^{c_m}$ are the actual labels for fake and minority classes respectively. According to Equation 4.5, the goal of $\mathcal{G}$ is to minimize the G_Loss, so it tends to reduce the correct estimation of $\mathcal{D}$ on fake samples by suppressing the term $\log P(S = fake|\mathcal{X}_{m_{fake}})$. For the second objective, it will try to increase the value of $\log P(C = c_m|\mathcal{X}_{m_{fake}})$ so that the second term in the equation can also be suppressed in value.

4.2.4 Discriminator

For the $\mathcal{D}$ model of EVAGAN, the majority and minority class estimations were separated using two different sigmoid (σ) functions as demonstrated in Figure 4.2. The benefit of separating both estimations is that only minority class estimation can be fed back to $\mathcal{G}$. The other advantage of this structure is that the estimation of both classes could be separately calculated on test datasets to compare it with the ACGAN model later done in section 4.4. The objective function of $\mathcal{D}$ has three parts as given by the Equations 4.6, 4.7 and 4.8. For the minority class terminologies, 'm', and for the majority class, 'M' was used in the following equations.

$$L_M = \mathbb{E}[\log P(C = c_M | X_{M_{real}})] \tag{4.6}$$

$$\begin{aligned} L_{S_m} &= \mathbb{E}[logP(S = real | X_{m_{real}})] + \\ &\mathbb{E}_{[}logP(S = fake | X_{m_{fake}})] \end{aligned} \tag{4.7}$$

$$\begin{aligned} L_m &= \mathbb{E}[\log P(C = c_m | X_{m_{real}})] + \\ &\mathbb{E}[\log P(C = c_m | X_{m_{fake}})] \end{aligned} \tag{4.8}$$

The first goal of $\mathcal{D}$ is to correctly estimate the majority class distribution from the real samples only as $\mathcal{G}$ does not generate the majority class samples. Equation 4.6 denotes the log-likelihood for the real majority class samples. Equation 4.7 represents the source log-likelihood for the real and fake minority class samples. Equation 4.8 summarises the real and fake log-likelihoods from $\mathcal{D}$ for minority class samples. Hence, the objective function of $\mathcal{D}$ can be represented as the sum of the three log-likelihoods to be maximised by $\mathcal{D}$ as given by Equation 4.9

$$L^{\mathcal{D}}(\mathcal{D}) = L_M + L_{S_m} + L_m \tag{4.9}$$

The loss function of $\mathcal{D}$ can be derived using Equation 4.1 and 4.2, given by Equation 4.10.

$$
\begin{aligned}
D\_Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_{x_i}^{c_M}(\log P(S = c_M|\mathcal{X}_{M_{real}}))+ \\
y_{x_i}^{real}(\log P(S = real|\mathcal{X}_{m_{real}}))+ \\
(1 - y_{x_i}^{real})(1 - \log P(S = real|\mathcal{X}_{m_{real}}))+ \\
y_{x_i}^{c_{m_{real}}}(\log P(C = c_m|\mathcal{X}_{m_{real}}))+ \\
(1 - y_{x_i}^{c_{m_{real}}})(1 - \log P(C = c_m|\mathcal{X}_{m_{real}}))]
\end{aligned}
\tag{4.10}
$$

In Equation 4.10, the loss of $\mathcal{D}$ has been derived from three different binary cross-entropy losses for majority class, sources and minority class estimations. Note that the loss on c_M was ignored for being fake because no majority class samples are being generated by $\mathcal{G}$.

4.3 Implementation Details

4.3.1 Experimental Setup

The experiments were performed on a GPU workstation, AMD Ryzen threadripper 1950x with a 16-core processor and GeForce GTC 1070 Ti (8GB) graphics card, running ubuntu 20.04. Keras, TensorFlow, Sklearn and Numpy libraries were used in the Jupyter notebook and visual studio code (VSCode). The source code of EVAGAN has been provided on GitHub under MIT license[1].

4.3.2 Data Preparation

For experimentation, CC botnet and CV MNIST datasets were used. The quantitative analysis of EVAGAN was performed on CC datasets. ISCX-2014, CIC-2017 and CIC-2018 CC datasets were employed as previously used in Chapter 3. The reader may refer to the previous chapter for

[1]https: //github.com/rhr407/EVAGAN

more details on the feature set selections as given in 3.2 and the benign vs botnet samples distribution mentioned in Table 3.1. The qualitative analysis was performed using visual inspection. For this purpose, the MNIST handwriting digits dataset was used.

4.3.3 CV Dataset

4.3.3.1 MNIST Dataset

The MNIST dataset is a simplified collection of handwritten digits ranging from 0 to 9 for training and testing various ML algorithms [86]. The purpose of using this dataset was to evaluate the performance of EVAGAN against ACGAN in terms of the visual quality of the images generated in balanced and unbalanced scenarios.

4.3.4 Model Comparison of EVAGAN with ACGAN

For comparison, four different variants of GANs were constructed, respectively ACGAN_CC, EVAGAN_CC, ACGAN_CV, and EVAGAN_CV. ACGAN_CC and EVAGAN_CC were trained and tested on CC datasets, and ACGAN_CV and EVAGAN_CV used CV datasets. The implementation details of each version in terms of hyperparameters can be found in Table 4.2.

4.3.4.1 ACGAN_CC & EVAGAN_CC

The structure of ACGAN_CC and EVAGAN_CC was made up of a densely connected feed-forward neural network (FFNN) for both $\mathcal{G}$ and $\mathcal{D}$. The activation functions in hidden layers for both GANs were rectified linear units (ReLU). The hidden layers were regularised using batch normalisation, and the optimiser type was Adam with binary cross-entropy (BCE). The difference between ACGAN_CC and EVAGAN_CC is in the output layers of $\mathcal{D}$. $\mathcal{D}$ of ACGAN_CC outputs two neurons, one for the source probability and the other for the class probability for two classes (normal and botnet). The activation function is sigmoid for both outputs. The output layer structure of EVAGAN_CC has three neurons, one for the normal class, the second for the source probability, and the third for the botnet class (minority class). Each of the three outputs leverages the sigmoid as the activation function.

4.3.4.2 ACGAN_CV & EVAGAN_CV

The CV-based GAN architecture is different as it deals with image data compared to tabular data in CC. The convolutional neural network (CNN) was used instead of FFNN with other layers specific for image generation or detection. The output layer of $\mathcal{D}$ is similar to CC-based GAN implementations, except the Adam optimiser's loss function has BCE for source estimations and sparse categorical cross-entropy (SCCE) for class labels. Here, BCE could have been used; however, minimal changes to the code were made to maintain the integrity of the original ACGAN. However, in ACGAN_CC, BCE was used as the CNN-based code was converter to FFNN. In this way, ACGAN_CC and EVAGAN_CC could be kept as similar as possible for a fair comparison.

Table 4.2: CC and CV GAN Models

Parameter	**ACGAN_CC**	**EVAGAN_CC**	**ACGAN_CV**	**EVAGAN_CV**
Network Type	FFNN		CNN	
Number of Layers	$\mathcal{G}$: 5, $\mathcal{D}$: 5		$\mathcal{G}$: 2, $\mathcal{D}$: 3	
Activations	$\mathcal{G}$: ReLU, $\mathcal{D}$: LeakyReLU (output: sigmoid)		$\mathcal{G}$: ReLU (output: tanh), $\mathcal{D}$: ReLU (output: sigmoid, softmax)	
Batch Size (b)	256			
Neurons in input layer	$\mathcal{G}$: latent dimension, class label vector size, $\mathcal{D}$: feature size			
Neurons in layer 1	$\mathcal{G}$: 32, $\mathcal{D}$: 128		$\mathcal{G}$: 128, $\mathcal{D}$: 32	$\mathcal{G}$: 128, $\mathcal{D}$: 32
Neurons in layer 2	$\mathcal{G}$: 64, $\mathcal{D}$: 64		$\mathcal{G}$: 64, $\mathcal{D}$: 64	
Neurons in layer 3	$\mathcal{G}$: 128, $\mathcal{D}$: 32		$\mathcal{D}$: 128	
Neurons in output layer	$\mathcal{G}$: feature size, $\mathcal{D}$: 2	$\mathcal{G}$: feature size, $\mathcal{D}$: 3	$\mathcal{G}$: feature size, $\mathcal{D}$: 2	$\mathcal{G}$: feature size, $\mathcal{D}$: 3
Layer Regularization	$\mathcal{G}, \mathcal{D}$: $BatchNorm$			
Optimizer	Adam (beta_1=0.0002, beta_2=0.5)			
Loss Function	BCE		BCE, SCCE	
Learning Rate	5e-4			
Epochs	150			

4.4 Results

This section shows the results of the GAN implementations around two types of datasets: CC and CV GANs.

4.4.1 CC GANs

The results for quantitative analysis of $\mathcal{D}$'s performance on generated samples validity (GEN_VALIDITY), fake/generated botnet samples evasion (FAKE_BOT_EVA), real normal/majority class estimation (REAL_NORMAL_EST) and real botnet/minority class evasion (REAL_BOT_EVA) have been demonstrated in Figure 4.3. The ML classifier results are also shown in this figure for the three CC datasets for comparison. Equations from 4.11-4.14 represent the mathematical expressions for these performance indicators. The Keras *model.predict* function was used to compute the values where the *model* is $\mathcal{D}$ as the prime objective is to devise an intelligent evasion-aware classifier. Following is a brief detail of each evaluation parameter.

4.4.1.1 GEN_VALIDITY

In Equation 4.11, $\hat{\mathcal{G}}(z, c_m)[0]$ denotes the predicted value for the source being fake or real. The Keras *model.predict* function outputs an array, so the average of the first elements in the array will be the source validity of the generated samples after every epoch. The more this value is close to '1', the more it will be regarded as real.

$$GEN\_VALIDITY = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[0]]}{N} \tag{4.11}$$

4.4.1.2 FAKE_BOT_EVA

In Equation 4.12, $\hat{\mathcal{G}}(z, c\_m)[1]$ represents the probability estimation of generated minority/botnet class samples. Since the label for the minority/botnet class is '0' so ideally, it is expected that the model will output a value close to '0'. This estimation was represented as the evasion of the generated samples. So the more this value is close to '0', the less evasion will be. Note that this is the second value in the sum of the *model.predict* function output.

$$FAKE\_BOT\_EVA = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[1]]}{N} \tag{4.12}$$

4.4.1.3 REAL_NORMAL_EST

In Equation 4.13, $\hat{\mathcal{X}}_{normal_{test}}[2]$ represents the probability estimation of majority/normal class samples. Since the majority/normal class label is '1', ideally, it is expected that the model will output a value close to '1'. Note that this is the third value in the sum of the *model.predict* function output for the normal samples from the test set.

$$REAL\_NORMAL\_EST = \frac{\sum[\hat{\mathcal{X}}_{normal_{test}}[2]]}{N} \tag{4.13}$$

4.4.1.4 REAL_BOT_EVA

In Equation 4.14, $\hat{\mathcal{X}}_{botnet_{test}}[1]$ represents the probability estimation of the real minority/botnet class samples. The expectation from the model is to output the value close to '0', similar to FAKE_BOT_EVA. This is the second value in the sum of the *model.predict* function output for the botnet samples from the test set.

$$REAL\_BOT\_EVA = \frac{\sum[\hat{\mathcal{X}}_{botnet_{test}}[1]]}{N} \tag{4.14}$$

4.4.1.5 Losses

The losses of $\mathcal{D}$ for real and fake minority classes and majority/normal class and the loss of $\mathcal{G}$ have been demonstrated in Figure 4.4 for both ACGAN and EVAGAN.

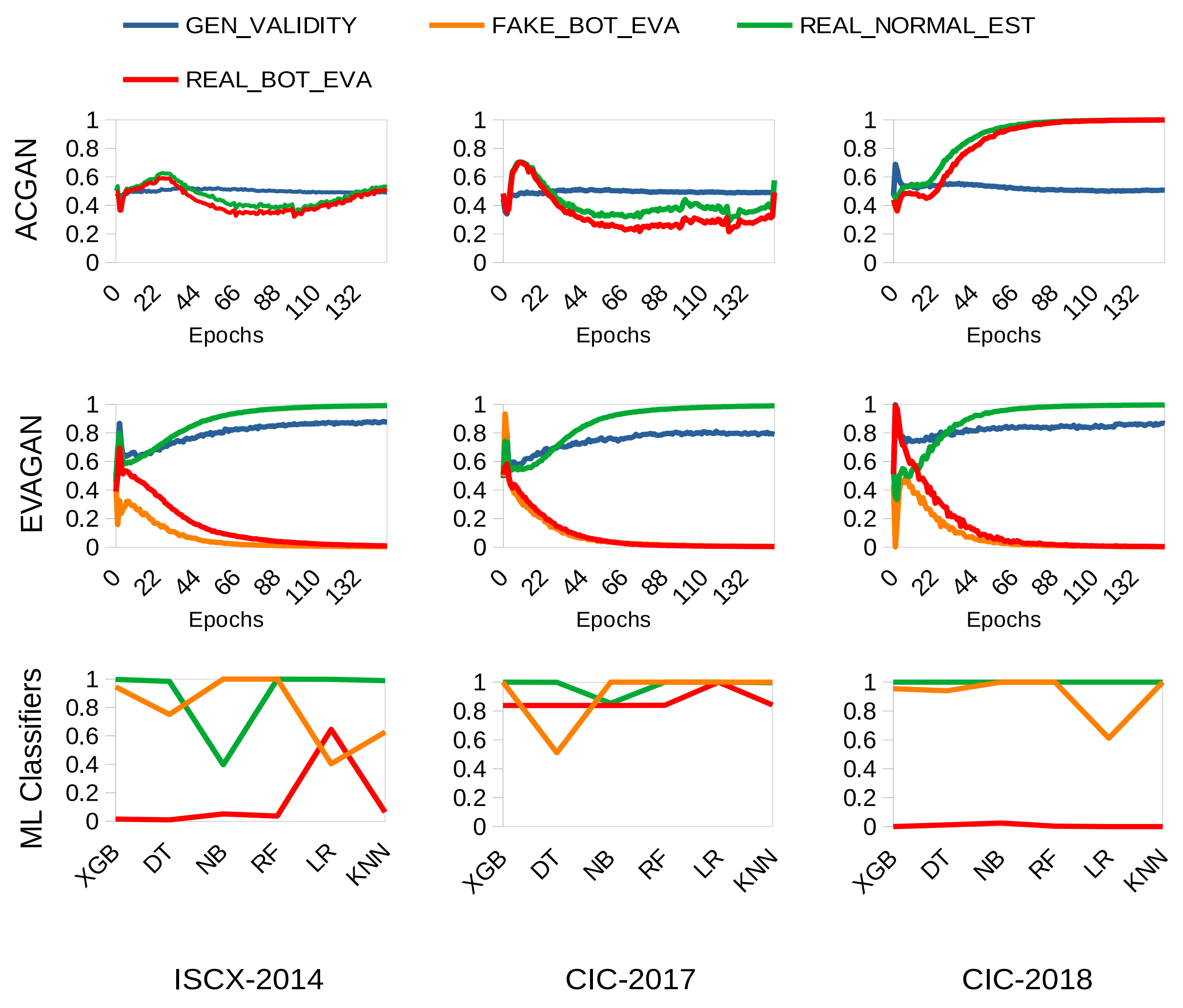


Figure 4.3: CC Estimations: The estimations on test data and data generated by the relative GANs along with the results of six different ML-classifiers

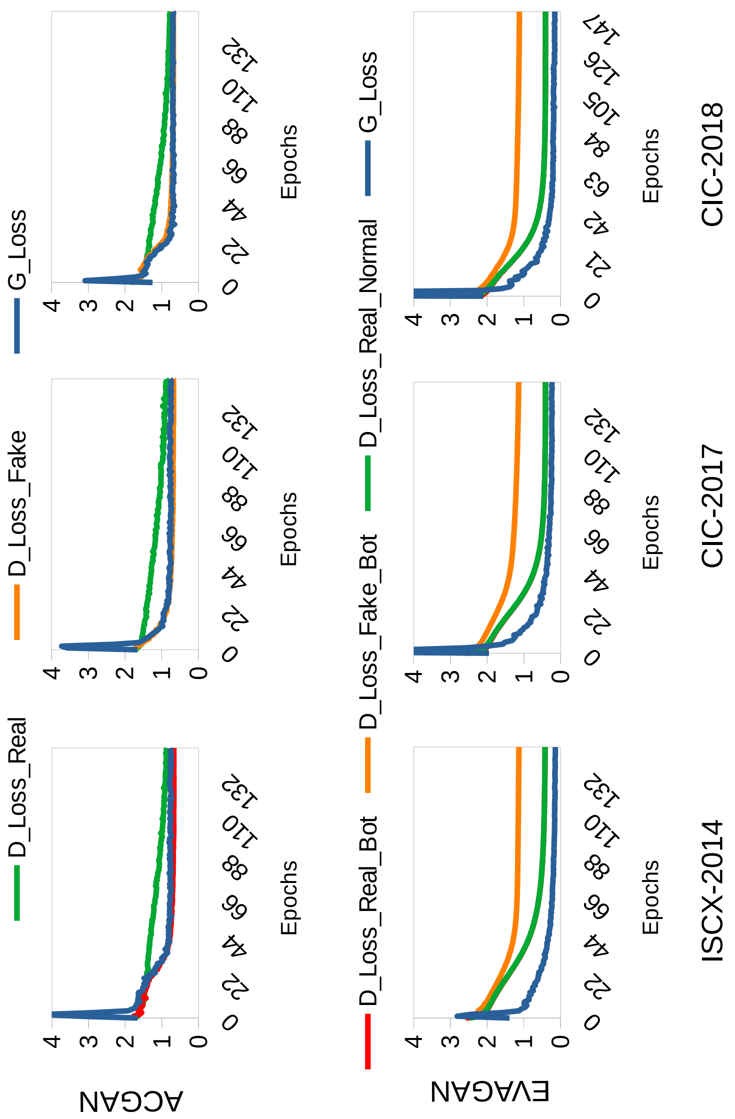


Figure 4.4: CC GANs Losses: The training losses for ACGAN and EVAGAN on three different CC datasets

4.4.2 CV GANs

For ACGAN_CV and EVAGAN_CV, MNIST handwritten digits dataset was used. Only two classes of digits, '0' and '1', were used in ACGAN_CV, as due to SCCE, its model does not accept fewer than two classes. For ECAGAN_CV, only the '0' digit as the minority class was used. Since the MNIST data is already balanced, it is desired to undersample the values of the '0' digit class to demonstrate the difference in performance. Four different undersampling levels have been devised in section 4.5. The evaluation parameters were equivalent to those used in CC GANs. For instance, GEN_Validity is the same as GEN_VALIDITY; GEN_Eva is similar to FAKE_BOT_EVA with the minority class from MNIST, i.e. '0' in this case. Similarly, ONE_Est is equivalent to REAL_NORMAL_EST, and ZERO_Eva is comparable to REAL_BOT_EVA in CC GANs. Figure 4.5 demonstrates the quantitative results for the four undersampling scenarios. Note that out of four; the first scenario exhibits 0% undersampling. There are three scenarios with undersampling, 50%, 90% and 99%. For qualitative analysis, the output from $\mathcal{G}$s of both ACGAN_CV and EVAGAN_CV have been demonstrated in Figures 4.7 and 4.8. These results are also based on the undersampling cases.

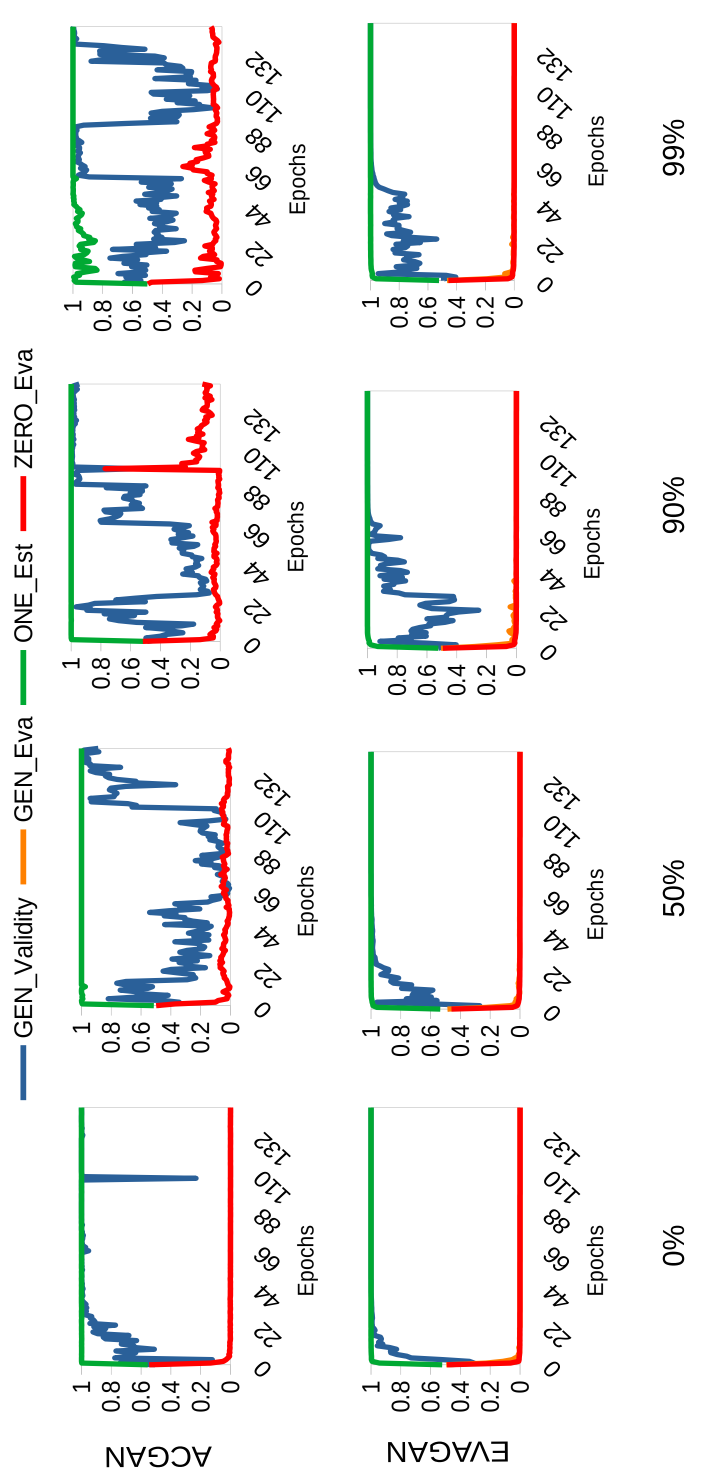


Figure 4.5: CV GANs estimations: The estimations on the test set for ACGAN and EVAGN for MNIST dataset in different undersampling scenarios. The range of the estimation value on the y-axis is from 0 to 1

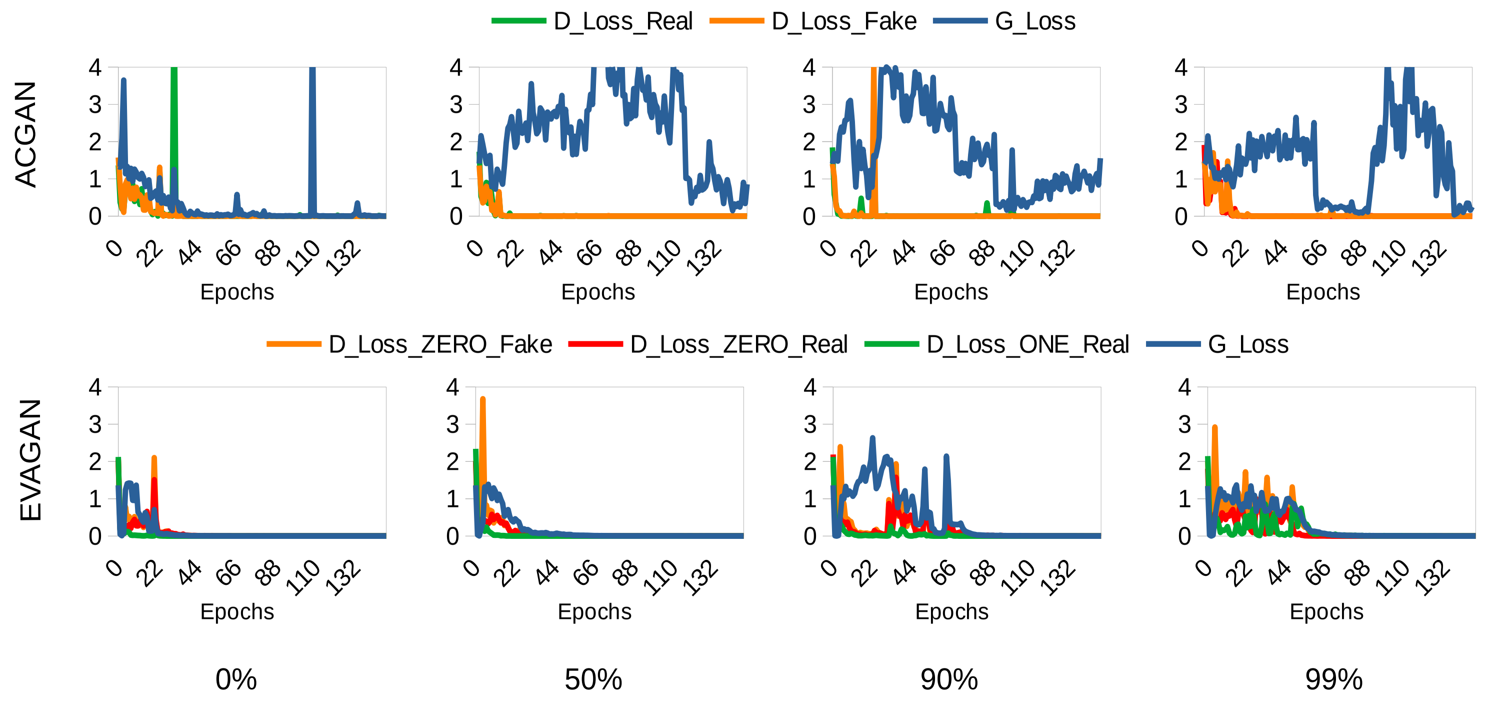


Figure 4.6: CV GANs losses: The training losses on train set for ACGAN and EVAGN for MNIST dataset in different undersampling scenarios. The upper limit to the loss has been fixed to 4 for consistency to highlight the difference.

4.5 Discussion

4.5.1 Detection Performance

In Figure 4.3, for the ACGAN_CC, the values for the REAL_NORMAL_EST and REAL_BOT_EVA remain close to each other. This implies that $\mathcal{D}$ of ACGAN_CC cannot discriminate between the majority and minority classes well due to the imbalance problem in all three CC datasets. $\mathcal{D}$ of ACGAN_CC remains confused for the two classes in ISCX-2014 and CIC-2017 datasets. For the majority class, ACGAN_CC performs equally well as EVAGAN_CC for CIC-2018 (shown in the second row of Figure 4.3). However, due to the small number, it regards the minority class samples as the majority class instances. The second row in Figure 4.3 shows the results of EVAGAN_CC for the estimations on the test set. It can be observed that as compared to ACGAN_CC, $\mathcal{D}$ of EVAGAN_CC perfectly differentiates between the majority and minority classes and, after each epoch, tends to improve its detection performance for all the three CC datasets.

The FAKE_BOT_EVA was used as an indicator of evasion awareness of $\mathcal{D}$ in the case of EVAGAN_CC only because ACGAN_CC generates two classes of data, so $\mathcal{G}$ of ACGAN_CC would generate a random number of samples from both classes leading to non-deterministic values of FAKE_BOT_EVA. However, the performance of this metric was compared with ML classifiers. The last row of Figure 4.3 shows the results of the six different ML classifiers for the values of the majority, minority and generated class samples. It can be inferred that EVAGAN_CC tends to outperform the ML classifiers for all three values after a certain number of epochs. The ML classifiers for black-box testing perform worst in the case of FAKE_BOT_EVA as compared to EVAGAN_CC for all three CC datasets. This implies that $\mathcal{D}$ of EVAGAN_CC is not only adept at discriminating between real minority samples but can also easily detect the fake minority samples that ML classifiers are not good at discerning. Another significant advantage of this $\mathcal{D}$ is that it is not needed to employ ML classifiers in CC for learning adversarial evasion. Researchers use GANs to generate adversarial samples to be augmented with the training set for retraining ML classifiers to make them evasion-aware. In the case of EVAGAN_CC, that time could be saved, as $\mathcal{D}$ classifier/detector model is trained alongside the GAN training.

It can be further illustrated from Figure 4.3 that the value of GEN_VALIDITY in the case of ACGAN_CC seems to remain close to 0.5 for all three CC datasets. It means that $\mathcal{D}$ is confused in deciding whether the generated samples from $\mathcal{G}$ are real or fake. However, in the case of EVAGAN_CC, for all three datasets, $\mathcal{G}$'s performance is improving with each epoch. This implies that $\mathcal{D}$ is being fooled and still learning, while in the case of ACGAN_CC, $\mathcal{D}$ has already been saturated because $\mathcal{G}$ is not generating new samples that can fool $\mathcal{D}$.

4.5.1.1 CV GANs

Figure 4.5 demonstrated the results of different undersampling scenarios to mimic the low data regimes for the MNIST dataset. Note that the detection performance of $\mathcal{D}$ for both ACGAN_CV and EVAGAN_CV for the majority and minority classes remains ideal from the very start. The reason is that, unlike CC datasets, the CV dataset has many strong features due to which $\mathcal{D}$ is easily able to differentiate between the '0' digit and '1' digit samples. However, the undersampling effect can be seen for the minority class or digit '0' data. In contrast, $\mathcal{D}$ of EVAGAN_CV seems to be smart enough to give steady values for all the undersampling cases, especially for minority class evasion (as depicted in red colour lines). Due to the sufficient number of samples, the majority class should be detected easily by both GANs. However, in the case of 99% undersampling, ACGAN_CV exhibits a poor performance even detecting this class. For GEN_Validity (represented by the blue lines), Figure 4.5 shows that in undersampling cases, the performance of $\mathcal{G}$ of ACGAN_CV deteriorates in the worst manner and does not show any useful pattern of learning. This implies that in low data regimes, $\mathcal{G}$ is not performing as well as EVAGAN_CV is. However, EVAGAN_CV also shows the deterioration in $\mathcal{G}$'s performance, but that is not as phenomenal as that of ACGAN_CV.

4.5.2 Stability

The Figure 4.4 shows $\mathcal{D}$ and $\mathcal{G}$ losses for CC GANs. It can be inferred from this diagram that the values for all the losses seem to be converging. This shows that the GANs are saturating towards Nash equilibrium. However, in the case of EVAGAN_CC, the losses tend to be more steady with each epoch and achieve the lowest point sooner than ACGAN_CC. Similarly, for CV GANs, the EVAGAN_CV losses in all the undersampling cases tend to be more stable compared to ACGAN_CV as demonstrated in Figure 4.6.

4.5.3 Qualitative Performance

It is non-trivial to demonstrate the performance of a GAN in the case of CC datasets [87]. Since it was not possible to visualise the generated network traffic, so it was desired to validate the EVAGAN with the help of CV datasets. The rationale for using CV datasets is that if EVAGAN outperforms ACGAN in unbalanced scenarios, then it would be equally acceptable for CC datasets. Since the purpose is not to generate quality traffic for CC, it was needed to design an evasion-aware anomaly detector. So, evaluating EVAGAN_CC for quality traffic generation is not within the scope of this work.

The previously mentioned undersampling scenarios for CV GANs have been demonstrated in Figures 4.7 and 4.8. There are two 15×10 matrices of pictures in each figure. The number of images in each matrix equals the total number of epochs, i.e. 150. In each figure, the upper row belongs to the ACGAN_CV output of $\mathcal{G}$, and the lower row corresponds to the output from $\mathcal{G}$ of EVAGAN_CV. Note that for ACGAN_CV, two classes are output from $\mathcal{G}$, and for EVAGAN_CV, only one '0' digit class is generated. For the undersampling scenario, which contains 50% fewer '0' class samples, the deterioration for ACGAN_CV starts getting evident, but EVAGAN_CV can generate '0' digits. For the case of 90% undersampling, the ACGAN_CV quality miserably deteriorates; however, EVAGAN_CV is still generating the '0' class samples, although slightly faded. In the 99% undersampling case, as expected, the ACGAN_CV is still struggling to generate the minority class digit '0', but an interesting case has happened for EVAGAN_CV. Since the number of samples is minuscule, the feedback taken from $\mathcal{D}$ by $\mathcal{G}$, on some accidentally generated '1' digit, gave a small value of G_Loss. Due to this, $\mathcal{G}$ started generating the majority class '1' digit after epoch 47. This situation is called a mode collapse, an inherent problem in GANs. However, it can be inferred that EVAGAN_CV may not perform well in a highly unbalanced scenario. This is an interesting research direction to investigate further using other CV datasets. On the other hand, ACGAN_CV is also stuck in mode collapse after epoch 140, where in place of class '0', the '1' class samples start appearing. However, in the case of EVAGAN_CV, despite mode collapse, the generated samples from class '1' are of higher quality which means that its $\mathcal{G}$ is more powerful than that of ACGAN in highly unbalanced scenarios.

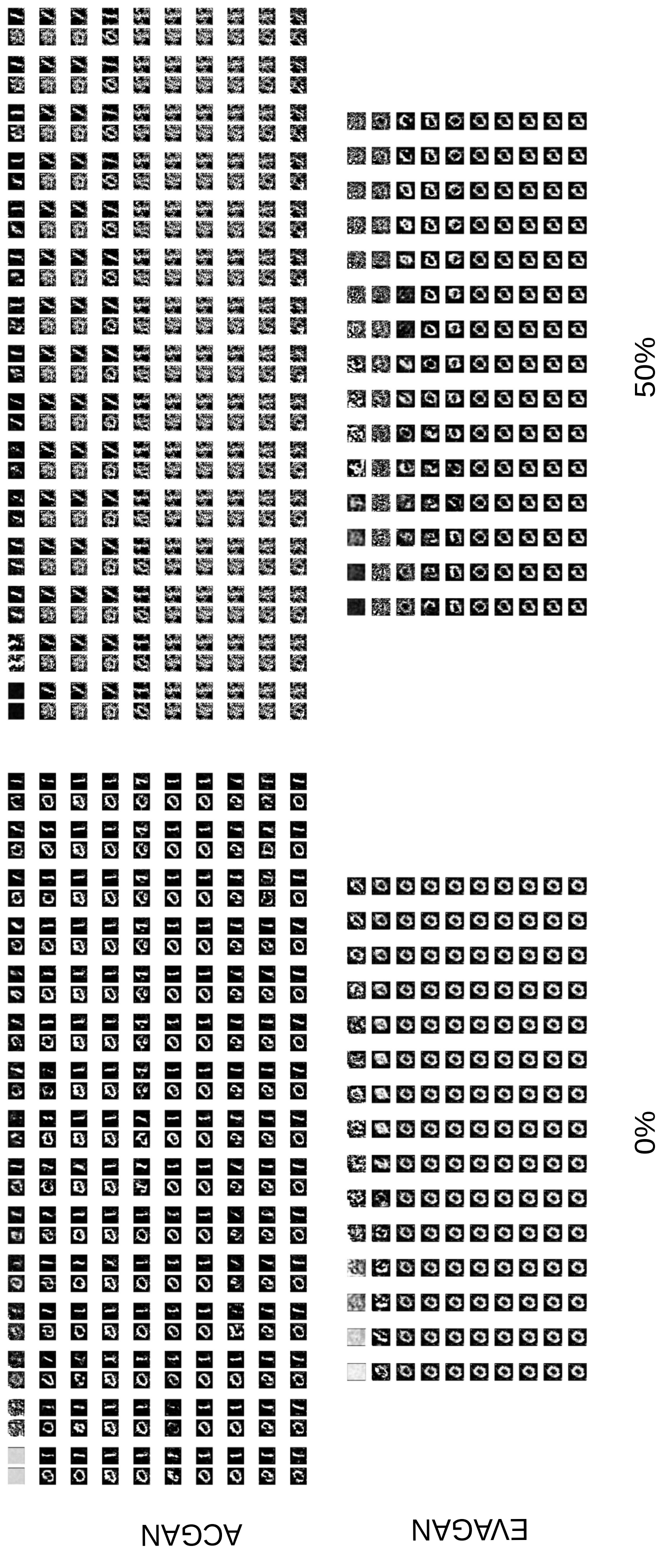


Figure 4.7: Qualitative Analysis of $\mathcal{G}$ for CV GANs with 0% and 50% Undersampling of '0' digit class

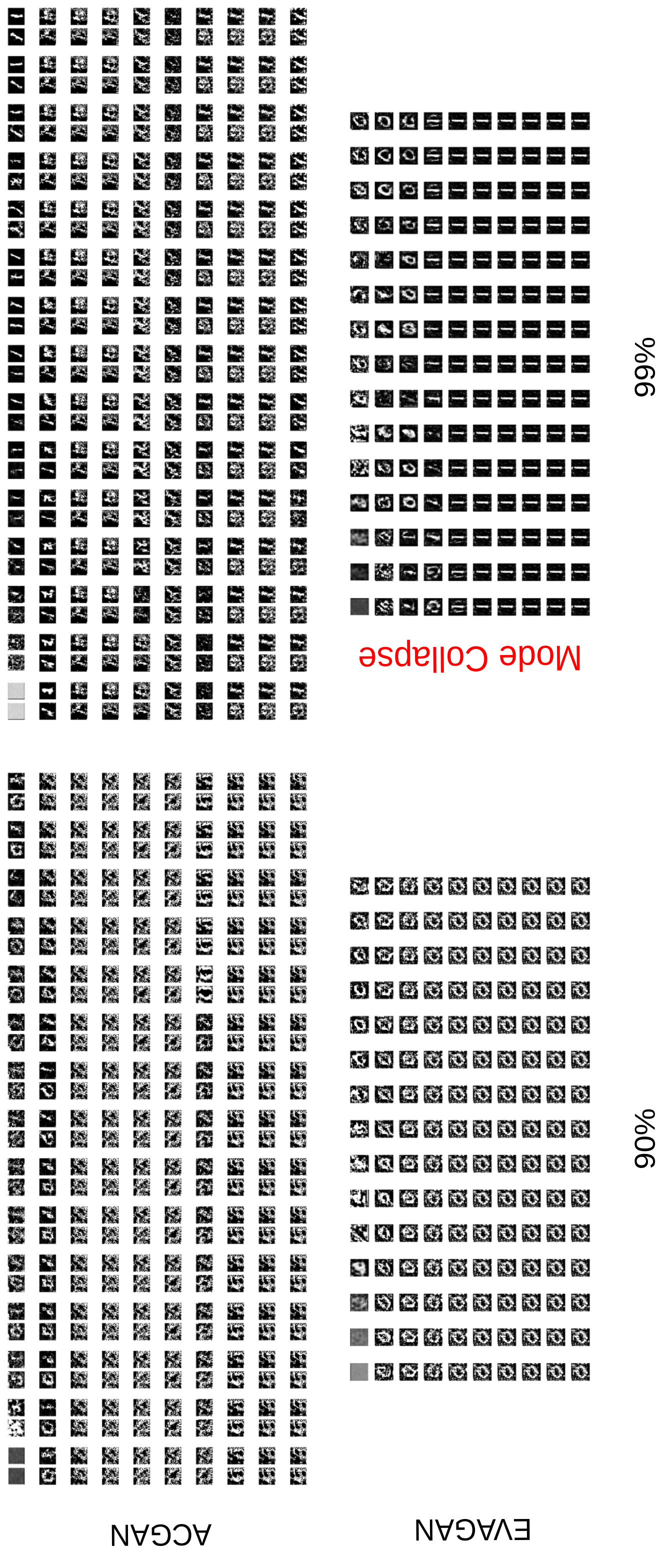


Figure 4.8: Qualitative Analysis of $\mathcal{G}$ for CV GANs with 90% and 99% Undersampling of '0' digit class

4.5.4 Time Complexity

The time complexity bar chart has been demonstrated in Figure 4.9, where the y-axis represents the values of the training time in minutes. The MNIST dataset case with no undersampling was used to compare the results. The time complexity may vary on different platforms (for instance, Google Colab); however, the plot in Figure 4.9 shows the results on the workstation that was previously used (as mentioned in section 4.3). It can be observed that EVAGAN always takes less time than its counterpart for all four datasets. The reason is that $\mathcal{G}$ of EVAGAN, in the cases of all the datasets, needs to follow lesser diversity compared to ACGAN. Although the batch size of 256 (given in Table 4.2) is the same for both GANs, the amount of time taken by EVAGAN is always less. Due to the stochastic nature of the input noise z for $\mathcal{G}$, the exact time in minutes can not be estimated for every training cycle; however, the average time of EVAGAN always remains less as compared to ACGAN.

A question might arise why ACGAN was not made to generate only the minority class samples. The answer to this question is that there would be a need to make changes in the structure of both $\mathcal{G}$ and $\mathcal{D}$ along with the loss functions. The SCCE loss does not allow us to use less than two classes, so BCE loss needs to be used with other structural modifications. EVAGAN is the name of this transformation.

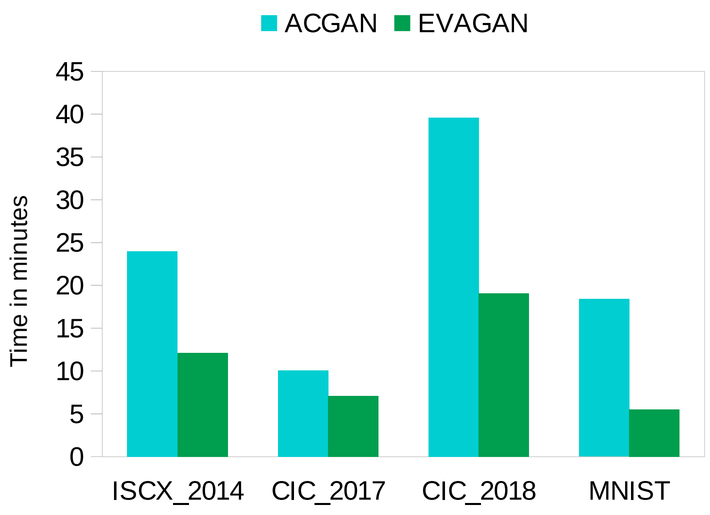


Figure 4.9: Time Complexity

4.6 Comparison of EVAGAN with Peer Techniques

The EVAGAN model is an enhanced version of ACGAN, dedicated to low data regimes for learning adversarial evasion examples generated during GAN training. So the most suitable existing model for the comparison can be ACGAN, the details of which have been explained in section 4.5 previously. However, this section mentions peer techniques similar to EVAGAN that indirectly address the adversarial evasion problem. The comparison in the following subsections has been summarised and then in the form of a Table 4.3.

4.6.1 Data Augmentation

There are several techniques both in CV and CS that propose the data augmentation for enhancing the ML classifiers' performance [19, 62, 65, 88–92] . However, EVAGAN itself acts as a powerful adversarial evasion-aware model in which the discriminator ($\mathcal{D}$) acts as a classifier. So there is no need to generate evasion samples from a GAN model, augment with the training set and then train a separate ML classifier. This property of EVAGAN makes it superior to all the techniques based on data augmentation in terms of time complexity.

4.6.2 Computer Vision vs Cybersecurity Low Data Regimes

Many works address the problem of low data regimes [93–99]. However, their datasets and model architectures differ from those used in this work. A few have been mentioned in Table 4.3. Since ML studies are biased towards data, experimenting with other datasets can be a potential future work.

4.6.3 Architecture Comparison

Authors in [24] proposed a model in which the discriminator acts as a multiclass classifier; however, their work is not destined towards adversarial evasion generation in low data regimes as they are considering the normal class samples to train the generator of their GAN. The thesis work differs in a way that the generator ($\mathcal{G}$) is not fed with normal class samples, which makes $\mathcal{G}$'s job easier. This saves training time and improves the estimation accuracy of malicious samples, even being scanty.

4.6.4 Accuracy and Time Complexity

EVAGAN produces ideal results of estimation for both majority and minority classes, as high as 100% for all the datasets used, as mentioned in section 4.4. The comparison has been provided with ACGAN; however, compared with other similar models, the accuracy is also at par. The accuracy values determined from the literature for some other models addressing similar problems have been given in Table 4.3. The time complexity compared to the ACGAN model has been discussed in section 4.4; however; it would be non-trivial to compare with other peer models in respect of training time as the model architecture and hyperparameters vary enormously. The experiments for EVAGAN and ACGAN were performed on the same machine as mentioned in section 4.3, so the claim is made with the time complexity comparison with ACGAN only.

Table 4.3: Comparison with Peer Works

Paper	**Addressing Evasion Problem**	**Adversarial Training/ Augmentation**	**Low Data Regime**	**Architecture**	**Datasets Used**	**Maximum Accuracy**
ID-GAN [24]	✗	✗	✓	multiclass ACGAN	NSL-KDD	83.10%
G-IDS [65]	✗	✓	✓	vanilla GAN	NSL-KDD	-
AE-CGAN [88]	✗	✓	✓	Auto Encoder with Conditional GAN	CIC-2017	100%
[89]	✓	✓	✗	ANN, CNN, RNN	UNSW-NB15, NSL-KDD	97%
Attack-GAN [62]	✓	✓	✗	Sequence GAN	CTU-13	-
GADoT [90]	✓	✓	✗	WGAN-GP	Custom-SYN, Scapy-SYN, CICIDS2017, UNB201X	-
DIGFuPAS [91]	✓	✓	✓	WGAN	CICIDS2017	-
min-max Training [100]	✓	✓	✓	DNN	NSL-KDD	93.4%
attackGAN [101]	✓	✗	✓	WGAN	NSL-KDD	-
CVAE-AN [92]	✓	✓	✓	Conditional VAE and GAN	CICIDS2017	98%
CEGAN [102]	✗	✓	✓	CNN	MNIST, EMNIST, F-MNIST CIFAR-10, CINIC-10	96.48%
EVAGAN	✓	✗	✓	binary class ACGAN	ISCX-2014, CIC-2017, CIC-2018, MNIST	100%

4.7 Summary

Adversarial evasion attacks on AI-based systems are a portending threat that needs to be dealt with using intuitive methods. Adversarial learning is one of the modern techniques to make ML classifiers proactively adept at detecting adversarial evasion samples. In this chapter, a novel GAN model called EVAGAN is proposed that generates adversarial evasions in low data regimes.

- EVAGAN is an enhancement of a well-known model called ACGAN. The purpose of EVAGAN is to design an evasion-aware classifier for anomaly detection.
- Two types of datasets were used; one from the cybersecurity domain for botnets and the other from the computer vision called MNIST.
- EVAGAN's discriminator proves superior to ACGAN in terms of detection performance, stability, and time complexity.
- At the same time, the qualitative analysis shows that EVAGAN outperforms ACGAN in unbalanced scenarios.

EVAGAN model has been designed for binary classification problems. Further investigation for multiclass design is a potential research direction. Experiments with other datasets would be highly desirable to evaluate EVAGAN for the said parameters further. For the qualitative analysis, handwritten digits other than '0' and '1' could further validate EVAGAN's superiority over ACGAN. A comparison with few-shot learning could be an exciting research direction.

Chapter 5

Deep Reinforcement Learning-based Evasion Generative Adversarial Network

5.1 Introduction

Botnet detectors based on machine learning are potential targets for adversarial evasion attacks. Several research works employ adversarial training with samples generated from generative adversarial nets (GANs) to make the botnet detectors adept at recognising adversarial evasions. However, the synthetic evasions may not follow the original semantics of the input samples. This chapter proposes a novel GAN model leveraged with DRL to explore semantic aware samples and simultaneously harden its detection. A DRL agent is used to attack the discriminator of the GAN that acts as a botnet detector. The discriminator is trained on the crafted perturbations by the agent during the GAN training, which helps the GAN generator converge earlier than the case without DRL. This model is named as RELEVAGAN, i.e. [”relive a GAN” or deep REinforcement Learning-based Evasion Generative Adversarial Network] because, with the help of

DRL, it minimises the GAN's job by letting its generator explore the evasion samples within the semantic limits. During the GAN training, the attacks are conducted to adjust the discriminator weights for learning crafted perturbations by the agent. RELEVAGAN does not require adversarial training for the ML classifiers since it can act as an adversarial semantic-aware botnet detection model. The code will be available at https://github.com/rhr407/RELEVAGAN.

RELEVAGAN is an effort toward a unifying model concept that would solve the problem of data imbalance, provide adversarial semantic awareness and save training time. RELEVAGAN is equipped with an integrated DRL agent to achieve the said goals. RELEVAGAN name was chosen for two reasons. First, it is a deep REinforcement Learning-based Evasion Generative Adversarial Network. Second, it relieves the employed GAN model to make its job easier by letting the generator of the GAN explore the semantic-aware samples, i.e. within certain boundary conditions. Either way, RELEVAGAN proves to be an improved technique compared to the peer models like Auxiliary Classifier GAN (ACGAN) and Evasion Generative Adversarial Network (EVAGAN).

The DRL agent attacks the RELEVAGAN's discriminator, which acts as a botnet detector. The attack generation is based on manipulating the real attack samples to evade the botnet detector. As the RELEVAGAN training proceeds, the agent learns to evade the botnet detector. The discriminator is adversarially trained on the DRL attacker's evaded samples and the generator's synthetic samples in each training iteration. After a certain number of epochs, the discriminator becomes hardened against the samples from the DRL agent and the generator. The detection estimations for the benign, real and generated samples, and the generator training settle to the desired values in fewer training iterations than the EVAGAN model. The experimental analysis shows the considerable performance of the RELEVAGAN model against EVAGAN in terms of detection estimation and stability of training for three different botnet datasets. It is argued that the learning of GAN follows semantic awareness because GAN is also trained on the attacks generated by the DRL model.

The following are the main contributions of this chapter:

1. A novel DRL-based GAN model is proposed to address the problems of data imbalance, evasion awareness and functionality preservation in synthetic botnet traffic generation.
2. By help of experiments it was demonstrated that DRL plays a role in GAN training for the detection of synthetic as well as real attacks.
3. Because of the integrated self-learning DRL attacker, the proposed model can be envisaged as sustainable against evolving botnet.
4. It was determined that RELEVAGAN outperforms EVAGAN in terms of early convergence of the training.

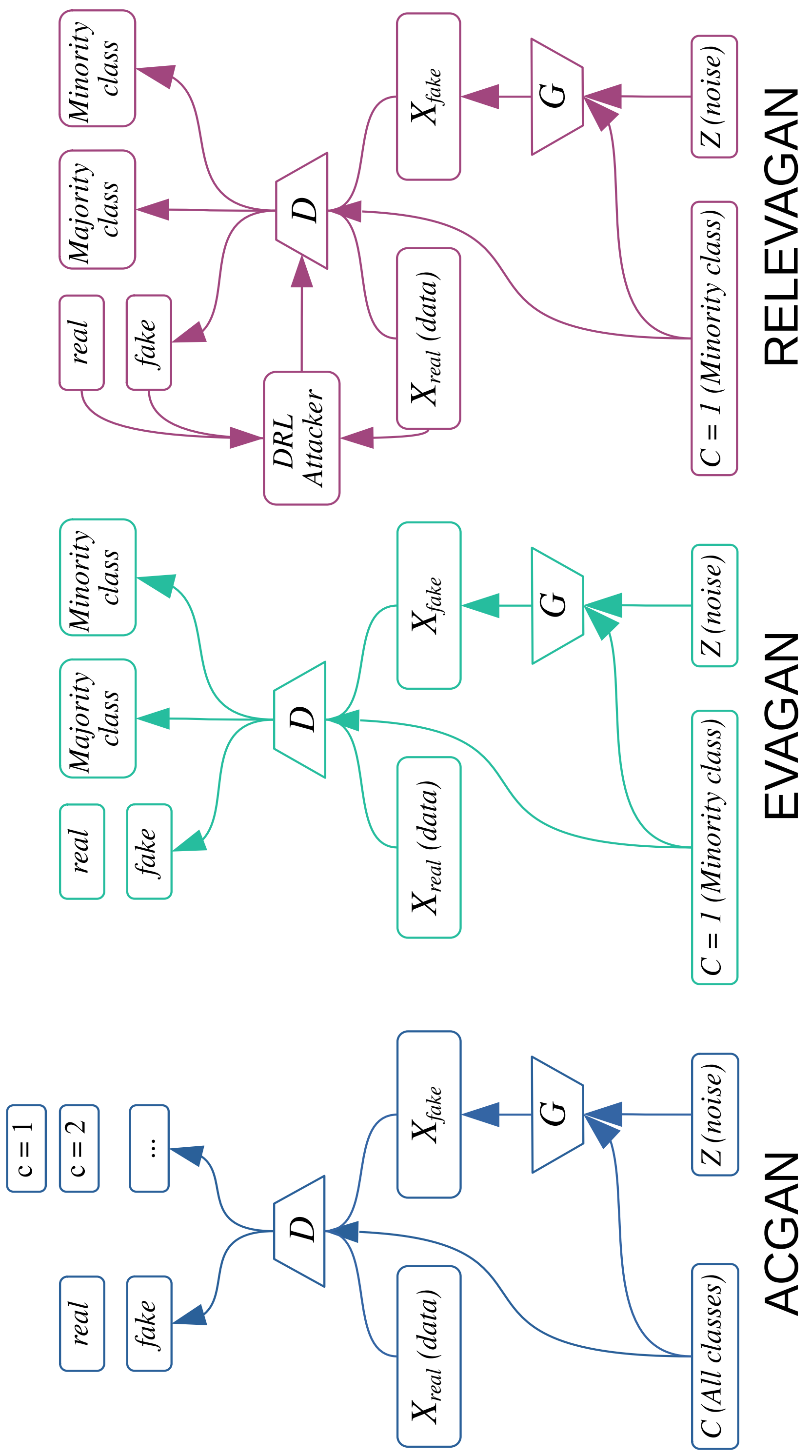


Figure 5.1: Comparison of RELEVAGAN model with ACGAN and EVAGAN

5.2 RELEVAGAN

In this section, the motivation behind the design of RELEVAGAN is discussed, especially the structural explanation of its DRL attacker. As illustrated by Figure 5.1, the DRL attacker has been introduced in RELEVAGAN. The rest of the architecture is similar to EVAGAN. The impact of using the DRL attacker helps improve the EVAGAN performance, the details of which will be discussed in section 5.2.

5.2.1 Motivation

GAN-based samples generation follows the probability distribution of the input data samples as $\mathcal{G}$ trains itself based on the feedback from $\mathcal{D}$. $\mathcal{G}$ tries to explore the new sample spaces that are unknown by $\mathcal{D}$ so that it can fool $\mathcal{D}$; however, this process can lead to the creation of samples that may not follow the real malicious semantics. To address this issue, DRL samples can help $\mathcal{G}$ learn the boundaries of the real samples. For this reason, a DRL agent can be leveraged to explore samples in a defined observation space. $\mathcal{D}$ can be trained on these generated samples, giving the feedback to $\mathcal{G}$ in the GAN training. Eventually, $\mathcal{G}$ can start learning from the updated feedback from $\mathcal{D}$ and generates the samples within a defined range set by the DRL agent. This process can help converge $\mathcal{G}$ training earlier while at the same time achieving high accuracy in a lesser number of epochs. Although semantic awareness comes with additional training costs for the DRL part but motivated by this rationale, the RELEVAGAN is a step further toward a more intelligent functionality-preserving GAN design.

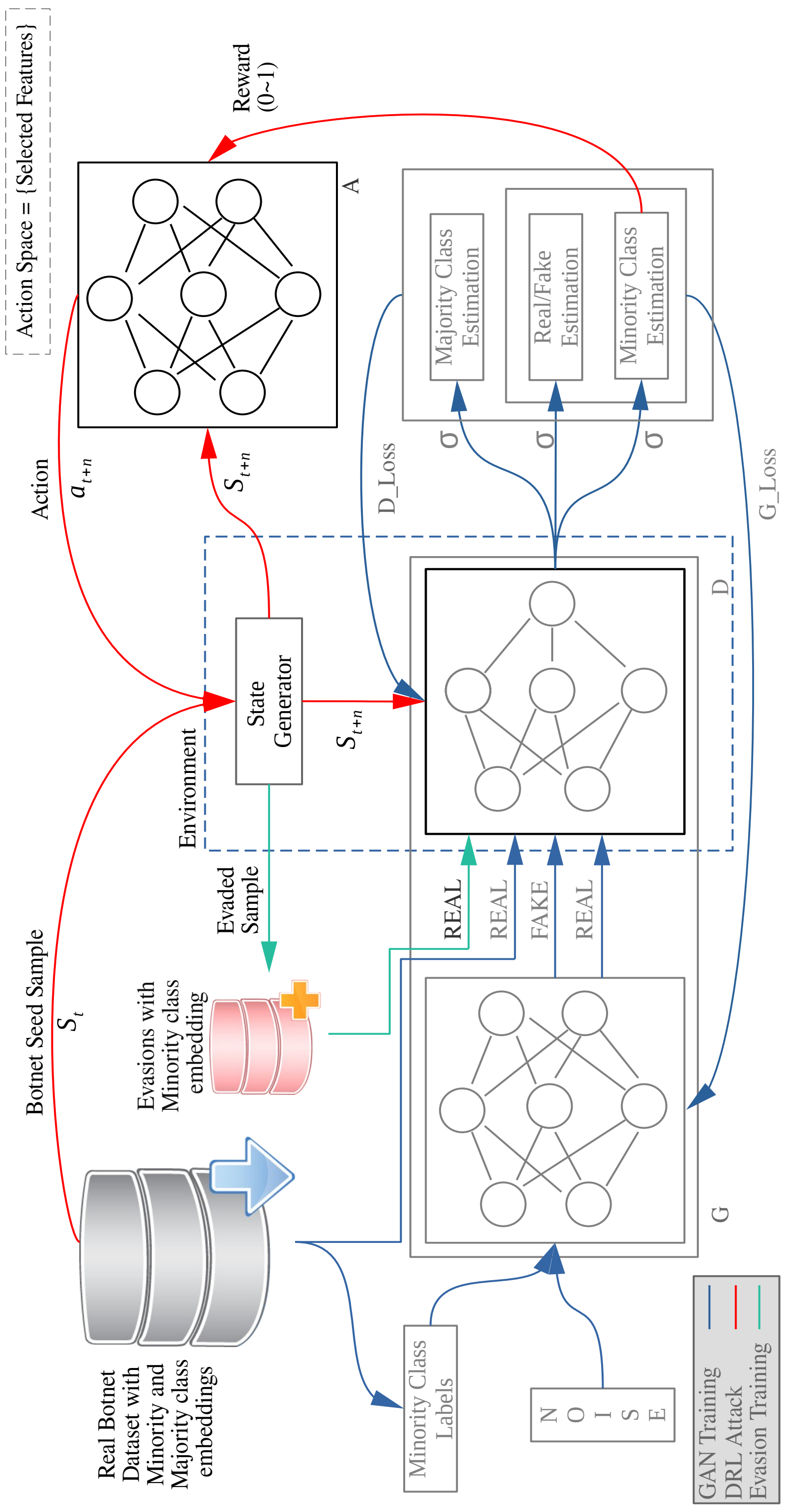


Figure 5.2: RELEVAGAN Architecture

5.2.2 Architecture

Figure 5.2 shows the architecture of RELEVAGAN. The structure of RELEVAGAN has mainly two components. One is EVAGAN, from Chapter 4, and the other part is a DRL model. Like other DRL attackers for evasion generation using a black box attack, [25–27, 69, 70], the proposed DRL agent attacks $\mathcal{D}$ of EVAGAN, which acts as a black box classifier. $\mathcal{D}$'s output for minority class estimation is used as the reward for the agent to adjust its weights and generate a new action a_{t+n} based on some policy π. The new action is fed to the environment where the state generator creates a new state taking another seed sample from the real data set. As the result of a single iteration, the new state and the collected reward are fed back to the agent. As a result of the positive reward, the evasion samples are fed to $\mathcal{D}$ of EVAGAN to adversarially train it. In this way, $\mathcal{D}$ becomes proactively aware of any possible future evasions and ready to give better feedback to $\mathcal{G}$ to train to confine its boundaries for evasion generation. The process leads to the early convergence of $\mathcal{G}$ training.

5.2.3 Environment

The environment in RELEVAGAN consists of mainly two parts:

5.2.3.1 State Generator

The state generator is responsible for three different jobs:

- It takes a botnet seed sample as the current state S_t from the real botnet dataset and transforms it into a feature vector accepted by $\mathcal{D}$ of EVAGAN based on some action index n coming from the agent.
- It feeds back the new state S_{t+n} to the agent.
- If the sample is evaded by $\mathcal{D}$, the state generator is also responsible for storing and/or feeding it to $\mathcal{D}$ to train in parallel with EVAGAN training adversarially.

5.2.3.2 Botnet Detector

The target model is $\mathcal{D}$ of EVAGAN. In Figure 5.2, the EVAGAN model has been illustrated in grey-coloured border lines, and the difference has been highlighted in black lines to understand better where the RELEVAGAN is different from EVAGAN.

5.2.4 Action Space

The following feature set gives the action space through which the agent chooses the most appropriate index n to gain the maximum reward by evading the target botnet model. These features have been chosen based on the work in [25] for the three datasets used in this work. The details of the datasets are mentioned in Section 5.3.

- FlowDuration
- FlowBytes/s
- FlowPackets/s
- FwdPackets/s
- BwdPackets/s
- TotalLengthofFwdPacket
- TotalLengthofBwdPacket
- BwdPackets/s
- SubflowFwdBytes
- FwdHeaderLength
- BwdHeaderLength
- Down/UpRatio
- AveragePacketSize

To keep the functionality reservation, the change in the feature value is limited to Δ, which is the minimum value of the particular feature within the data set as given by Equation 5.1.

$$\Delta_n = \min_{\forall m \in F_n} X(m) \tag{5.1}$$

In Equation 5.1, Δ_{F_n} is the minimum value for all the rows m of a particular feature F and n is the action index coming from the agent as a particular feature number from the action table.

5.2.5 Agent

The agent is a deep neural network with the size of the observation space as input and the number of actions as the output. The observation space in RELEVAGAN is a complete botnet sample as a feature vector. The agent is responsible for choosing an action index n among the features in the action table as mentioned in Subsection 5.2.4. Based on this index, a training step is executed, which feeds back the reward and the new state to the agent.

5.2.6 Reward

The reward in a typical botnet evasion generation model using a DRL black-box attack is the output of the botnet detector [25, 27], which can be a real number in the range of [0, 1]. In the proposed case, for the botnet sample, the expected value is ’0’, and for a normal traffic sample, the output should be ideally ’1’. Hence for a botnet sample to be considered a successful evasion by a DRL attacker, the threshold is set for the reward to be greater than 0.5. In other terms, if the sample generated by the DRL attacker is evaded with more than 50% confidence, the reward will be ’1’ and ’0’ otherwise.

5.2.7 Training

RELEVAGAN training is similar to EVAGAN except for adding a couple of extra steps for the DRL agent after every training batch. In Algorithm 4, steps 3 and 4 discriminate the training between EVAGAN and RELEVAGAN for a defined number of batches. The sequence of the steps is crucial for understanding the rationale behind RELEVAGAN. Note that $\mathcal{G}$ is trained after the evasion training of $\mathcal{D}$. Since $\mathcal{D}$'s weights are adjusted as per the evasions generated by the DRL attacker in Step 4, it will feed the G_Loss back to$\mathcal{G}$ more cognitively as compared to the case of EVAGAN training. The DRL attack is executed in every batch of training.

Algorithm 4 RELEVAGAN Training

```
1: for i = 1, 2, 3, ..., number of batches do
2:     Step 1: Train $\mathcal{D}$ on real data
3:     Step 2: Train $\mathcal{D}$ on generated data
4:     Step 3: Execute DRL $\mathcal{A}$ for generating evasion on batch size
5:     Step 4: Train $\mathcal{D}$ on $\mathcal{A}$ generated evasions
6:     Step 5: Train $\mathcal{G}$
7: end for
```

5.3 Implementation Details

5.3.1 Experimental Setup

As mentioned in Chapter 4, the experiments for the RELEVAGAN were performed on a GPU workstation, AMD Ryzen threadripper 1950x, equipped with a 16-core processor and an 8GB memory GeForce GTC 1070 Ti graphics card. The OS used was ubuntu 20.04, running Keras, TensorFlow, Sklearn and Numpy libraries within the Jupyter notebook. The source code of RELEVAGAN is also available on GitHub under MIT license[1].

5.3.2 Data Preparation

For experimentation, the CIC botnet datasets, ISCX-2014, CIC-2017 and CIC-2018, were used according to Chapter 3 and 4.

[1]https://www.github.com/rhr407/RELEVAGAN

5.3.3 DRL Attacker

For the implementation of the DRL attacker, the OpenAI Gym and gym-malware tool kits [26, 103] were used. Keras-rl and Keras-rl2 libraries were used for the selection of the DQN agent. Unlike a typical DRL algorithm, a new training session was executed in every batch of the RELEVAGAN training where the weights of the neural network are not reset to ensure that the agent is learning in each batch iteration of RELEVAGAN. The reason for keeping the weights is that it's non trivial estimate the actual number of training iterations that would traverse the whole batch of botnet samples for generating manipulations. Hence, the session is reinitialised after each batch keeping the agent's neural network unchanged. A single training session in each batch iteration lasts until the following two cases appear:

- The evasion is successful.
- The number of tries saturates.

In either of the cases mentioned above, a new botnet seed sample is selected until the total number of samples in a batch is traversed. Each training step takes an action index that selects the corresponding feature from the botnet seed sample for manipulation. The modified sample is tried on the trained $\mathcal{D}$ using the Keras $model.predict$ function, which gives the estimation of the botnet sample being from a minority class. This output estimation is used to set the agent's reward as mentioned in subsection 5.2.6. As a result of each step, the reward and the new state (alternatively, the manipulated botnet sample) are returned to the agent. The details of the hyperparameters of the DRL attacker part have been mentioned in Table 5.1 and Table 5.2 respectively.

Table 5.1: Hyperparameters of DRL Attacker

Parameter	**Value**
Agent Type	DQN
Action Space	13
Policy	BoltzmannQPolicy
Double DQN	True
Target Model Update	1e-3
Number of turns	13
Number of rounds	256

Table 5.2: Hyperparameters of DRL Neural Network

Parameter	**Value**
Network Type	FFNN
Number of Layers	4
Activations	ReLU, linear
Neurons in input layer	size of observation space
Neurons in layer 1	64
Neurons in layer 2	128
Neurons in output layer	number of actions
Layer Regularization	$BatchNorm$

5.4 Results & Discussion

The performance analysis of RELEVAGAN is identical to that of EVAGAN for botnet datasets. The metrics used were generated samples validity (GEN_VALIDITY), fake/generated botnet samples evasion (FAKE_BOT_EVA), real normal/majority class estimation (REAL_NORMAL_EST), and real botnet/minority class evasion (REAL_BOT_EVA). Figure 5.3 shows the results for the estimations of ACGAN, EVAGAN and RELEVAGAN for comparison. The mathematical expressions for the evaluation metrics have been represented using Equations 5.2-5.5. These estimations were computed using the Keras $model.predict$ function. The details of these metrics can be found in [104]. Figure 5.4 illustrates the losses of $\mathcal{D}$ for real and fake minority classes and majority/normal classes and of $\mathcal{G}$ for ACGAN, EVAGAN and RELEVAGAN.

$$GEN\_VALIDITY = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[0]]}{N} \tag{5.2}$$

$$FAKE\_BOT\_EVA = \frac{\sum[\hat{\mathcal{G}}(z, c_m)[1]]}{N} \tag{5.3}$$

$$REAL\_NORMAL\_EST = \frac{\sum[\hat{\mathcal{X}}_{normal_{test}}[2]]}{N} \tag{5.4}$$

$$REAL\_BOT\_EVA = \frac{\sum[\hat{\mathcal{X}}_{botnet_{test}}[1]]}{N} \tag{5.5}$$

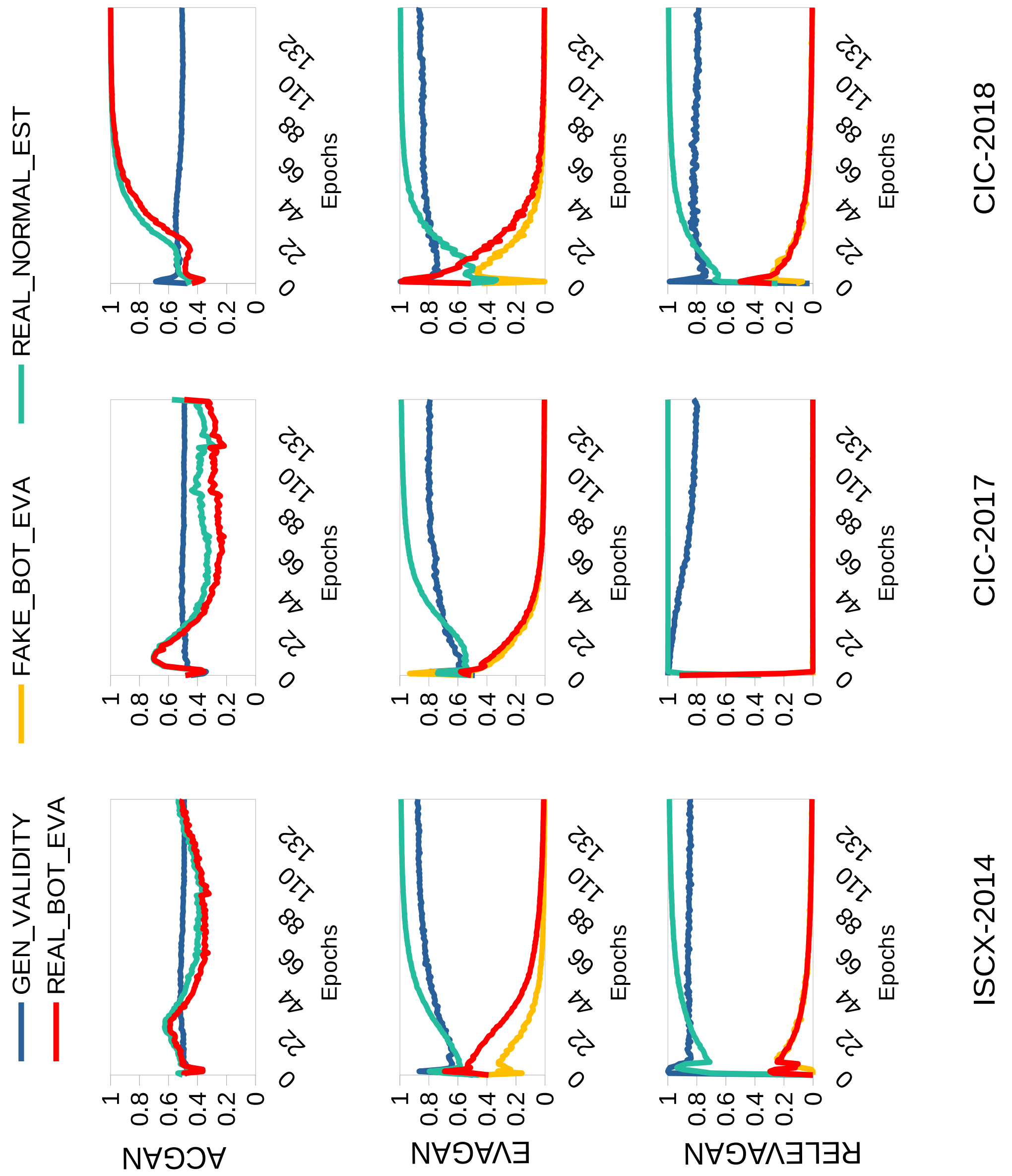


Figure 5.3: The estimations on test data and data generated by the relative GANs generated data

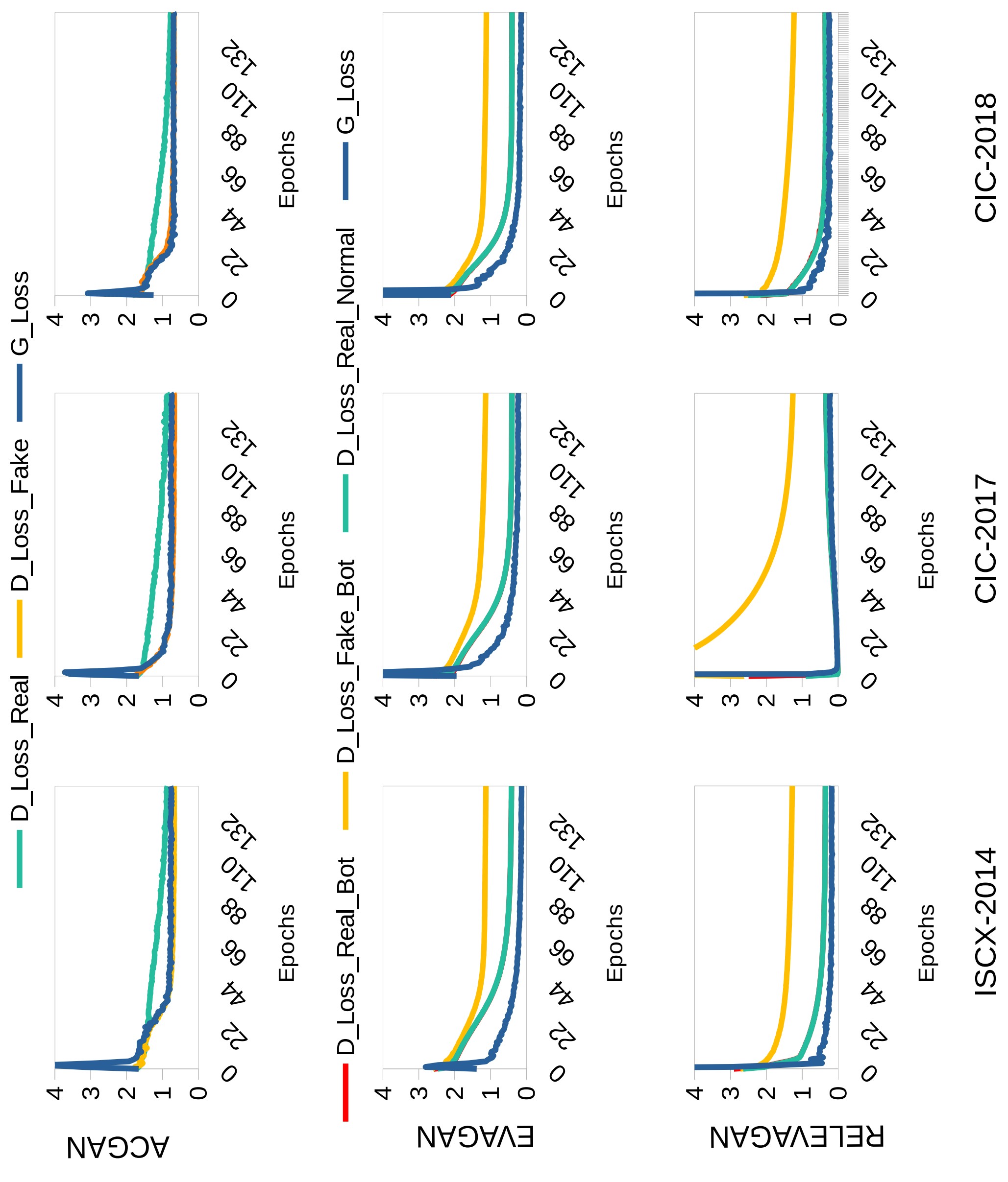


Figure 5.4: The training losses for ACGAN, EVAGAN and RELEVAGAN

5.4.1 Detection Performance

The discussion on the performance comparison of ACGAN with EVAGAN has been mentioned in Chapter 4; however, in Figure 5.3, the estimations for ACGAN have also been included for a clear comparison. The performance of ACGAN in discerning the minority class deteriorates in low data regimes, especially in cybersecurity datasets. On the other hand, $\mathcal{D}$ of EVAGAN discerns the difference between the minority of majority classes. However, regarding RELEVAGAN, the estimations approach the desired values quicker than EVAGAN after a few initial unstable values. The RELEVAGAN was pre-trained for the values coming from the real and generated sources, especially from epochs 0-44. This is the contribution of the DRL attacker part. The most interesting case occurs for the CIC-2017 dataset, which took only a couple of epochs to train itself and prepare for both the minority and majority classes. The role of DRL with EVAGAN demonstrates the value addition, especially for the CIC-2017 dataset. This pattern encourages us to explore further the potential of RELEVAGAN on other datasets, which is left to future work.

The EVAGAN is being used as the base model, which does not require adversarial training of dedicated ML classifiers because $\mathcal{D}$ of EVAGAN itself works as an evasion-aware botnet detector. So RELEVAGAN also does not need the adversarial training; however, the evasions that the DRL attacker produces need to be fed to $\mathcal{D}$. This step imitates the back-propagation step in GAN training.

GEN_VALIDITY also illustrates the early convergence of $\mathcal{G}$, which achieves the Nash equilibrium quickly in RELEVAGAN while $\mathcal{G}$ of EVAGAN is still learning. Since the main objective is to improve the detection performance of the model for botnets in low data regimes, there is no need to let $\mathcal{G}$ train for a more significant number of epochs. For example, if a 100% detection performance is achieved, as in the case of the CIC-2017 dataset, the GAN training can be stoped after a couple of steps.

5.4.2 Stability

Figure 5.4 demonstrates $\mathcal{D}$ and $\mathcal{G}$ losses for ACGAN, EVAGAN and RELEVAGAN. It turns out that the losses tend to converge for all the GANs. The values for RELEVAGAN tend to achieve the lowest point sooner than both EVAGAN and ACGAN. The values for D_Loss_Fake for RELEVAGAN are higher than other GANs. This can be because $\mathcal{D}$ of RELEVAGAN is struggling to discriminate between the evasion generated by the DRL and $\mathcal{G}$. The evasion samples coming from DRL are labelled as 'REAL', and when similar samples arrive from $\mathcal{G}$, the wrong estimation increases the loss. However, the overall detection performance improves because $\mathcal{G}$ now tends to generate the samples within the semantic constraints. This can be due to mode collapse; however, the detection performance is better than EVAGAN, so this factor is disregarded.

5.4.3 DRL Attacker Reward/Evasions

Figure 5.5 shows the number of evasions the DRL attacker generated or the reward collected during the exploration or training phase of RELEVAGAN of the first ten epochs. No evasions happen after epoch number eight. Note that the highest number of evasions was in the case of the CIC-2017 dataset, which gives the best results for estimations in Figure 5.3. A similar pattern is seen in the cases of ISCX-2014 and CIC-2018 datasets, but no evasions for the CIC-2018 dataset were generated after epoch three. If the number of generated evasions by the DRL attacker is correlated with the performance of $\mathcal{D}$, it turns out that there is an inverse relationship between the number of evasions and convergence of the training of RELEVAGAN. Trying the model on other datasets can give more insights into this relationship.

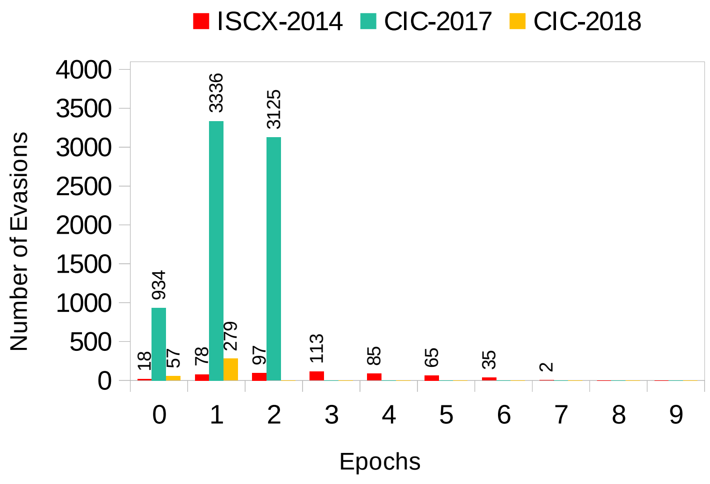


Figure 5.5: RELEVAGAN reward and evasions for the three datasets

5.4.4 Time Complexity

Figure 5.6 shows the training time complexity of ACGAN, EVAGAN and RELEVAGAN for the three datasets. RELEVAGAN training time for 150 epochs is always greater than that of EVAGAN because DRL has its own cost. However, the early convergence achieves the maximum detection performance as manifested in Figure 5.3 for the CIC-2017 dataset. Hence the performance of 100% is achieved within the time way less than taken by EVAGAN, but it turns out to depend on the dataset. As for the case of the ISCX-2014 dataset, the time complexity is even more than ACGAN's training time. Hence, the working towards more datasets is considered for greater insight into the time complexity pattern.

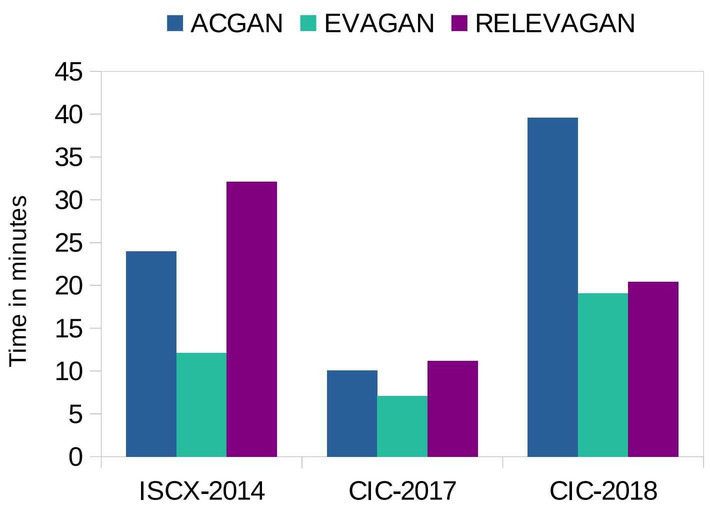


Figure 5.6: Time complexity

5.5 Comparison of RELEVAGAN with Peer Techniques

The RELEVAGAN model is an enhanced version of EVAGAN, dedicated to low data regimes for learning adversarial evasion examples generated during GAN training with the additional support of DRL. So the most suitable existing model for the comparison can be EVAGAN, the details of which have been explained in section 4.2 previously. However, this section mentions peer techniques similar to RELEVAGAN that indirectly address the adversarial evasion problem using DRL while preserving the functionality of the attack operation. Authors in [26, 69, 70] have provided DRL-based model to attack a malware detector by making changes based on a DRL agent output as a result of a reward coming out of the classifier. Similar concept has been applied while designing the DRL attacker in RELEVAGAN. A closer work to RELEVAGAN is [27] that uses the botnet datasets to generate evasion samples using DRL, however the RELEVAGAN is different in a way that it automatically is making the target learn the attack samples being generated from the DRL agent so the adjacent EVAGAN model learns simultaneously during training. This synergy makes the generator training faster. The work done by [25] is also significantly relevant to RELEVAGAN, however they are using adversarial training at the end of the attack generation while RELEVAGAN learns that attacks immediately after its discovery which makes it a proactive approach as compared to the peer work.

5.6 Summary

The semantic-aware adversarial botnet detector is essential for countering modern evasion attacks. In this regard, the detection models must proactively be aware of the possible adversarial perturbations. Researchers employ DRL to generate adversarial evasion examples to preserve the original functionality of the botnet/malware samples. The motivation is to generate samples that could be used later for adversarial training of the ML classifiers. The RELEVAGAN is proposed, which proves to be an adversarial semantic-aware evasion detection model that does not need exclusive adversarial training.

- The discriminator of RELEVAGAN is based on EVAGAN, which did not consider the semantic awareness that RELEVAGAN resolves.
- The same three datasets used in Chapter 4 were employed for the better comparison and coherency.
- The results demonstrate the supremacy of RELEVAGAN to EVAGAN in early convergence of the detection model training.

Testing RELEVAGAN's performance against other cybersecurity datasets is highly encouraged for future research. Few-shot learning could also be tried on the datasets used in this chapter to compare performance.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

Several research works have highlighted the fragility of AI-based detection models. Botnet detectors are no exception when it comes to adversarial evasion attacks. More data is needed to address the issue of data imbalance in research data sets for the botnet. Synthetic oversampling can be used to generate data that can help in improving detection performance. GANs, at the same time, can act as oversamplers and adversarial sample generators to mitigate the effects of evasion attacks. In this thesis, the research began by keeping GANs in mind as the dual panacea of the problems mentioned earlier. However, the research was devising a novel model specialised for botnet detection and other low data regimes. An improved version of the model was then proposed that addressed another perspective of the multifaceted problem in the adversarial domain of AI in low data regimes. The research was named ”Evasion-Aware Botnet Detection using AI”. This research is considered a beginning toward designing a sophisticated botnet detection model, as there is still a long way to go to address the problem of adversarial evasion attacks completely.

Following is the detail of contributions and the future research direction:

6.1.1 BotShot

In Chapter 3, GANs were employed to generate improved quality samples to reduce false negatives or, in other terms, improve the detection of evasion samples. GANs proved to be better than other synthetic oversamplers for botnet traffic generation for all the botnet datasets used in this research. A novel technique called *Botshot* was proposed that improved the detection performance of the GAN-generated adversarial samples that evaded the traditional ML-based classifiers. The general method to utilise GANs to generate quality samples is based on the loss functions of generator and discriminator networks. However, the quality of generated data could be further evaluated using the classifiers under test. The results demonstrated the *Botshot*'s superiority over other synthetic oversamplers in case of a black box attack. The black box test over GAN-generated samples evaded all the classifiers for the three datasets, making the recall score as low as 0%. After the AT, the improvement was nearing 100% for all the classifiers under test. However, in the case of the peer synthetic oversamplers, the improvement was not as significant. The classical GAN-based *Botshot* gave the highest improvement (98.64%) in recall score in the case of the CIC-2017 dataset. The results encouraged further investigation of other GANs and botnet datasets for performance improvement against adversarial evasion samples using the proposed technique.

6.1.2 EVAGAN

In Chapter 4, a novel GAN model called EVAGAN was proposed that efficiently generates adversarial evasions in low data regimes. The model is an enhanced version of the ACGAN. The supremacy of EVAGAN or ACGAN has been demonstrated in the results where for binary class low data regimes, EVAGAN provides 100% detection accuracy as compared to ACGAN and the traditional ML classifiers. EVAGAN acts as an oversampler for adversarial evasion samples and an evasion-aware detection model for botnets in low-data regimes. The scope of EVAGAN is spanned over other domains like computer vision for medical anomaly detection, where data imbalance is a crucial concern. The results for the computer vision data set, MNIST, demonstrate the superiority of EVAGAN over ACGAN in terms of detection performance, stability, and time complexity. At the same time, the qualitative analysis shows that EVAGAN outperforms ACGAN in unbalanced scenarios.

GANs have proved to be very effective in computer vision-based applications. However, it can be inferred from this work that GANs are suitable candidates to address multiple challenges within the cybersecurity domain as well. Significantly, the effects of adversarial evasion attacks can be mitigated proactively using GANs' generated traffic augmentation to the original train sets. The general method to utilise GANs to generate quality samples is based on the loss functions of generator and discriminator networks. However, the quality of generated data could be further evaluated using the classifiers under test.

6.1.3 RELEVAGAN

In Chapter 5, an improved version of EVAGAN was proposed to provide online learning for adversarial and semantic awareness of the generated samples. A sophisticated botnet detection model can use deep reinforcement learning to generate adversarial evasion samples to preserve the original functionality of the botnet/malware samples. When coupled with EVAGAN's discriminator model, deep reinforcement learning can help the generator of EVAGAN to confine the exploratory space within semantic limits. The results show that RELEVAGAN can help in early convergence for botnet detection in low data regimes by incorporating deep reinforcement learning compared to EVAGAN.

6.2 Threat Model

The schemes proposed in this research are based on the evasion samples generated from the public datasets. As per the taxonomies of [46, 105, 106] the proposed schemes are prone to attacks where the attacker has access to the training set and selected features. However, the chance of a model trained on a known dataset being fooled by the samples generated from the same dataset by an adversary is also based on the choice of the internal parameters. The proposed work can be transformed into a black box model where the internals of the architecture can be kept confidential without giving access to the oracle to the attacker.

Regarding the adversarial manipulation depth, the universal validation of the model against all types of known attacks is linearly proportional to the number of unique samples input for training. As the deep learning has the potential to absorb more training data with the increase in the model parameters, it is expected to be an open problem to solve. So the more the model has the knowledge about the new unique features and better optimizing techniques, it has the potential of being dataset agnostic given the attack semantics of the generated samples remain within the functional boundaries. However, the possibilities of samples being in evasion space are intractable for the unknown attacks. A GAN based evasion generation is a semi-supervised process however, with the use of DRL, we can explore unseen samples and keep on training our model but again around the known semantic/functionalities of the attacks samples.

6.3 Future Directions

Further research directions can significantly contribute toward an ultimate design of an evasion-aware botnet detection model. The landscapes can be further spread over other domains as well.

6.3.1 Novel Features Exploration

Further exploration of modern botnet behaviours is cardinal for improving the detection performance of botnet traffic. Since the botnets are evolving with time by exploiting the vulnerabilities in the existing AI system, a watch on new features is needed.

6.3.2 Use of Modern GAN models

GANs have proved to be effective in adversarial evasion detection improvement. The models proposed in this research are based on primitive GAN models. There is considerable room for incorporating modern concepts in GAN research that can further enhance the proposed research.

6.3.3 Traffic Validation

The generated traffic from the proposed GAN models like EVAGAN and especially RELEVAGAN needs to be validated by relaying the RESTful API over the internet. The model could be further enhanced by discarding the invalid traffic samples and back-propagating the loss to the proposed models in this research. RELEVAGAN helps in preserving the semantics of the botnet samples. However, the generated samples in this research were not tested. Future researchers can first make the TCP/IP packets using the generated feature vectors from the generators of both EVAGAN and RELEVAGAN and then relay the traffic on the internet.

6.3.4 Multiclass Landscape

Both EVAGAN and RELEVAGAN have been devised based on binary class datasets. The multiclass design of the proposed models is an exciting research direction where future researchers can expand the discriminator model to output multiple class probabilities instead of just two.

6.3.5 Comparison with Few-shot and Zero-shot Learning

Few-shot or zero-shot learning could be compared with the proposed research keeping the low data regime factor in mind where the anomaly samples are scarce. Few or zero-shot learning could be a strong competitor to EVAGAN and RELEVAGAN for low data regimes.

6.3.6 Use of other Cybersecurity Datasets

The work can be expanded further using other cybersecurity data sets in the cybersecurity domain for IoT and cyber-physical systems.

6.3.7 Testing the models on Computer Vision Datasets

Only one computer vision data set was used in this research to show the power of EVAGAN; however, testing the models on other computer vision data sets where data is imbalanced, especially the medical anomaly detection, is highly recommended.

6.3.8 Testing on Single Board Computers

The proposed solution can be transformed into a portable product that could be used as a filter for home users to avoid adversarial evasion attacks on the internet. A possible direction for the embedded software researcher working in AI-based cybersecurity could be using NVIDIA Jetson single-board computers.

Glossary

$\mathcal{D}$	Discriminator. 11–15, 18
$\mathcal{G}$	Generator. 5, 11–15, 18, 23
ACGAN	Auxiliary Classifier Generative Adversarial Network. iv, 59
AT	Adversarial Training. iv
DNNs	Deep Neural Networks. 1, 21
DRL	Deep Reinforcement Learning. iv, v, 1, 5, 9, 20
EVAGAN	Evasion Generative Adversarial Network. iv
GAN	Generative Adversarial Network. iv, 2, 11
IDSs	Intrusion Detection Systems. 1
ML	Machine Learning. iv
RELEVAGAN	Deep Reinforcement Learning based Generative Adversarial Network. iv, v

SMOTE	Synthetic Minority Oversampling TEchnique. 17, 18, 37–40, 51, 58

References

[1] Partha Pratim Kundu, Tram Truong-Huu, Ling Chen, Luying Zhou, and Sin G Teo. Detection and classification of botnet traffic using deep learning with model explanation. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[2] Rizwan Hamid Randhawa, Abdul Hameed, and Adnan Noor Mian. Energy efficient cross-layer approach for object security of coap for iot devices. *Ad Hoc Networks*, 92:101761, 2019.

[3] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C Suh, Ikkyun Kim, and Kuinam J Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, pages 1–13, 2017.

[4] Samaneh Mahdavifar and Ali A Ghorbani. Application of deep learning to cybersecurity: A survey. *Neurocomputing*, 347:149–176, 2019.

[5] Mohammad Alauthman, Nauman Aslam, Mouhammd Al-Kasassbeh, Suleman Khan, Ahmad Al-Qerem, and Kim-Kwang Raymond Choo. An efficient reinforcement learning-based botnet detection approach. *Journal of Network and Computer Applications*, 150: 102479, 2020.

[6] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1):25–45, 2021.

[7] Andrew McCarthy, Essam Ghadafi, Panagiotis Andriotis, and Phil Legg. Functionality-preserving adversarial machine learning for robust classification in cybersecurity and intrusion detection domains: A survey. *Journal of Cybersecurity and Privacy*, 2(1):154–190, 2022.

[8] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*, pages 582–597. IEEE, 2016.

[9] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[10] Rupam Kumar Sharma, Hemanta Kr Kalita, and Biju Issac. Are machine learning based intrusion detection system always secure? an insight into tampered learning. *Journal of Intelligent & Fuzzy Systems*, pages 1–17, 2018.

[11] György Kovács. Smote-variants: A python implementation of 85 minority oversampling techniques. *Neurocomputing*, 366:352–354, 2019.

[12] György Kovács. An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing*, 83:105662, 2019.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[14] Justin Engelmann and Stefan Lessmann. Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications*, 174:114582,

2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.114582. URL https://www.sciencedirect.com/science/article/pii/S0957417421000233.

[15] Chuanlong Yin, Yuefei Zhu, Shengli Liu, Jinlong Fei, and Hetong Zhang. An enhancing framework for botnet detection using generative adversarial networks. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 228–234. IEEE, 2018.

[16] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[17] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 2018.

[18] Konda Reddy Mopuri, Utkarsh Ojha, Utsav Garg, and R Venkatesh Babu. Nag: Network for adversary generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 742–751, 2018.

[19] Muhammad Usama, Muhammad Asim, Siddique Latif, Junaid Qadir, et al. Generative adversarial networks for launching and thwarting adversarial attacks on network intrusion detection systems. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 78–83. IEEE, 2019.

[20] Yang Xin, Lingshuang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, Haixia Hou, and Chunhua Wang. Machine learning and deep learning methods for cybersecurity. *Ieee access*, 6:35365–35381, 2018.

[21] Daniel S Berman, Anna L Buczak, Jeffrey S Chavis, and Cherita L Corbett. A survey of deep learning methods for cyber security. *Information*, 10(4):122, 2019.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[23] Ruslan Salakhutdinov. Learning deep generative models. *Annual Review of Statistics and Its Application*, 2:361–385, 2015.

[24] Chuanlong Yin, Yuefei Zhu, Shengli Liu, Jinlong Fei, and Hetong Zhang. Enhancing network intrusion detection classifiers using supervised adversarial training. *The Journal of Supercomputing*, pages 1–30, 2019.

[25] Giovanni Apruzzese, Mauro Andreolini, Mirco Marchetti, Andrea Venturi, and Michele Colajanni. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management*, 17(4):1975–1987, 2020.

[26] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[27] Di Wu, Binxing Fang, Junnan Wang, Qixu Liu, and Xiang Cui. Evading machine learning botnet detection models via deep reinforcement learning. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[28] Hanan Hindy, David Brosset, Ethan Bayne, Amar Seeam, Christos Tachtatzis, Robert Atkinson, and Xavier Bellekens. A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. *arXiv preprint arXiv:1806.03517*, 2018.

[29] Paulo Angelo Alves Resende and André Costa Drummond. A survey of random forest based methods for intrusion detection systems. *ACM Computing Surveys (CSUR)*, 51(3): 48, 2018.

[30] Antonia Nisioti, Alexios Mylonas, Paul D Yoo, and Vasilios Katos. From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods. *IEEE Communications Surveys & Tutorials*, 20(4):3369–3388, 2018.

[31] David Santana, Shan Suthaharan, and Somya Mohanty. What we learn from learning-understanding capabilities and limitations of machine learning in botnet attacks. *arXiv preprint arXiv:1805.01333*, 2018.

[32] Ahmad Azab, Mamoun Alazab, and Mahdi Aiash. Machine learning based botnet identification traffic. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 1788–1794. IEEE, 2016.

[33] Ankit Bansal and Sudipta Mahapatra. A comparative analysis of machine learning techniques for botnet detection. In *Proceedings of the 10th International Conference on Security of Information and Networks*, SIN '17, pages 91–98, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5303-8. doi: 10.1145/3136825.3136874. URL http://doi.acm.org/10.1145/3136825.3136874.

[34] Joshua Ojo Nehinbe. A critical evaluation of datasets for investigating idss and ipss researches. In *Cybernetic Intelligent Systems (CIS), 2011 IEEE 10th International Conference on*, pages 92–97. IEEE, 2011.

[35] Nour Moustafa, Jiankun Hu, and Jill Slay. A holistic review of network anomaly detection systems: A comprehensive survey. *Journal of Network and Computer Applications*, 128: 33–55, 2019.

[36] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4487–4492. IEEE, 2013.

[37] Jana Uramová, Pavel Scgeč, Marek Moravčík, Jozef Papán, Martin Kontšek, and Jakub Hrabovskỳ. Infrastructure for generating new ids dataset. In *2018 16th International*

Conference on Emerging eLearning Technologies and Applications (ICETA), pages 603–610. IEEE, 2018.

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

[39] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, 2013.

[40] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[41] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *2011 Ninth annual international conference on privacy, security and trust*, pages 174–180. IEEE, 2011.

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

[43] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[44] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[45] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. *arXiv preprint arXiv:1804.00097*, 2018.

[46] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[47] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[48] Sicong Zhang, Xiaoyao Xie, and Yang Xu. A brute-force black-box method to attack machine learning-based systems in cybersecurity. *IEEE Access*, 8:128250–128263, 2020.

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

[50] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[51] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

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

[53] Rizwan Hamid Randhawa, Nauman Aslam, Mohammad Alauthman, Husnain Rafiq, and Frank Comeau. Security hardening of botnet detectors using generative adversarial networks. *IEEE Access*, 9:78276–78292, 2021.

[54] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR, 2017.

[55] Giovanni Apruzzese and Michele Colajanni. Evading botnet detectors based on flows and random forest with adversarial samples. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[56] Nuno Martins, José Magalhães Cruz, Tiago Cruz, and Pedro Henriques Abreu. Adversarial machine learning applied to intrusion and malware scenarios: a systematic review. *IEEE Access*, 8:35403–35419, 2020.

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

[58] Hyrum S Anderson, Jonathan Woodbridge, and Bobby Filar. Deepdga: Adversarially-tuned domain generation and detection. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 13–21. ACM, 2016.

[59] Gernot Vormayr, Tanja Zseby, and Joachim Fabini. Botnet communication patterns. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, 19(4):2768–2796, 2017.

[60] Yannic Kilcher and Thomas Hofmann. The best defense is a good offense: Countering black box attacks by predicting slightly wrong labels. *arXiv preprint arXiv:1711.05475*, 2017.

[61] Phan The Duy, Nghi Hoang Khoa, Anh Gia-Tuan Nguyen, Van-Hau Pham, et al. Digfupas: Deceive ids with gan and function-preserving on adversarial samples in sdn-enabled networks. *Computers & Security*, 109:102367, 2021.

[62] Qiumei Cheng, Shiying Zhou, Yi Shen, Dezhang Kong, and Chunming Wu. Packet-level adversarial network traffic crafting using sequence generative adversarial networks. *arXiv preprint arXiv:2103.04794*, 2021.

[63] Andrea Venturi, Giovanni Apruzzese, Mauro Andreolini, Michele Colajanni, and Mirco Marchetti. Drelab-deep reinforcement learning adversarial botnet: A benchmark dataset

for adversarial attacks against botnet intrusion detection systems. *Data in Brief*, 34: 106631, 2021.

[64] Shuokang Huang and Kai Lei. Igan-ids: An imbalanced generative adversarial network towards intrusion detection system in ad-hoc networks. *Ad Hoc Networks*, 105:102177, 2020.

[65] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 376–385. IEEE, 2020.

[66] Markus Ring, Alexander Dallmann, Dieter Landes, and Andreas Hotho. Ip2vec: Learning similarities between ip addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 657–666, 2017. doi: 10.1109/ICDMW.2017.93.

[67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518 (7540):529–533, 2015.

[68] Mohammad Mansour Alauthman. *An efficient approach to online bot detection based on a reinforcement learning technique*. University of Northumbria at Newcastle (United Kingdom), 2016.

[69] Zhengyang Mao, Zhiyang Fang, Meijin Li, and Yang Fan. Evaderl: Evading pdf malware classifiers with deep reinforcement learning. *Security and Communication Networks*, 2022, 2022.

[70] Zhiyang Fang, Junfeng Wang, Boya Li, Siqi Wu, Yingjie Zhou, and Haiying Huang. Evading anti-malware engines with deep reinforcement learning. *IEEE Access*, 7:48867–48879, 2019.

[71] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, pages 1–31, 2020.

[72] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*, pages 247–255. IEEE, 2014.

[73] S Garcia. Malware capture facility project. *cvut*, 2013.

[74] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *Advances in neural information processing systems*, pages 700–709, 2018.

[75] Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

[76] Michael Moret, Lukas Friedrich, Francesca Grisoni, Daniel Merk, and Gisbert Schneider. Generative molecular design in low data regimes. *Nature Machine Intelligence*, 2(3): 171–180, 2020.

[77] Ly Vu and Quang Uy Nguyen. Handling imbalanced data in intrusion detection systems using generative adversarial networks. *Journal of Research and Development on Information and Communication Technology*, 2020(1):1–13, 2020.

[78] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[79] Vadim Sushko, Jurgen Gall, and Anna Khoreva. One-shot gan: Learning to generate samples from single images and videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2596–2600, 2021.

[80] Hung Ba. Improving detection of credit card fraudulent transactions using generative adversarial networks. *arXiv preprint arXiv:1907.03355v1*, 2019.

[81] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[82] Tim Merino, Matt Stillwell, Mark Steele, Max Coplan, Jon Patton, Alexander Stoyanov, and Lin Deng. Expansion of cyber attack data from unbalanced datasets using generative adversarial networks. In *International Conference on Software Engineering Research, Management and Applications*, pages 131–145. Springer, 2019.

[83] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In João Gama, Tianrui Li, Yang Yu, Enhong Chen, Yu Zheng, and Fei Teng, editors, *Advances in Knowledge Discovery and Data Mining*, pages 79–91, Cham, 2022. Springer International Publishing. ISBN 978-3-031-05981-0.

[84] He Zhang, Xingrui Yu, Peng Ren, Chunbo Luo, and Geyong Min. Deep adversarial learning in intrusion detection: A data augmentation enhanced framework. *arXiv preprint arXiv:1901.07949*, 2019.

[85] Qiao Yan, Mingde Wang, Wenyao Huang, Xupeng Luo, and F Richard Yu. Automatically synthesizing dos attack traces using generative adversarial networks. *International Journal of Machine Learning and Cybernetics*, 10(12):3387–3396, 2019.

[86] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[87] Yu Guo, Gang Xiong, Zhen Li, Junzheng Shi, Mingxin Cui, and Gaopeng Gou. Ta-gan: Gan based traffic augmentation for imbalanced network traffic classification. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[88] JooHwa Lee and KeeHyun Park. Ae-cgan model based high performance network intrusion detection system. *Applied Sciences*, 9(20):4221, 2019.

[89] Rana Abou Khamis and Ashraf Matrawy. Evaluation of adversarial training on different types of neural networks in deep learning-based idss. In *2020 international symposium on networks, computers and communications (ISNCC)*, pages 1–6. IEEE, 2020.

[90] Maged Abdelaty, Sandra Scott-Hayward, Roberto Doriguzzi-Corin, and Domenico Siracusa. Gadot: Gan-based adversarial training for robust ddos attack detection. In *Ninth IEEE Conference on Communications and Network Security (IEEE CNS 2021)*, 2021.

[91] Cao Phan Xuan Qui, Dang Hong Quang, Phan The Duy, Van-Hau Pham, et al. Strengthening ids against evasion attacks with gan-based adversarial samples in sdn-enabled network. In *2021 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 1–6. IEEE, 2021.

[92] Ulya Sabeel, Shahram Shah Heydari, Khalid Elgazzar, and Khalil El-Khatib. Cvae-an: Atypical attack flow detection using incremental adversarial learning. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2021.

[93] Wei Li, Xiang Zhong, Haidong Shao, Baoping Cai, and Xingkai Yang. Multi-mode data augmentation and fault diagnosis of rotating machinery using modified acgan designed with new framework. *Advanced Engineering Informatics*, 52:101552, 2022.

[94] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[95] Xin Yi, Ekta Walia, and Paul Babyn. Generative adversarial network in medical imaging: A review. *Medical image analysis*, 58:101552, 2019.

[96] Abdul Waheed, Muskan Goyal, Deepak Gupta, Ashish Khanna, Fadi Al-Turjman, and Plácido Rogerio Pinheiro. Covidgan: data augmentation using auxiliary classifier gan for improved covid-19 detection. *Ieee Access*, 8:91916–91923, 2020.

[97] Vanchinbal Chinbat and Seung-Hwan Bae. Ga3n: Generative adversarial autoaugment network. *Pattern Recognition*, page 108637, 2022.

[98] Wonkeun Jo and Dongil Kim. Obgan: Minority oversampling near borderline with generative adversarial networks. *Expert Systems with Applications*, page 116694, 2022.

[99] Aswathy Madhu and Suresh K. Envgan: a gan-based augmentation to improve environmental sound classification. *Artificial Intelligence Review*, Feb 2022. ISSN 1573-7462. doi: 10.1007/s10462-022-10153-0. URL https://doi.org/10.1007/s10462-022-10153-0.

[100] Sam Grierson, Craig Thomson, Pavlos Papadopoulos, and Bill Buchanan. Min-max training: Adversarially robust learning models for network intrusion detection systems. In *2021 14th International Conference on Security of Information and Networks (SIN)*, volume 1, pages 1–8. IEEE, 2021.

[101] Shuang Zhao, Jing Li, Jianmin Wang, Zhao Zhang, Lin Zhu, and Yong Zhang. attackgan: Adversarial attack against black-box ids using generative adversarial networks. *Procedia Computer Science*, 187:128–133, 2021.

[102] Sungho Suh, Haebom Lee, Paul Lukowicz, and Yong Oh Lee. Cegan: Classification enhancement generative adversarial networks for unraveling data imbalance problems. *Neural Networks*, 133:69–86, 2021.

[103] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[104] Rizwan Hamid Randhawa, Nauman Aslam, Muhammad Alauthman, and Husnain Rafiq. Evagan: Evasion generative adversarial network for low data regimes. *arXiv preprint arXiv:2109.08026*, 2021.

[105] Pavel Laskov et al. Practical evasion of a learning-based classifier: A case study. In *2014 IEEE symposium on security and privacy*, pages 197–211. IEEE, 2014.

[106] Giovanni Apruzzese, Mauro Andreolini, Luca Ferretti, Mirco Marchetti, and Michele Colajanni. Modeling realistic adversarial attacks against network intrusion detection systems. *Digital Threats: Research and Practice (DTRAP)*, 3(3):1–19, 2022.