SOPHON IDS: MITIGATING THE EFFECTIVENESS OF GAN-BASED ATTACKS VIA TAILORED MISINFORMATION

by

Zihao Liu

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2023

To my beloved mother, whose selfless love and unwavering support have been my guiding light throughout the past years. Your belief in me has given me the strength to overcome every challenge. I am also deeply grateful to my late father, whose memory continues to inspire me. I carry his legacy with me and strive to make him proud. I want to extend my heartfelt thanks to my roommate and dear friend, Jimmy. Your companionship during the two years abroad has made the journey all the more memorable and enjoyable.

Table of Contents

List of Tables

List of Figures

Abstract

In the realm of safeguarding real networks against malicious activities, Intrusion Detection System (IDS) assumes a critical role. Despite the advancements brought about by machine learning and deep learning in enhancing its performance, IDS is still vulnerable to adversarial samples stemming from Generative Adversarial Network (GAN). IDSGAN is one of the most effective attacking schemes that are based on GAN. In this thesis, we propose a novel anti-IDSGAN method, Sophon IDS (S-IDS), which transmits deceptive information to IDSGAN-based attackers in order to disrupt their training process, ultimately mitigating the effectiveness of IDSGAN-based attacks. Technically, the deceptive information is generated by flipping the benign/malicious labels of network flows. In our research, we compared the performance of a series of label-flipping strategies. Our experimental results indicate that the 'DVT-U-01' strategy leads to the highest detection rate for IDSGAN-based network flows. In addition, we found that RNN-based S-IDS outperforms LR-based and MLP-based S-IDS.

Acknowledgements

I am deeply grateful to my supervisor, Dr. Qiang Ye, whose guidance and support have been instrumental in shaping the trajectory of my research. Dr. Ye's unique insights and invaluable suggestions have consistently enriched the quality of my work and inspired new avenues of exploration. Moreover, his willingness to provide external assistance resources has been a catalyst in achieving the final results of this study. I would also like to extend my heartfelt thanks to Frank for his kindness and patience in sharing his research ideas and experiences from his thesis paper. His generous guidance has proven to be a tremendous asset, significantly aiding me in developing my own research based on the foundation of his accomplished work. Also, I want to thank all the colleagues and professors sharing their ideas and advice to me.

Chapter 1

Introduction

1.1 Intrusion Detection System

In the real world, network systems are vulnerable to massive attacks, such as denial of service (DoS) and malware. To prevent these attacks, many strategies are employed, and the intrusion detection system (IDS) is one of them. In most cases, IDS functions as software that monitors traffic flows. Once a flow is detected as malicious, the remaining flow packets will not be delivered to the server, ensuring the server can still work properly despite receiving only a few packets. As the volume of network traffic expands rapidly, the demand for IDS increases as well.

The first-generation IDS detects threats based on signatures, hence the name signature-based IDS. In cybersecurity, a signature refers to a specific feature of a traffic flow. The IDS stores all known malicious traffic signatures in its database. If the IDS finds a match between the signature of a new traffic flow and any in its database, it will reject the flow. However, the IDS has to update its database frequently to detect all the malicious traffic flows. With the increasing types of attacks, it may struggle to identify new attacks with utmost precision.

Another type of IDS is the anomaly-based IDS. Unlike the signature-based IDS, it identifies normal traffic by following certain rules or employing heuristic methods. If the traffic flow cannot be identified, it will be marked as anomalous, and all related activities will be denied. Also, it does not rely on highly specific signatures or require frequent updates.

Due to the rapidly increasing demands of modern network infrastructures, the flows among them have become more frequent. Additionally, as the complexity and diversity of these networks continue to grow, their vulnerabilities also increase. The traditional signature-based IDS and human-updated anomaly-based IDS are no longer able to maintain high performance. As a result, researchers are turning their attention to machine learning (ML) and deep learning (DL) technologies as they can find the

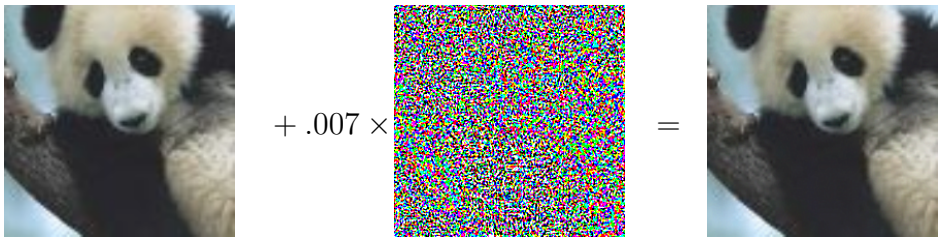

Figure 1.1: An example of using FGSM to generate adversarial sample based on GoogLeNet [1] [2]. Only a tiny well-designed noise can influence the prediction

regulations from the big dataset and ultimately build the models with the impressive performance.

1.2 Adversarial Samples for Machine Learning

While ML models can achieve impressive performance in classification tasks with large datasets, achieving one hundred percent accuracy remains elusive. In certain cases, these models may even perform worse than humans. Consequently, some researchers have recognized the potential to create artificial samples to exploit vulnerabilities in the model's defences.

Adversarial samples are one example of such artificially crafted data points. They do not originate from randomness but are derived from existing samples. By incorporating well-structured noise into the samples, their predictions differ from what the AI model would typically predict. For instance, the Fast Gradient Sign Method (FGSM) is a well-known framework used to generate such adversarial samples [1]. FGSM leverages the gradients of the AI model to compute another vector. By applying only a small perturbation using this vector to the sample, the prediction can significantly deviate from the original, as illustrated in Figure 1.1. Therefore, understanding and addressing these adversarial vulnerabilities in ML models are crucial for enhancing their robustness and real-world applicability.

Defending models can be categorized into two groups based on the level of information required: white-box models and black-box models. White-box models are

completely transparent to external users, allowing access to all operations and returned values. However, this transparency also makes them vulnerable to attacks like FGSM, which relies on accessing the model's gradients. On the other hand, black-box models only reveal the results of sample inputs, providing limited information compared to white-box models. As a result, generating efficient adversarial samples to compromise black-box models becomes exceptionally challenging due to the scarcity of detailed information.

1.3 Generative Adversarial Networks and IDS

Among all the attacks related with adversarial samples, the Generative Adversarial Network (GAN) has earned significant attention in recent years [3]. GAN describes a competition with two players: generator and discriminator. The task for the generator is to generate the samples that the discriminator is not able to tell whether they are from the original dataset or the generated dataset. For each round, both the generator and discriminator can increase their performance thanks to the information from the opponent. In the end, given a random noise, the generator is able to generate a new sample that the discriminator cannot give a clear prediction. In other words, the generator can establish a brand new dataset which approximates the original one. After the GAN is introduced, many new types of GAN models are raised [4][5][6].

Reserach on applying GANs in Intrusion Detection System (IDS) primarily falls into two categories. The first involves using GAN to generate the adversarial samples to attack the IDS. Attackers firstly use the existing samples to probe the IDS and then build the dataset. Thereafter, a GAN-based framework is used to train the generator and discriminator repeatedly based on the dataset. While the second category works in the opposite way. It takes use of the adversarial samples from the GAN-based attacks. These samples operate as the data augmentation to increase the detection rate against the adversarial malicious samples.

In fact, these two types of efforts have distinct goals, leading to an ongoing cycle of competition. Attackers continually devise new adversarial malicious samples, while defenders can leverage these samples to fortify their IDSs. No matter which side gains improvement, the other side could find the approach to counter it. Therefore, as only using GAN as the augmentation is not a efficient method in a long term, we expect

to explore a new strategy to defend the GAN-based adversarial samples.

1.4 Major Contributions

In our study, we present a novel framework named Sophon IDS (S-IDS), designed to counter adversarial malicious samples generated by GAN-based attackers. Unlike other strategies, S-IDS aims for providing the misleading information to the GAN-based attacks during the training process. It is expected to lead to the failure of generating effective adversarial samples. The contribution of this paper are summarized as follows.

Firstly, we propose the framework of S-IDS. Through an in-depth analysis of IDS-GAN, an widely-known GAN-based schemes, we find that misleading information can influence GAN model training. As GAN-based attacks largely rely on the information from the black-box IDS, the deceived information has the negative impact on model training. Thus, when the training is finished, GAN-based attackers struggle to create effective adversarial malicious samples, thus failing to deceive the original IDS.

Secondly, we devise an algorithm for training S-IDS, leveraging a well-trained IDS by altering the labels of selected samples. To explore all possible scenarios, we configure several parameters that influence the flipping process. We design two functions to calculate the distance value of a sample and a flag to decide the priority of these samples. Meanwhile, we control the number and type of the selected samples by introducing the noise ratio and another side flag. With the combination of these parameters, we then apply the algorithm for selecting the samples and train our S-IDS.

Finally, we carry out simulations to evaluate the performance of S-IDS. We consider three ML models: Multilayer Perceptron (MLP), Recurrent Neural Network (RNN), and Logistic Regression (LR). Additionally, we develop an IDSGAN framework to serve as the attacker. We evaluate the detection rates of the original IDS and several variants of S-IDS based on different parameter configurations. Our experimental results indicate that S-IDS consistently outperforms the original IDS in countering IDSGAN attacks. Notably, the RNN-based S-IDS employing the 'DVT-U-01' strategy with $\omega = 0.8$ demonstrates the best performance.

1.5 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we present some related techonologies regarding the machine learning and deep learning, the history of IDS development, several GAN models with their applications and the adversarial attacks against the IDS. In Chapter 3, we introduce the S-IDS, including the theoretical analysis, essential assumptions and the algorithm behind. In Chapter 4, we will present the testbed of our evaluation and the preprocession of the dataset CICIDS2017. Then, we show the experiment results comparing S-IDSs under different configurations and IDS using the MLP, RNN and LR models. In Chapter 5, we make a conclusion of the whole thesis paper. Moreover, we present some interesting findings during the whole evaluation and give some ideas for the future research.

Chapter 2

Related Work

2.1 Machine Learning and Deep Learning

Machine learning (ML) is a part of artifical intelligence (AI) research that focuses on algorithms. By training on datasets, the ML model can imitate human actions and make predictions for new samples. Depending on whether labels for the dataset are required, ML algorithms are divided into two categories: supervised learning and unsupervised learning.

In supervised learning, the algorithm uses labeled datasets to train the model for accurate data classification or outcome prediction. Based on the specific tasks of classification and value prediction, supervised learning models are further categorized into classification models and regression models. Common supervised learning algorithms include Logistic Regression (LR), Support Vector Machines (SVM), Naive Bayes (NB), Decision Tree (DT) and Neural Networks (NN). On the other hand, unsupervised learning algorithms do not require labeled datasets; instead, they discover patterns within the dataset. They are particularly useful when clear patterns are not readily apparent. Therefore, these algorithms primarily focus on solving clustering problems. Traditional clustering algorithms include k-means clustering and hierarchical clustering.

In practical scenarios, annotating datasets with labels demands a substantial investment of time and resources. Due to the scarcity of labeled samples and the abundance of unlabeled ones, researchers have delved into the concept of semi-supervised learning (SSL) as a potential solution. The primary objective of SSL is to enhance accuracy when compared to algorithms that rely solely on the labeled data. Nevertheless, for SSL to yield favorable outcomes, the dataset must adhere to critical assumptions, including the self-training assumption and clustering assumption. Notable SSL algorithms encompass co-training and self-learning techniques.

While ML models have exhibited commendable performance, they face a challenge of limited model size as the volume of data grows substantially. NN models offer a potential solution in this context, as they can significantly scale by increasing the number of neurons or layers. Consequently, researchers are directing their focus towards NN to address this concern. Within the realm of deep learning (DL) models, diverse features exist across various layers, propagation algorithms, and neuron designs. These factors contribute to the versatility and effectiveness of deep learning models.

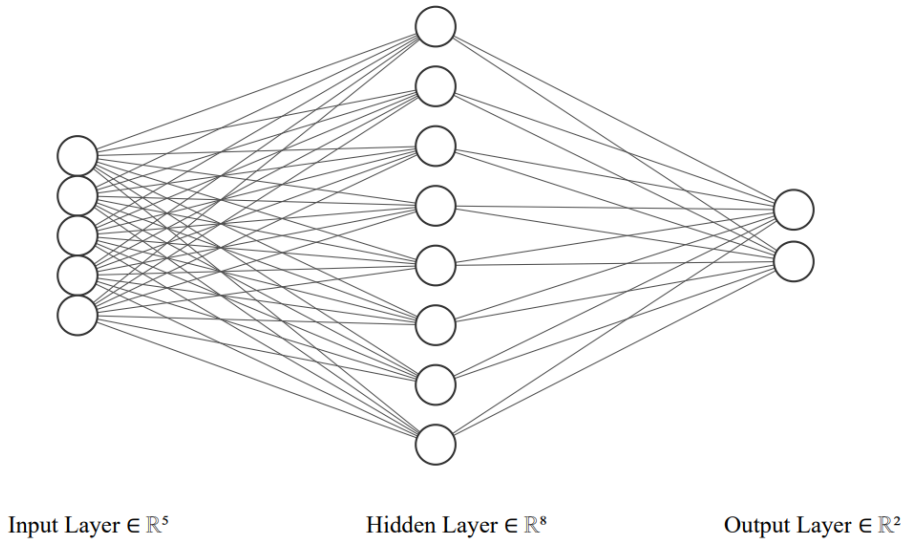

Figure 2.1: An example of Multilayer Perceptron (MLP) network. It consists of input layer, hidden layers and output layer. Each layer is fully connected to the nearby layers.

The simplest deep learning model is the Multilayer Perceptron (MLP), extensively employed for classification and regression tasks. As depicted in Figure 2.1, the MLP comprises three types of layers: the input layer, hidden layers, and the output layer, with each layer being fully connected to its adjacent layer. The input and output layers facilitate communication with the user. Typically, the number of neurons in the input layer equals the number of features in the dataset being processed. In the output layer, the number of neurons varies based on the task at hand. For regression problems, the output layer generates specific values, and the number of neurons corresponds to these values. Conversely, in classification tasks, the output layer produces probabilities for each class, and thus, its size aligns with the number

of unique classes in the dataset. For binary classification, some researchers opt for a single neuron, using a threshold to determine the final class.

The performance of the hidden layers significantly hinges on their design. Users can adjust the number of hidden layers or the number of neurons within them. However, as the depth and width of the hidden layers increase, so does the number of parameters in the model. A surplus of parameters can lead to overfitting, making it crucial to strike a balance and appropriately configure the MLP model for optimal results.

The Recurrent Neural Network (RNN) is another type of deep learning model that includes a special recurrent layer, as illustrated in Figure 2.2. In the recurrent layer, the output is determined by the equations shown in Eq. (2.1) and (2.2). Here, V, U, and W represent the parameters connecting the layers, while f and g are the activation functions. The outputs H_t and H_{t-1} correspond to the hidden layer's output at time t and the previous time step $t-1$, respectively. As we can observe, the current hidden layer output H_t is influenced not only by the current input but also by the previous output H_{t-1}. This characteristic enables the RNN to leverage past information to generate subsequent results, making it suitable for problems where the input order is critical.

The capacity of RNN to retain information about the input sequence's order makes it particularly effective in various natural language processing (NLP) tasks. In NLP, the prediction of a sentence often relies not only on the words used and their frequencies but also on the sequence in which they appear. As a result, RNN is extensively employed in NLP research. Building upon RNN designs, other models like Long Short-Term Memory (LSTM) have been developed by researchers and widely applied to NLP tasks due to their ability to capture long-term dependencies effectively.

$$O_t = g(V * H_t) \tag{2.1}$$

$$H_t = f(U * I_t + R * H_{t-1}) \tag{2.2}$$

The Convolutional Neural Network (CNN) model is another type of deep learning model, and its major specialty lies in its convolutional layer. This layer performs convolutional calculations on higher-dimensional inputs, effectively reducing their dimensions. The primary objective of this process is to strengthen the connections

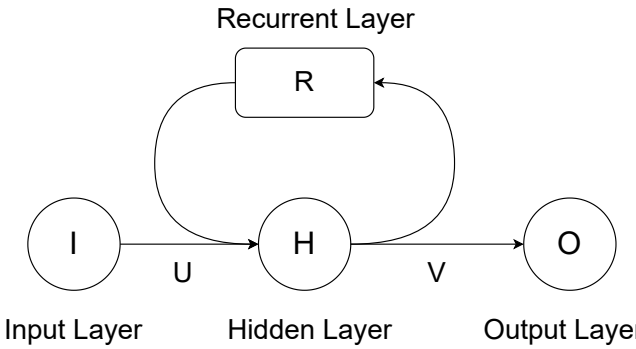

Figure 2.2: The simple design of recurrent neural network. The output of hidden layer also relies on the previous outputs.

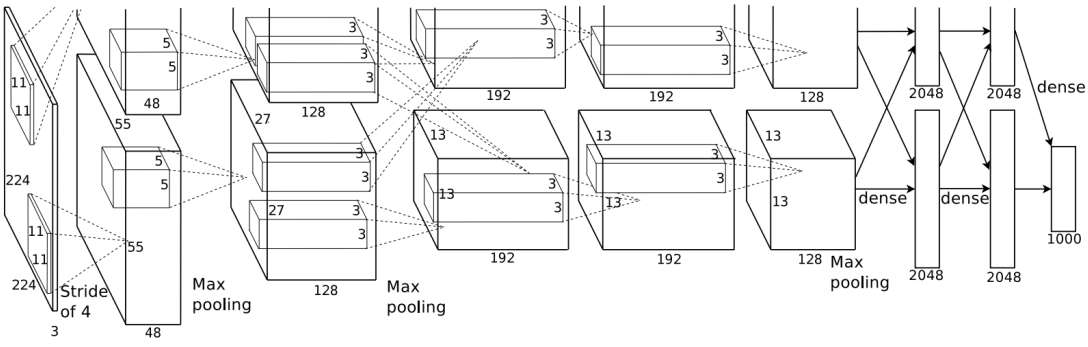

Figure 2.3: The design of the AlexNet. The convoluntional layer is between the first two blocks.

between a value and other nearby values within the input data. Furthermore, to minimize the input size, CNNs also incorporate pooling layers. These layers employ different types of pooling to replace small matrices within the input and combine them to form a new matrix. The utilization of convolutional layers empowers CNN models to excel in image classification problems. For instance, Krizhevsky et al. [7] constructed a deep convolutional neural network, shown in Figure 2.3, and trained it on the ImageNet dataset. This model, now known as AlexNet, exhibited exceptional performance among all the participants at that time.

2.2 Evolution of Intrusion Detection System

The inception of signature-based Intrusion Detection Systems (IDS) dates back to 1987 when D.E. Denning published the first work in this area [8]. Denning's paper

described a real-time IDS built upon detecting abnormal patterns in system records. This independent IDS model transcends specific system and application environments. For a significant duration, Denning's work served as a reference for subsequent publications [9][10]. Over time, the simplicity of signature-based IDS struggled to cope with the rapid data growth, leading researchers to explore anomaly-based IDSs while also introducing advancements in signature-based approaches. For example, Li et al. [11] presented a signature-based IDS built on collaborative blockchain technology, demonstrating strong performance even amidst the era of data explosion.

With the evolution of machine learning technology, anomaly-based IDSs have emerged as powerful tools due to their performance and advantages. The progress of these IDSs owes much to the valuable contribution of cybersecurity research datasets. Several renowned datasets have played a pivotal role in advancing IDS research. The KDD CUP 99 dataset, introduced in 1999 by University of California researchers, features 41 attributes and encompasses attack types such as Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and probe activities. Subsequently, the NSL-KDD dataset was introduced as an improved version, addressing redundancy by eliminating duplicate records [12]. In recent times, the Canadian Institute of Cybersecurity developed the CICIDS2017 and CSE-CIC-IDS2018 datasets [13], which incorporate network profiles, offering more comprehensive features and attack types. Foundational datasets like KDD CUP 99 and NSL-KDD serve as cornerstones for most IDS research. However, other datasets such as those from DARPA [14], Kyoto University [15] and CDX [16] also play crucial roles in the field. These diverse datasets significantly contribute to the development and evaluation of IDS solutions, enabling researchers to explore various scenarios and challenges within the cybersecurity domain.

Eskin et al. [17] introduced a geometric framework for unsupervised anomaly-based detection. This framework involves mapping data features to a designated feature space, often a vector space, using predetermined kernels. Anomalies are identified in sparse regions within this feature space. This framework was tested on the KDD CUP 99 dataset using three unsupervised learning algorithms: clustering, K-nearest neighbors, and SVM, achieving an optimal detection rate of 98 percent. Meanwhile, Jiang et al. [18] enhanced the nearest-neighbor algorithm and developed

a cluster-based approach for detection, showcasing improved performance over [17] with linear time complexity. Meng applied SVM and DT algorithms to intrusion detection for each attack type in the KDD CUP 99 dataset [19]. With the exception of the U2R attack due to limited records, detection rates for all other types exceeded 95 percent. Additionally, Choudhury and Bhowal [20] tested models including NB, LR and Random Forest (RF) on the NSL-KDD dataset [12], demonstrating that these ML models generally perform well in intrusion detection.

With advancements in hardware and deep learning technologies, the integration of deep learning models into IDSs has gained popularity. Ding and Zhai developed a CNN-based IDS architecture, utilizing three one-dimensional convolutional layers to extract features from raw data records [21]. Extracted features were then passed through multiple dense layers for classification. The CNN's output layer incorporated a softmax function, accommodating various attack classes within the NSL-KDD dataset [12]. A comprehensive comparison between their CNN and four other ML/DL algorithms, including RF, SVM, MLP and LSTM, was conducted. The findings highlighted the CNN's superior performance, with enhanced detection accuracy and a reduced false positive rate compared to other algorithms. In the realm of software-defined networks (SDNs), Tang et al. [22] developed an anomaly-based IDS using a deep RNN. This innovative RNN, constructed with Gate Recurrent Units (GRUs), formed a GRU-RNN. The GRU-RNN's performance surpassed conventional RNNs and standard deep neural networks (DNNs), achieving a remarkable 89 percent detection accuracy on the NSL-KDD dataset [12], despite utilizing only six raw features.

2.3 Generative Adversarial Network

The Generative Adversarial Network (GAN) was introduced by Goodfellow et al. [3] in 2014. The research proposed two Multi-Layer Perceptron (MLP) models: the generator G and the discriminator D. The generator is trained to learn the data distribution of the original dataset, while the discriminator is designed to detect whether a sample comes from the original dataset.

Suppose the data X follows the distribution p_{data}. To enable the generator to learn a distribution p_g of data X, the researchers first define a noise input $p_z(z)$ and use it

to generate output samples denoted as $G(z)$. Simultaneously, the outputs $D(x)$ is a probability indicating whether x comes from G instead of following the distribution p_{data}. During each iteration, both the generator and the discriminator update their parameters based on Eq. (2.3) and (2.4) respectively, where θ_g and θ_d represent the parameters of G and the D.

$$\bigtriangledown_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^i) + \log(1 - D(G(z^i)))] \tag{2.3}$$

$$\bigtriangledown_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(z^i)))] \tag{2.4}$$

The primary objective of G is to maximize the probability that D cannot successfully distinguish between the generated data and the real data. Conversely, D is trained to avoid being deceived by the generator and accurately differentiate between real and generated samples. This dynamic interaction leads to a minimax game, as represented in Eq. (2.5). Goodfellow et al. [3] demonstrated that when the training process converges to global optimality, the generator's learned distribution p_g matches the original data distribution p_{data}. In other words, G successfully learns the underlying data distribution, while D achieves a detection accuracy of approximately 50 percent, reflecting its inability to differentiate between real and generated data.

While GAN has demonstrated its efficacy in generating adversarial samples, the original GAN design faces several existing issues. To make it applicable to a broader range of scenarios and tasks, many researchers have made modifications and developed enhanced frameworks.

One such modification is the introduction of the conditional GAN (CGAN), proposed by Mirza and Osindero [5]. In their paper, they incorporate the labels of the dataset into the training process, transforming the GAN into a supervised learning model. Unlike GAN, the CGAN's discriminator (D) predicts not only the quality of the generated sample but also the likelihood of its label. Therefore, the CGAN's ultimate objective differs slightly from the GAN shown in Eq. (2.6), where y represents the discriminative function result. Consequently, given a specific label, CGAN is able to generate samples of it.

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.5)$$

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x \mid y)] + E_{z\sim p_z(z)}[\log(1 - D(G(z \mid y)))] \quad (2.6)$$

Meanwhile, Arjovsky and Bottou [4] conducted extensive research and analysis on this issue and introduced the Wasserstein GAN (WGAN). In [3], the loss function relies on the Jensen-Shannon (JS) divergence, which involves two Kullback-Leibler (KL) divergences as shown in Eq. (2.7) and (2.8), with $\mathbb{P}g$ and $\mathbb{P}r$ representing two distributions on the dataset and $\mathbb{P}_m$ being their average. Despite its good performance, the traditional GAN's optimization process does not always converge, leading to potential issues of model collapse. To address this challenge, Martin Arjovsky introduced the earth mover's distance (EMD) in Eq. (2.10) as a replacement for the JS divergence. This innovation led to the development of WGAN. Later, he further presented the WGAN-GP in his paper [23], which includes a gradient penalty in the loss function to overcome some challenges encountered during WGAN training. In addition to these two enhanced models, there are many other GAN-based models, such as Deep Convolutional GAN (DCGAN) [6], StyleGAN [24] and CycleGAN [25]. These models show better performances in their particular tasks.

With the exceptional performance of GANs and their various enhanced iterations, a plethora of models and datasets have emerged. Jin et al. [26] analyzed the challenges encountered during their GAN training and successfully stabilized and improved the model's quality. They then utilized this model to generate anime characters, making it available on their website. Subsequently, numerous anime enthusiasts adopted their model, resulting in the creation of several open-source projects, such as the 'pokeGAN' project, which specializes in generating 'Pokemon' characters. Another notable example is the 'pix2pix' model [27]. Operating on the principles of Conditional Adversarial Networks, 'pix2pix' offers a versatile tool for various image-to-image tasks. These tasks include translating black and white photos into colored images, as well as transforming images from day to night. Additionally, the introduction of BigGAN marked another milestone in enhanced GAN models. Trained at a resolution of 128x128 using the ImageNet dataset [28], this new model achieved an impressive Inception Score of 166.5, surpassing the previous best score of 52.52.

$$KL(\mathbb{P}_r \parallel \mathbb{P}_g) = \int \log(\frac{\mathbb{P}_r(x)}{\mathbb{P}_g(x)}\mathbb{P}_r(x)d\mu(x)) \tag{2.7}$$

$$JS(\mathbb{P}_r \parallel \mathbb{P}_g) = KL(\mathbb{P}_r \parallel \mathbb{P}_m) + KL(\mathbb{P}_g \parallel \mathbb{P}_m) \tag{2.8}$$

$$\mathbb{P}_m = (\mathbb{P}_r + \mathbb{P}_g)/2 \tag{2.9}$$

$$EMD(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \epsilon \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\parallel x - y \parallel] \tag{2.10}$$

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)} EMD[D(x)] + E_{z\sim p_z(z)} EMD[(1 - D(G(z)))] \tag{2.11}$$

2.4 White-box and Black-box Attacks

The attacks against the IDS are grouped into white-box attacks and black-box attacks. In [29], Wang employed multiple algorithms, including Jacobian-based Saliency Map Attack (JSMA) [30] and FGSM, to generate adversarial samples for the white-box attacks. The results demonstrated that these adversarial samples exhibited a lower detection rate by IDS, which is reasonable. However, the practical effectiveness of applying these algorithms in real-world attacks remains questionable, as it is challenging to completely evade the IDS and gather all necessary information from it. Therefore, most research focus on approaches in the black-box attacks.

GAN-based attacks are increasingly likely to be applied, particularly due to their ability to target black-box IDS. In a noteworthy research effort, Lin et al. [31] introduced the IDSGAN framework. They employed the WGAN design to train the generator and discriminator based on feedback from a black-box IDS, utilizing the NSL-KDD dataset [12]. During the evaluation process, they observed that a significant portion of the adversarial samples generated by IDSGAN went undetected by many types of IDS. Chauhan and Shah Heydari [32] applied the similar framework on CICIDS2017 dataset. They used the GAN to generate the polymorphic adversarial DDoS samples by changing the attack profile. Their results indicate that defensive systems are still vulnerable to the continuous updating of attacking profile, even if these defensive systems use the incremental learning. Kumar and Sinha [33] instead used the Wasserstein Conditional GAN (WCGAN) and boost it by a XGBoost Classifier. They compared the WCGAN framework with other GAN variants framework on NSL-KDD, UNSW-NB15 and BoT-IoT dataset. In their evaluation, the results

show that the CWGAN performs the best among all the GAN variants and it has a reasonable improvements in XGBoost Classifier but not expected performance on the RF and SVM.

Thanks to the availability of adversarial samples from GANs, scientists are actively exploring two different approaches to enhance IDS performance. One approach involves increasing the number of malicious samples in the training set. By incorporating these adversarial samples during the training process, the IDS becomes more adept at handling limitations and drawbacks, leading to an improved detection rate for the test set. Msika et al. [34] proposed SIGMA that leverages new adversarial samples to enhance the IDS against new attacks. The cycle consists of three steps: generating the samples by GAN and metaheuristics, calculating the detection rate of new samples and training IDS with these samples. After two cycles, the new IDS can already have a more than 99% detection rate. Lee and Park [35] used the GAN to generate the new attacking samples to eliminate the data imbalance. Their results indicate that the RF model trained with the balanced dataset perform better than that trained with original imbalanced dataset. Meanwhile, they also show that their strategy implementing new samples perform better than the other widely used ones.

The other approach is to integrate GANs directly into the training process, providing additional feedback to increase the IDS's ability to detect adversarial malicious samples. de Araujo-Filho et al. [36] proposed a GAN-based IDS using temporal convolutional networks and self-attention to detect the attacks. This new IDS is designed for the edge servers. Their experiments show that the IDS meets the requirements of detection rate and time cost. Meanwhile, it is more accurate and faster than some well-known GAN-based IDSs [37][38].

In conclusion, the application of GANs in cybersecurity resembles a cat-mouse game. Attackers continuously generate new adversarial samples, which can be used to test IDS defenses. For instance, frameworks like DIGFuPAS [39] and DIGFuPAS-2 [40] were developed to generate adversarial samples with high chances of evading detection. However, once these samples are used as an augmentation dataset for IDS training, their effectiveness in deceiving the IDS diminishes. Nevertheless, if the DIGFuPAS training begins again on an updated IDS, the new adversarial samples remain valid. This result is also found in [32]. This continuous cycle of attack

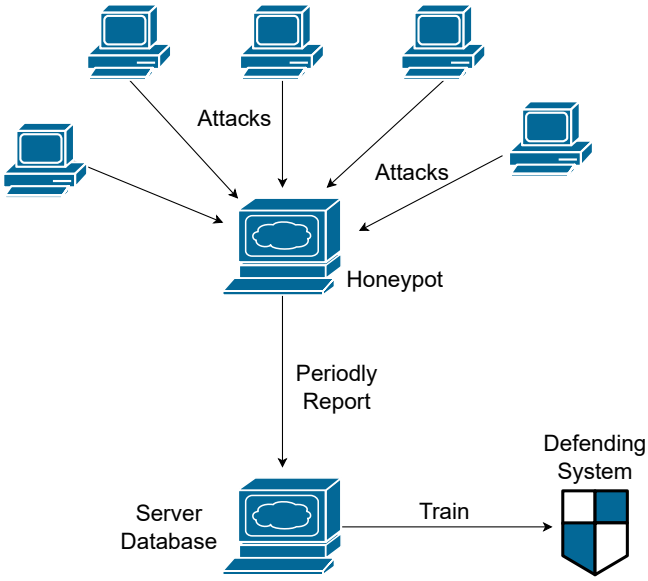

Figure 2.4: The idea of the honeypot. It is used for attracting attacks in the network and reporting these data to the server to help it build the defence.

and defense makes it challenging to predict which side will ultimately prevail. The ongoing dynamic between attackers and IDS researchers demonstrates the complexity of addressing adversarial threats in cybersecurity.

2.5 Honeypot

In addition to active defense using IDS, honeypots serve as a valuable second-layer defense [41]. The fundamental concept is depicted in Figure 2.4. Unlike IDS which aims to prevent attacks, the purpose of a honeypot is to attract attackers and lure them to interact with it. Once an attack is detected, the honeypot collects data and periodically reports recent malicious activities to the actual server, hidden in the protected network. These gathered data allow the server to develop a more robust defense system tailored to combat specific attack patterns. Additionally, honeypots can incorporate technologies to track the IP information of malicious flows, enabling the server to proactively block flows from the same sources before any detection.

Honeypots should be designed to appear vulnerable in specific aspects, such as the firewall and system, to entice potential attackers. However, they should not be entirely transparent. Despite their controlled environment, honeypots can still pose risks if compromised, potentially affecting other devices, including the real server. Thus,

striking the right balance between setting vulnerabilities and maintaining control is crucial. By strategically utilizing honeypots as part of a comprehensive defense strategy, organizations can gather valuable intelligence about potential threats, strengthen their security posture, and enhance their overall cybersecurity resilience.

In contemporary research, the integration of honeypots with IDS has proven to enhance overall performance. Muhammet and Resul introduced a Honeypot-based Intrusion Detection and Prevention System (IDPS) [42]. This system offers real-time visualization of network traffic on servers through animation and, importantly, can identify zero-day attacks through intrusion detection configuration. Another noteworthy example is the Honeypot TB-IDS, which presents a deception trace-back model [43]. Positioned within network intrusion deceptions, it monitors incoming traffic and continually assesses data routes. The process involves capturing incoming traffic and collecting pertinent information. Upon detecting an intruder, this information is sent to the honeypot, enabling the establishment of a mitigation point, leading to the subsequent blocking of all packets from the identified intruder. Mokube and Adams introduced an intrusion detection module built upon honeypot technology, incorporating the innovative IP Traceback technique [44]. This module, augmented by mobile agents, possesses the ability for distributed detection and response. Particularly noteworthy is the module's inherent flexibility, allowing for seamless expansion and dynamic configuration of the entire detection framework. Leveraging honeypot technology, this module excels in tracing intrusion sources to their fullest extent.

Chapter 3

S-IDS: A Misinformation-based Method

3.1 Overview of S-IDS

In the novel 'The Three-Body Problem', the author introduces a remarkable micro-computer known as 'sophon', designed with a size equivalent to that of a particle. Its primary role is to replace standard particles in collision experiments, skillfully following pre-planned paths to deceive and manipulate the final experimental outcomes. Inspired by this intriguing concept, we design an intelligent IDS named Sophon IDS (S-IDS), as it draws significant inspiration from the concept of 'sophon'. Our design centers around constructing an intelligent model that disrupts the training process of IDSGAN.

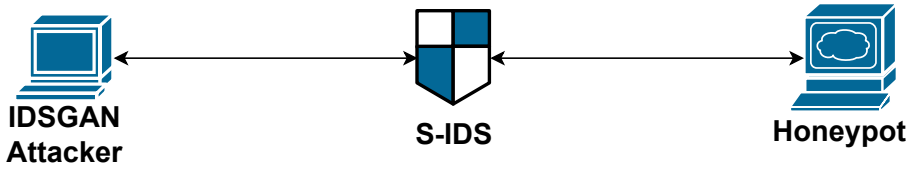

Figure 3.1: The framework of the Sophon IDS during the IDSGAN training. The honeypot is placed to attract the IDSGAN attacks and the S-IDS sends back the feedbacks so that the performace of IDSGAN drops.

Firstly, we introduce the framework of attacking process in Figure 3.1. A honeypot is put here to attract the malicious flows. Between the honeypot and the attacker, we put the S-IDS instead of the normal IDS to give the intelligent responses. The S-IDS only sends response to the GAN-based attacks like IDSGAN, while we remain using IDS on the real server. In addition, we will have a GAN detection model identifying whether the framework is under GAN-based attacks. If the traffic is marked as GAN-based attack, it will go through the S-IDS. Otherwise, it passes through the IDS in the normal way.

Indeed, there is a simpler method to counter GAN-based attacks. When we detect GAN-based attacks occurring, thanks to the GAN detection report, we can adopt an

approach where the server immediately blocks all connections from the attackers. This means that no feedback is provided to the attackers. Without information from the IDS, attackers are unable to train their models. In this method, the system's performance relies solely on the accuracy of detecting GAN-based attack features, rather than determining whether the traffic is malicious or not.

While this approach is straightforward and effectively prevents attacks, it also has a drawback. Attackers become aware of their attacks being thwarted since they receive no response. This prompts them to explore new attack strategies to bypass the defenses. For instance, they might seek advanced methods to generate flows that differ from typical GAN-based ones. If the defense mechanism is eventually compromised, the entire system becomes vulnerable. On the other hand, the S-IDS operates differently. Instead of merely blocking connections, it strategically provides deceptive feedback. This feedback is carefully designed to mislead attackers during their model training. From the attacker's perspective, they gain insights into which packets are accepted and which are rejected, but they remain oblivious to the deception. As the attacker's training progresses, their adversarial samples might not always be valid, and they remain unaware of this fact. The GAN detection model continues to function as expected. In a broader sense, even if the detection is not perfectly accurate, the IDS can still achieve a higher detection rate for these adversarial samples compared to the original adversarial samples.

3.2 Assumptions

Before delving into the specific design within this framework, several fundamental assumptions need to be addressed in our research.

Firstly, we assume that IDSGAN attackers have access to feedback from the server. This enables the GAN to discern predictions based on the feedback conditions. If no feedback is received, or a packet containing a 'connection refused' content is returned, we consider the packet flow as malicious. Otherwise, if the feedback indicates that the IDS believes the flow is normal, the packet is treated as normal. For most types of malicious traffic, the attackers typically disregard the payload in the returned packet. As a result, we assume that we can provide feedback of normal prediction by including irrelevant information as payload when the S-IDS predicts it as normal. Conversely,

for normal flows predicted as malicious by the S-IDS, we send a packet with an error code.

Furthermore, an essential concern is how to construct the dataset. In reality, most black-box IDSs are developed using private datasets, while attackers create separate datasets based on previous experience and feedback from servers. To simplify the problem, we assume that the IDS and S-IDS share the same dataset with the IDSGAN. Even if our S-IDS can access the dataset labels, we presume that the attackers also possess the ability to know the true labels.

Lastly, we assume that all flow information generated from generator G is valid. In cases where some feature values may be invalid, we adjust them to the closest valid values.

By establishing these basic assumptions, we can proceed with the subsequent details and evaluation of our design.

3.3 Details of IDSGAN

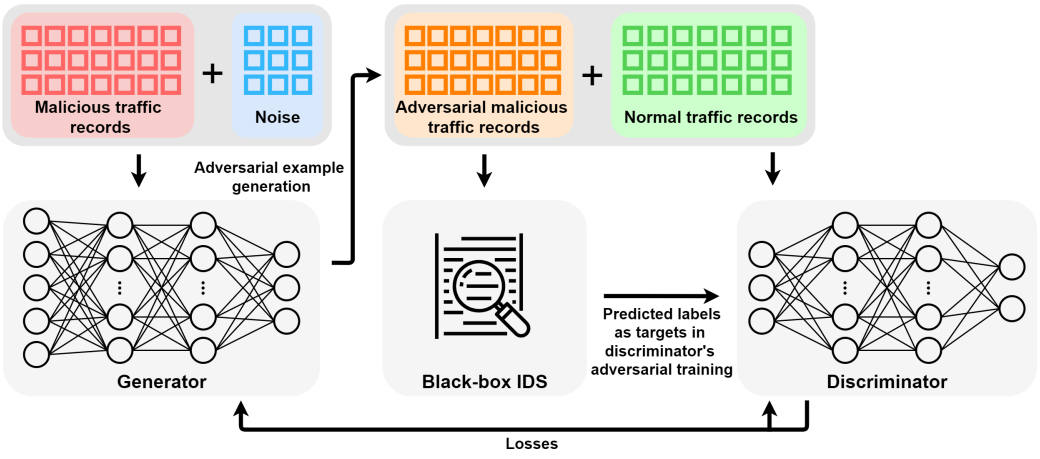

Figure 3.2: The framework of IDSGAN based on WGAN.

Before delving into the specific algorithm designs, it is crucial to consider the reasons why the S-IDS can help increase the detection rate against adversarial samples generated by the IDSGAN [31]. The entire evaluation framework of IDSGAN is depicted in Figure 3.2. To achieve its objectives, IDSGAN creates two MLP models: generator G and discriminator D. IDSGAN's primary goal is to generate adversarial samples while preserving the malicious features of the original samples. To achieve

this, G is limited to modifying only the non-functional features, which have less significance in the network. As a result, G takes the combination of non-functional features of malicious flows and latent noise vectors as input, while D receives the output of G and predicts whether the adversarial sample can be detected.

For G, the loss function is defined in Eq. (3.1), where S_{mal} represents the original malicious flows, nf denotes the non-functional features, and N stands for the latent noise vector. On the other hand, the loss function of D is defined in Eq. (3.2), where B_{normal} and B_{mal} represent the normal and malicious traffic records marked by the black-box IDS, respectively.

The underlying principle of IDSGAN is to ensure that D cannot differentiate between malicious and normal samples based on the feedback received from the black-box IDS. By adhering to this principle, IDSGAN strives to create undetectable adversarial samples while maintaining the malicious features of the original samples.

$$L_G = \mathbb{E}_{nf \in S_{mal}, N} D(G(nf, N)) \tag{3.1}$$

$$L_D = \mathbb{E}_{s \in B_{normal}} D(s) - \mathbb{E}_{s \in B_{mal}} D(s) \tag{3.2}$$

During the training process, IDSGAN undertakes two major steps. Firstly, G generates adversarial malicious traffic using the existing malicious samples S_{mal} and updates its parameters according to Eq. (3.1). Subsequently, both the black-box IDS and D provide predictions for the adversarial and normal samples, and D updates its parameters using Eq. (3.2). These steps are repeated iteratively until the detection rate for the adversarial samples converges. At this point, the outputs of D for adversarial samples become indistinguishable from benign samples, while the outputs of D for the two types of samples classified by the black-box IDS are nearly identical.

In conclusion, the primary method to interfere with IDSGAN training, aside from blocking feedback, is to inject fraud information into Eq. (3.1) and (3.2). However, given that we have assumed that IDSGAN knows the true labels of samples, the only feasible modifications we can make are to B_{normal} and B_{mal}. In essence, we need to modify the feedback, or more directly, the IDS itself, in order to disrupt the training process of IDSGAN effectively.

3.4 Details of MLP-based and RNN-based IDS

Before proceeding with our evaluations, the first step is to train a well-performing IDS. For this purpose, we choose two types of deep learning models: MLP and RNN. After conducting various tests, we present the final designs of our MLP and RNN models in Table 3.1 and Table 3.2, respectively. Additionally, to facilitate comparison, we include the LR model in the final tests, using the default configuration.

Table 3.1: Multilayer Perceptron IDS layers

Operation	Input Dimension	Output Dimension	Activation
Linear	input_dim	input_dim/2	ReLU
Linear	input_dim/2	input_dim/2	ReLU
Linear	input_dim/2	input_dim/2	ReLU
Linear	input_dim/2	output_dim	Sigmoid

Table 3.2: Recurrent Neural Network IDS layers

Operation	Input Dimension	Output Dimension	Activation
Recurrent Layer	input_dim	hidden_dim	None
Linear	hidden_dim	input_dim/2	None
Linear	input_dim/2	output_dim	Sigmoid

3.5 IDSGAN Training

For the G model, rather than adding noise directly to the original sample, we generate a vector of length 8, denoted as $N = (n_1, n_2, \ldots, n_8)$, following a normal distribution. This vector serves as the random noise appended to the end of the original sample. Consequently, the final input for the G model is represented as shown in Eq. (3.3).

$$x_{g-input} = (x_{nf}, N) \tag{3.3}$$

Then, for the D and S-IDS, we need to restore those functional features back and get the final adversarial sample in Eq. (3.4).

$$x_{adv} = (x_f, G(x_{g-input})) \tag{3.4}$$

The model descriptions for our G and D are provided in Table 3.3 and Table 3.4, respectively. During each epoch of the training process, the parameters of G and D are updated according to Eq. (3.1) and (3.2), respectively.

Table 3.3: Model details of Generator

Operation	**Input Dimension**	**Output Dimension**	**Activation**
Linear	input_dim	input_dim*2	ReLU
Linear	input_dim*2	input_dim*2	ReLU
Linear	input_dim*2	input_dim	ReLU
Linear	input_dim	input_dim/2	ReLU
Linear	input_dim/2	output_dim	Mixed

Table 3.4: Model details of Discriminator

Operation	**Input Dimension**	**Output Dimension**	**Activation**
Linear	input_dim	input_dim*2	LeakyReLU
Linear	input_dim*2	input_dim*2	LeakyReLU
Linear	input_dim*2	input_dim	LeakyReLU
Linear	input_dim	input_dim/2	LeakyReLU
Linear	input_dim/2	output_dim	None

There are two important clarifications to make. Firstly, the activation layer of the output layer in G is mixed. Since G generates continuous values, while some features are discrete, we use a threshold of 0.5 to set these features to either 0 or 1. For the remaining features, we use the LeakyReLU activation function.

The other issue pertains to the reasonability of the G's output. In the beginning, as the parameters are randomly initialized, the output for some features could be in a wide range, such as [-100, 100]. Although the update of G and D remains linear, these extreme values cannot be directly used for IDS detection. Hence, during the evaluation phase, we clamp these values using Eq. (3.5), where $min(i)$ and $max(i)$ represent the minimum and maximum values for feature i, respectively.

By addressing these issues, we ensure that the G and D models are appropriately adjusted during the training and evaluation phases, optimizing their performance for IDS detection.

$$x_i^{adv} \in [min(i), max(i)] \qquad i \in nf \tag{3.5}$$

3.6 Details of S-IDS

3.6.1 Training of S-IDS

Algorithm 1 Training of S-IDS

Require:

x^s, y^s, S-IDS;

Ensure:

Initialize that S-IDS equals with IDS;

1: **while** S-IDS has not converged **do**

S-IDS.$update(x^s, y^s)$

2: **end while**

3: $return$ S-IDS

After selecting the modified samples, the next step is to build the S-IDS. We propose the process in Algorithm 1. It updates the S-IDS based on the original IDS. In this method, a batch containing all the modified samples is created, and a large learning rate is used to update the model. This approach takes advantage of the concept of catastrophic forgetting observed in ML models. When the model is trained with new samples, its performance on old samples may decrease. By incorporating the modified samples and updating the model with a large learning rate, the S-IDS can potentially exhibit a greater difference from the original IDS.

3.6.2 Key Parameters of S-IDS

Before we discuss the exact algorithm, we firstly introduce some parameters in it. The specific algorithm may work differently due to these parameters. The meanings of these symbols and classifications are in Table 3.5. Now, we will give the reasons setting these parameters.

Actually, there are several ways to select the samples. As the S-IDS model we build is a binary-classification model, we can modify both labels or only one specific

Table 3.5: Definitions of Parameters

Group	Symbol	Definition
$S \in \{s_0, s_1, s_{01}\}$	s_0	Only change samples from malicious to benign
	s_1	Only change samples from benign to malicious
	s_{01}	Change samples with both labels
$P \in \{U, D\}$	U	Larger value has the priority in selection
	D	Smaller value has the priority in selection
TV $\in \{\text{DVT}, \text{DASOS}\}$	DVT	Distance between IDS value to threshold
	DASOS	Distance between adversarial sample and original sample
$\omega \in (0, 1)$	The percentage of selected samples	

label. The different choices could have different impacts on the S-IDS performance, such as the recall and F1-score. We would like to determine which kind of S-IDS can have the best effect. Therefore, we use the side flag S to control the changed labels, and the number of modified samples is controlled by argument ω.

Furthermore, in order to find the most appropriate samples, we need to rely on some result. As mentioned above, the IDS prediction loss and the distance between adversarial samples and original samples are the possible ones. Thus, we define two transformed value TV functions, and the functions are defined in Eq. (3.6) and (3.7). The parameter t represents the threshold, and its value depends on different IDS designs. For the 'sigmoid' activation, it is 0.5, while for 'tanh', it is 0. For the DASOS function, we use the Euclidean Distance. Meanwhile, whether a large value or a small value is needed could lead to different scenarios. Therefore, we set the priority flag P to select the relevant samples.

$$\text{DVT}(x) = |\text{IDS}(x) - t| \tag{3.6}$$

$$\text{DASOS}(x) = \sqrt{\sum_{i=1}^{n} (x_i^{adv} - x_i)^2} \tag{3.7}$$

3.6.3 Label Flipping in S-IDS

As mentioned above, we have a S flag and parameter ω controlling the number of samples selected. Given the imbalance of the dataset and the number of malicious samples are less than the benign samples, we use the former to calculate the number of modified samples of each label. Meanwhile, due to the S flag, we have the modified

samples of each label n_0 and n_1 shown in Eq. (3.8) and (3.9). With the number limitations, we can go on to the final algorithm of our flipping design.

$$n = len(X^{mal}) * \omega \tag{3.8}$$

$$(n_0, n_1) = \begin{cases} (n, n) & S = s \\ (0, n) & S = s_0 \\ (n, 0) & S = s_1 \end{cases} \tag{3.9}$$

Suppose we have the training set X and its real labels y_{real}. We clone another labels y_{noise} recording the new labels we have modifying. If we have modify it into another label, we cannot do any operations to this sample any longer. In order to make the models inaccurate, we first need to classify all the samples into two groups for the two label 'benign' and 'malicious'. For each sample in each group, the prediction result, the real label and the new label is same. If not, as discussed in previous sections, we think modifying this label can lead to the model behave more accurate. That is to say, for the samples x^{I_0} and x^{I_1}, they meet the requirements that:

$$x^{I_0} = \{x^i | \text{IDS}(x^i) \leq t, y^i_{real} = 0, y^i_{noise} = 0\} \tag{3.10}$$

$$x^{I_1} = \{x^i | \text{IDS}(x^i) > t, y^i_{real} = 1, y^i_{noise} = 1\} \tag{3.11}$$

where t is the threshold for the IDS. As mentioned in the DVT function, it is based on the design of the IDS.

After selecting and grouping these samples, we use the heap select algorithm, which has lower complexity, to choose a subset of the available samples x^{i_0} and x^{i_1} based on n_0, n_1 and the priority flag P. After modifying labels of those samples, we can then continue the S-IDS training. In conclusion, we can present our final flipping algorithm as shown in Algorithm 2.

3.7 Further Discussions

In fact, there are more analysis on the choices designing the algorithm. Given the limitations of the section, we put the related analysis in the following contents.

Algorithm 2 Label Flipping in S-IDS

Require:

$X, y_{real}, y_{noise}, \text{IDS}, n_0, n_1, \text{TV}, P$;

Ensure:

IDS is able to give probability value;

The S-IDS has the same layers as IDS;

$n_0, n_1 \geq 0$;

1: Initialize the S-IDS using IDS parameters;
2: $y = \text{S-IDS}(X)$;
3: $X^{I_0} = \text{TV}(Select(y, y_{real}, 0))$;
4: $X^{I_1} = \text{TV}(Select(y, y_{real}, 1))$;
5: $X^{i_0} = HeapSelect(X^{I_0}, n_0, P)$;
6: $X^{i_1} = HeapSelect(X^{I_1}, n_1, P)$;
7: $y^{i_0}_{noise} = 1, y^{i_1}_{noise} = 0$;
8: $\text{S-IDS}.train(X^{i_0 \cup i_1}, y^{i_0 \cup i_1}_{noise})$;

3.7.1 Sample Modification in Machine Learning

To modify the feedback from a Machine Learning (ML) model, the direct approach is changing the configuration of the model itself. The ML model is constructed based on the dataset, and in binary-classification tasks, the model functions as a boundary between different labels. Any modifications to the dataset will also alter the boundary of the model. As illustrated in Figure 3.3, simply changing the labels of the samples marked in light green and light red can shift the boundary from $f(x)$ to $f^{'}(x)$, while both boundaries still exhibit similar characteristics. For instance, the general behavior of y declining as x rises is retained in both cases. If we desire our model to behave differently from the original one, we must consider the fundamental principles that underlie the model. Merely changing labels may not be sufficient to achieve the desired shift in behavior. Instead, we need to delve into the model's architecture, loss function, or other aspects to effectuate significant changes in its behavior.

Looking back to the training process of ML models, the way it updates the model parameters is by calculating the training loss from the last iteration. The loss calculation is up to the specific algorithm selected for the ML model. As final goal for

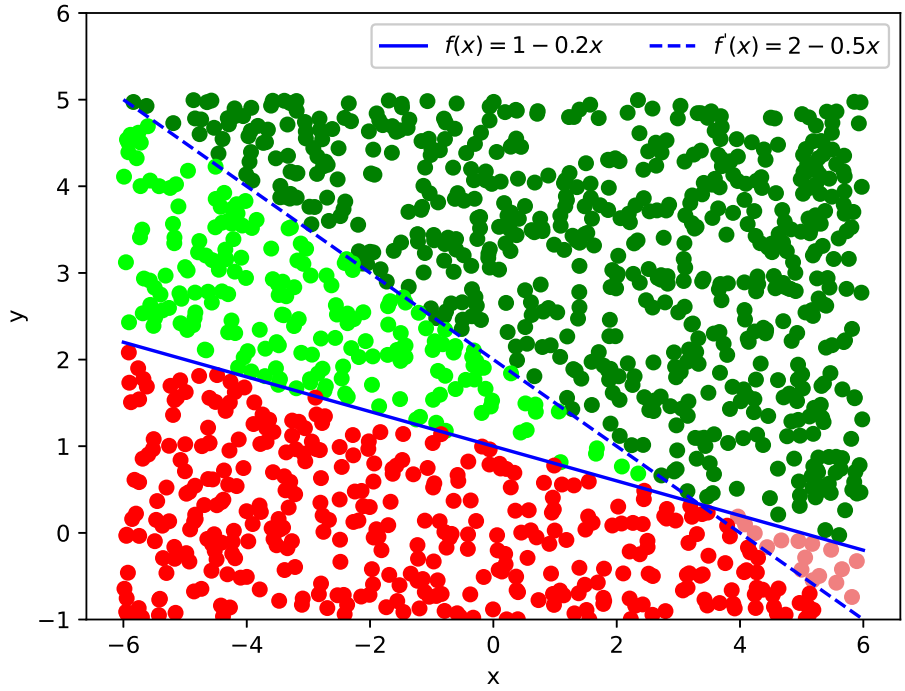

Figure 3.3: An example of dataset. Green and red dots stand for the two types of sample and it creates the classification boundary $f(x)$. If we modify the labels of the samples with light color, the boundary will be changed to $f^{'}(x)$ in the optimal result.

the model is minimizing the error of prediction, or the model loss, for each iteration, we will update the parameters in the model based on the derivation inside the loss function. In most cases, With the *loss* and learning rate ϵ, the parameters θ will be updated as shown in Eq. (3.12), where $L(\theta_t)$ is the loss for model for the epoch t.

$$\theta_{t+1} = \theta_t + \epsilon * \bigtriangledown L(\theta_t) \tag{3.12}$$

Suppose we have already a well-trained model M and the parameters θ using the dataset D. Now we modify some labels and get another dataset $D^{'}$. Meanwhile, we clone another model $M^{'}$ and the corresponding parameters $\theta^{'}$. If the training epoch goes, then for the next loss calculation, the loss result could be different. If the loss of model $M^{'}$ is no more than M, as M has already been converged, $M^{'}$ is also converged at this moment, which means the update for $M^{'}$ stops at this moment. Eventually, $M \approx M^{'}$. In the other side, if the loss of model $M^{'}$ is much larger than M, the new model could be not in convergence. Therefore, the model will continue updating based on the loss for each epoch. In the end, $\theta \neq \theta^{'}$. Meanwhile, as the loss is larger,

in order to be converged, the model has to follow some false labels, which means it is less accurate.

However, in some cases, even if the loss increases, the converged model $M^{'}$ is still likely to share the same accuracy with M. That is due to the inproper increase in loss. Even the parameters θ^{i} varies, the original dataset still follows the distribution in the parameters and eventually, the final performance remains unchanged. Therefore, in order to have the final M^{i} unlike M as much as possible, we need to maximize the artificial loss we add into the model. Then, there are two questions to be solved.

The first one is what kind of samples are under consideration. Given the prediction from the model and its original label in the dataset, there are two groups: samples predicted right and samples predict wrong. For the former group samples, as the original model gives them right predictions, they will be predicted wrong in the new dataset and contribute loss to the model. On the other hand, for the latter group samples, the loss will decrease after modifying them. In conclusion, we choose the former group samples.

The other one is about the priority of samples. Even if samples are in the same group, the loss contribution of them is different. In some regression loss function, the result is a continuous one. For some samples, the loss of them is large while for the rest, the loss is low. Apparently, we should consider the samples that have larger loss, as it can reshape the $\theta^{'}$ in a big step. However, in ML theory, not everything should seek for ultimation. The way training the ML model is to seek for a balance point. Possibly, there is a time or selection when not everything selected for the best can have a highest performance. But in anyway, using different priorities could help us find the best solution to it.

3.7.2 Features of Adversarial Samples

To create well-designed adversarial samples, the goal is to identify weaknesses in the ML model and make the smallest possible modifications to the input while still causing misclassification. As illustrated in Figures 3.4 and 3.5, consider the original normal sample x represented by the green dot. The black dot denotes the adversarial sample generated based on the modified boundary $f^{'}(x)$. In this case, the adversarial sample will lead to a different prediction from x according to $f^{'}(x)$, while it does not influence

$f(x)$, as shown by the red dot. The key to successful adversarial sample generation lies in finding the right balance between perturbation and closeness to the decision boundary. A carefully crafted adversarial sample can effectively expose vulnerabilities in the ML model without drastically altering the input's original characteristics.

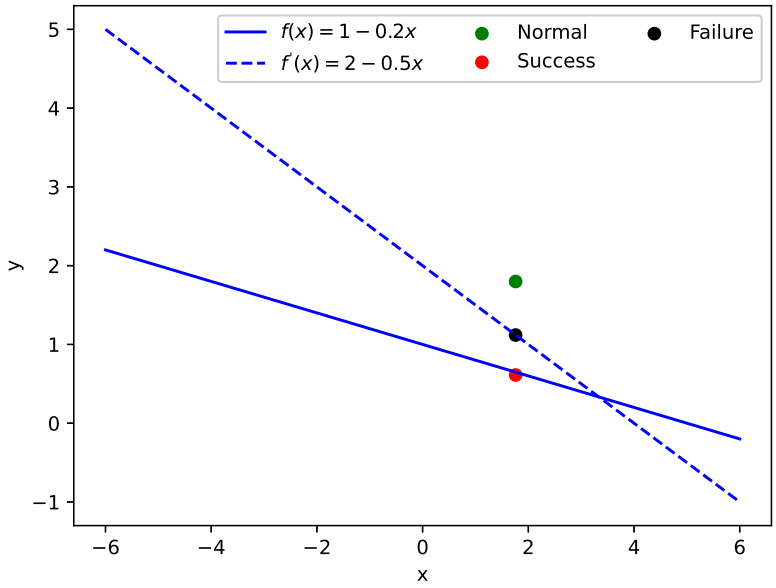

Figure 3.4: An example of adversarial sample generation in a linear model. By changing the boundary, the adversarial sample in black can only invade the new one but the sample in red is expected.

For D, the loss depends on the predicted labels of IDS for both adversarial and benign samples. If the S-IDS has a significant discrepancy with IDS, it can create a substantial impact on D. Similarly, the update of G is based on D's performance in detecting adversarial samples. If G behaves differently, D will follow suit. Additionally, since D takes the output of G as input, any changes in the training dataset of D can have further unpredictable effects.

Considering the above factors, we can modify the samples whose predicted and true labels are the same to increase the ML model loss, which is in line with the S-IDS's objective. To determine which samples have a considerable impact on G and D training, we propose a simple hypothesis that this impact is related to the distance between the original sample and the adversarial sample. Both positive and negative correlations between this distance and the impact are considered in our tests to account for different possibilities.

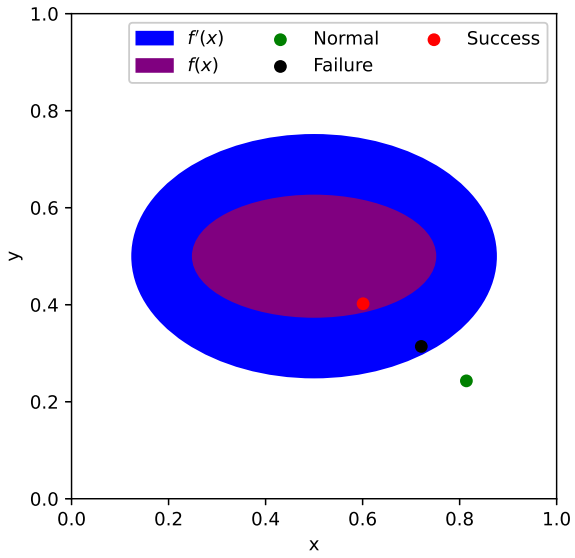

Figure 3.5: An example of adversarial sample generation in a ellipse boundary. By expanding the boundary, the adversarial sample in black can only invade the new one but the sample in red is expected.

3.7.3 Sample Selection Process

Given the samples selected, we have two ways to change the labels of these samples and the update the S-IDS. The first one is modifying them many times. In this way, we only select some of the samples and update our model each time until no samples are left. The other way is modifying them in just one time. In the end, we only select the second way based on the following analysis.

The first reason is about the data distribution. When we modify some samples, we can let S-IDS reshape another data distribution $p^{'}_{data}$, and most of the modified samples will be closer to it than the original one. However, when we modify another group of samples, they are from $p^{'}_{data}$, and S-IDS forms another data distribution $p^{''}_{data}$. When we look back at the first series of modified samples, they could be farther from $p^{''}_{data}$ than $p^{'}_{data}$. That means $p^{''}_{data}$ could partially bounce back to the p_{data}. In addition, for the rest groups of samples, the changes are not quite under control. When the numbers of modified samples are the same, modifying in one time creates a larger gap than modifying in several times.

The other reason is not enough loss. When modifying in several times, the noise ratio in each time is divided in the same number. For the small group of samples,

even if we select the best ones, the loss it contributes to the S-IDS is quite small. It is hard for the S-IDS to sacrifice the right results to decrease the loss. In the end, the model is changed in a small degree. Although we repeat it several times, the modification is still less than in only one time but all available samples, just like the distance principal shown in Eq. (3.13).

$$D(\vec{v}_0, \vec{v}_f) \geq D(\vec{v}_0, \vec{v}_n) \tag{3.13}$$

$$D(\vec{v}, \vec{v}_f) = \sum_{i=1}^{n} D(\vec{v}_{i-1}, \vec{v}_i) \tag{3.14}$$

Chapter 4

Performance Evaluation

In this chapter, we will present a comprehensive overview of the evaluations conducted during this research. Firstly, we introduce the evaluation setup, which includes the introuduction and preprocessing steps of the CICIDS2017 dataset [13] and details about the simulation environment. Next, we showcase the performances of our IDSs and the IDSGAN attacks under various configurations. Finally, we conduct multiple comparison experiments to assess the effectiveness of the Sophon IDS (S-IDS) framework in response to IDSGAN attacks.

4.1 CICIDS2017: A Comprehensive Dataset for IDS Studies

4.1.1 Overview of CICIDS2017

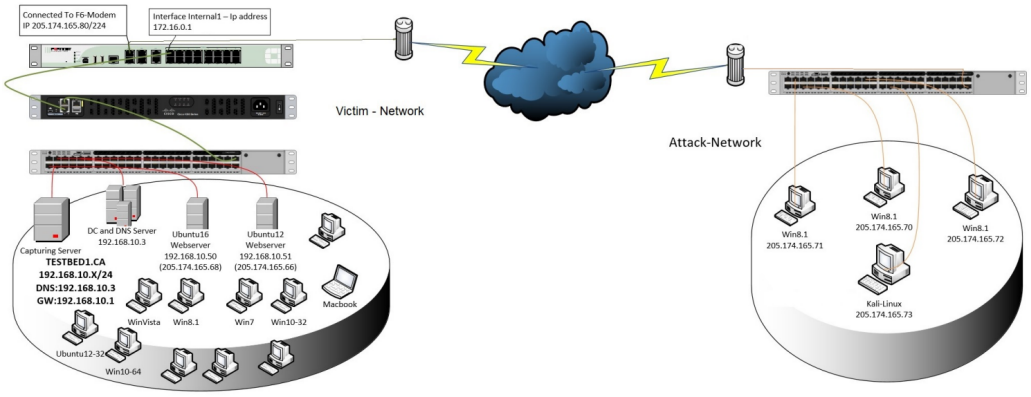

Figure 4.1: The testbed architecture in CICIDS2017. It includes the Victim-Network and Attack-Network, both of which contain all the necessary equipments and systems.

Sharafaldin et al. [13] proposed a new dataset called CICIDS2017. They employed a testbed architecture, as shown in Figure 4.1, to collect real-world traffic. Unlike previous datasets, CICIDS2017 includes all the essential equipment present in the Victim-Network. Additionally, they utilize the B-Profile system [45] to generate

benign traffic across multiple protocols. During a 5-day data capture of real-world traffic, they collected a substantial number of flows and analyzed the results using CICFlowMeter to extract relevant features. This dataset encompasses both benign traffic and a wide range of up-to-date attacks, such as denial of service (DoS), port scan, and infiltration. The number of samples for each attack type is shown in Table 4.1. Based on this dataset, they trained several IDSs and provided the top feature weights for each type of attack.

Table 4.1: Number of each type of traffic in CICIDS2017

Type of Traffic	**Number**
Benign	2273097
DoS Hulk	231073
PortScan	158930
DDoS	128027
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS Slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Web Attack: Brute Force	1507
Web Attack: XSS	652
Web Attack: Brute Force	36
Web Attack: Sql Injection	21
Heartbleed	11

4.1.2 Preprossing

In addition to the samples with bengin labels, we choose four types of attacks: DDoS, DoS Hulk, DoS GoldenEye and portscan as they are the types having the top 4 numbers. We mark all these four types as malicious so than we transform the task from the multi-classification into binary-classification.

Even if Sharafaldin et al. [13] provides the comma-seperated values (CSV) files of the dataset, the feature values are still not in a completely useful format in ML models and we need to do some preprocessing. In the first step, we have to remove those samples whose feature values are incomplete or invalid, which helps us in the

following training. Then, we will handle the features in the dataset. The features are classified into three major groups: categorical features x_{cat}, continuous features x_{con} and flags x_{flag}.

Firstly, for the continuous features, we scale the them into a range of 0 and 1. Suppose the values of one sample and the whole dataset of the feature i are x_i and X_i. We use the mini-max scale to get the new feature value $\bar{x}_i$ in Eq. (4.1). However, some reasonable values are still outside the [0,1] range, as in fact the valid range is larger than that in this dataset. Nevertheless, we do not use the theoretical minimum and maximum values to do the scaling because most new values are very small, which does no help to the training.

$$\bar{x}_i = \frac{x_i - \min(X_i)}{\max(X_i) - \min(X_i)} \qquad i \in con \tag{4.1}$$

Since we cannot apply Eq. (4.1) to the categorical values x_{cat}, a different approach is required. For categorical features with a large number of values, such as IP addresses and ports, we opt to remove them from the dataset, as they make minimal contributions to the detection task. Conversely, for categorical features with only a small number of unique values, we use dummy variable encoding. In this method, if a feature has n possible values, the size of the vector representing this feature is $\log n$, significantly reducing the dimensionality compared to one-hot encoding. We represent the encoded values of these categorical features as x_{ev}.

To simplify the training process, we reorder these features in Eq. (4.2) so that the models can easily select and concatenate different features. Subsequently, we split the samples into a training set and a test set in a 7:3 ratio. Both the training set and the test set are utilized during the IDS and S-IDS training phases, while only the training set is employed in IDSGAN training.

$$\bar{x} = (\bar{x}_{con}, x_{flag}, x_{ev}) \tag{4.2}$$

4.1.3 Functional and Non-functional Features

As mentioned earlier, the IDSGAN framework only permits modification of the non-functional features. Following the training process, we also need to separate the

Table 4.2: Feature weight for the each label

Type of traffic	**Feature**	**Weight**
Benign	B.Packet Len Min	0.0479
	Subflow F.Bytes	0.0007
	Total Len F.Packets	0.0004
	F.Packet Len Mean	0.0002
DoS Hulk	B.Packet Len Std	0.2028
	B.Packet Len Min	0.1277
	Flow Duration	0.0431
	Flow IAT Std	0.0227
PortScan	Init Win F.Bytes	0.0083
	B.Packets/s	0.0032
	PSH Flag Count	0.0009
DDoS	B.Packet Len Std	0.1728
	Avg Packet Size	0.0162
	Flow Duration	0.0137
	Flow IAT Std	0.0086
DDoS GoldenEye	B.Packet Len Std	0.1585
	Flow IAT Min	0.0317
	Fwd IAT Min	0.0257
	Flow IAT Mean	0.0214

functional features from the rest. According to [13], we can obtain the weight for each feature contributing to the decision of a certain type of traffic, as shown in Table 4.2. We select all the features listed in this table as the functional features, while the remaining features are considered non-functional features.

4.2 Evaluation Setup

The entire evaluation consists of two main parts. The first part involves the training of IDSGAN, where we employ the framework shown in Figure 3.1 to simulate real-world attacking scenarios. The second part is the performance test for S-IDS under the trained IDSGAN, utilizing the framework depicted in Figure 4.2. During this evaluation, we utilize all the samples from the test set, and the IDSGAN will reshape these samples to generate the corresponding adversarial samples.

Since the S-IDS is solely responsible for sending fraudulent information, we apply the original IDS to identify the adversarial samples. We collect the number of malicious and benign traffic instances for these malicious samples and calculate the detection rate (DR) accordingly. To account for the potential influence of random factors on performance, we repeat the tests 5 times.

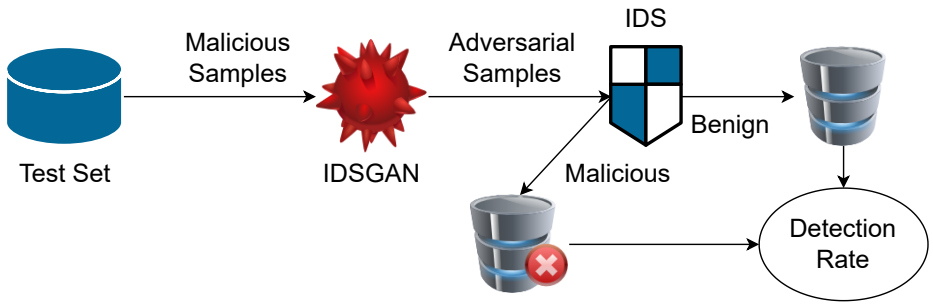

Figure 4.2: The testbed architecture in testing performance of S-IDS. Note that the effect of S-IDS is inside the IDSGAN framework.

4.3 Performance of IDS and IDSGAN

We have built four IDS models using the MLP, RNN and LR algorithms and compare the confusion matrix of them. As shown in the Figure 4.3, the MLP model is most accurate in both the benign samples and malicious samples. The RNN model performs the second and the LR model has the worst accuracy.

Next, we proceed to compare the performances of the IDSGAN framework on attacking the four IDS models mentioned earlier. For each original malicious sample, we utilize the corresponding adversarial sample generated by the IDSGAN's generator and assess the detection rates in these four IDS models.

As shown in Figure 4.4, the average detection rates for the original malicious samples of the IDSs in MLP, RNN and LR models are 97.7%, 95.7% and 90.3%, respectively. However, the average detection rates for the adversarial samples are only 8.7%, 17.0% and 21.4%, respectively. The comparison clearly demonstrates that the IDSGAN is capable of generating adversarial samples that largely go undetected by the IDS. This performance variation is evident across the different IDS model types, with the best results achieved for the MLP model and the worst for the LR model. Additionally, we observe that the performance of IDS in LR models is not

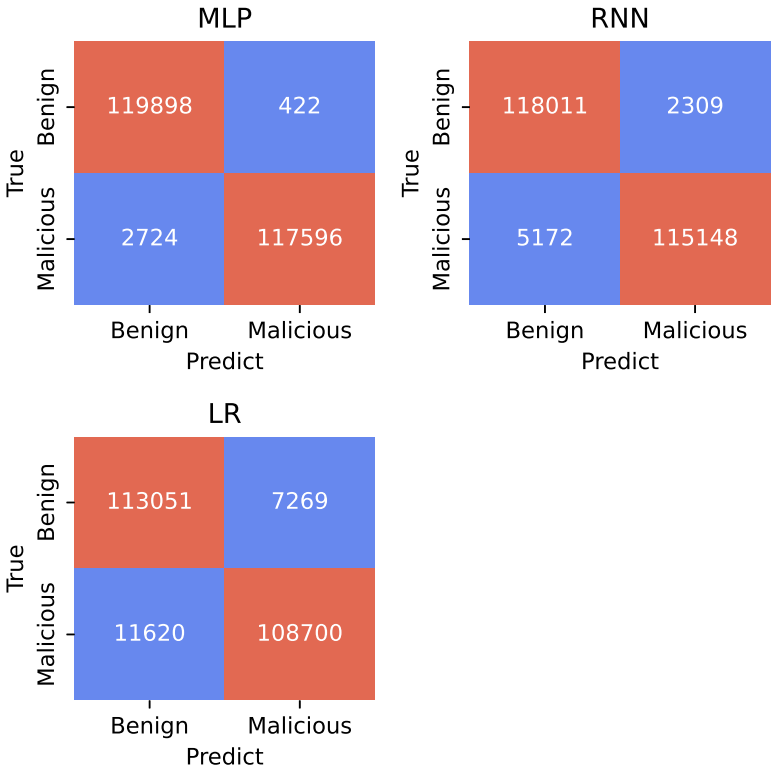

Figure 4.3: The confusion matrix of MLP, RNN and LR based IDS models

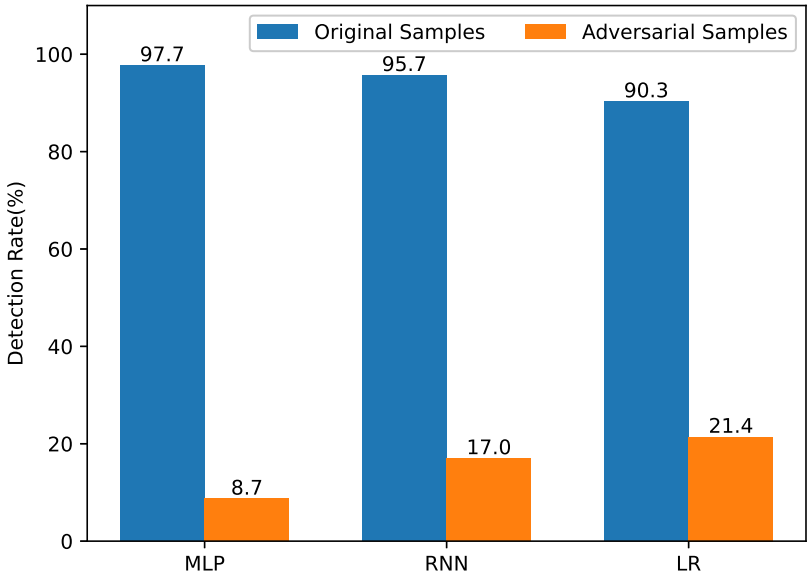

Figure 4.4: The average detection rate of original or adversarial malicious samples in MLP, RNN and LR based IDS models

as strong as that of the MLP and RNN. Clearly, the simple ML models struggle to handle this dataset effectively.

Then, We focus on the MLP and RNN models. The MLP-based IDS exhibits a higher detection rate for the original malicious samples compared to the RNN-based IDS, with the latter being approximately 2 percent lower. However, the average detection rate of the RNN-based IDS is about 8 percent higher than that of the MLP-based IDS when considering adversarial samples. Although these results might suggest that MLP is better at detecting original malicious samples while RNN is better at detecting adversarial samples, it is essential to note that even the RNN-based IDS cannot defend against the IDSGAN attacks.

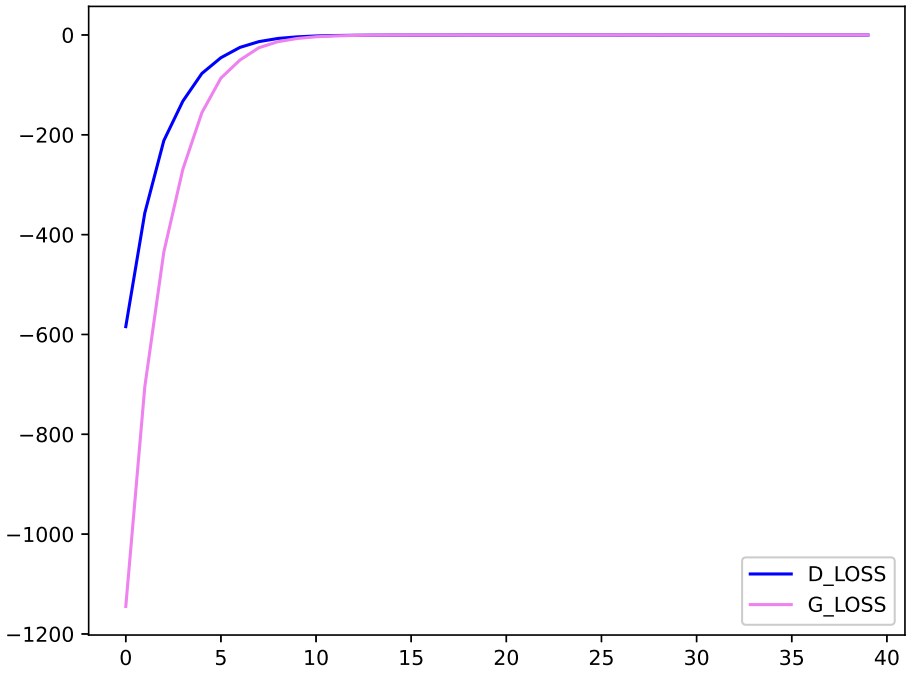

Figure 4.5: The loss of generator and discriminator during the IDSGAN training on MLP-based IDS.

Now, we record the loss values during the IDSGAN training. We take the training process on the MLP-based IDS as an example, and the results are shown in Figure 4.5. From the start, both the generator and discriminator show significant absolute loss values, and as the training progresses, these values gradually converge to zero. As discussed in the last chapter, this moment indicates that the adversarial samples generated by the generator are nearly successful, while the discriminator struggles to

distinguish between the benign and malicious samples from the black-box IDS.

4.4 Performance of S-IDS

In this section, we evaluate the performance of our S-IDS. To identify the best S-IDS configuration, we apply all possible combinations of the parameters mentioned in the last chapter. For this evaluation, we specifically choose $\omega \in \{0.3, \ldots, 0.9\}$. To facilitate comparison among all the potential strategies, we label them with different name-tags as shown in Table 4.3. To mitigate random bias to some extent, we repeat the tests 5 times and present the average results in the graphs. The values of ω are represented on the x-axis, while the average detection rates (DR) are plotted on the y-axis. Additionally, we include the detection rate of the random selection strategy as a reference for comparison.

Table 4.3: Name tag of all possible scenarios

Parameters	DASOS $P = U$	DASOS $P = D$	DVT $P = U$	DVT $P = D$
$S = s_0$	DASOS-U-0	DASOS-D-0	DVT-U-0	DVT-D-0
$S = s_1$	DASOS-U-1	DASOS-D-1	DVT-U-1	DVT-D-1
$S = s_{01}$	DASOS-U-01	DASOS-D-01	DVT-U-01	DVT-D-01

Firstly, we will compare the performance of these strategies under the RNN model. The results are shown in Figures 4.6 and 4.7.

As evident from these figures, in most cases, the detection rate rises as ω increases. Now, we compare the results shown in these two figures.

Firstly, we compare the results of the DASOS function in Figure 4.6. Here, we observe that as ω increases, the average detection rate generally rises. Depending on the specific strategy, the peak value appears at different ω values. For instance, the 'DASOS-D-1' strategy reaches a peak value of 47.1% at $\omega = 0.8$, while the 'DASOS-U-1' strategy peaks at 46.8% at $\omega = 0.6$. Overall, the average detection rate shows significant improvement, but it is still comparable to or worse than the random selection strategy.

Next, we examine the results of the DVT function in Figure 4.7. Similarly, the average detection rates generally increase with ω, and the peak value varies depending on the strategy. However, unlike the DASOS function, all the DVT-based strategies

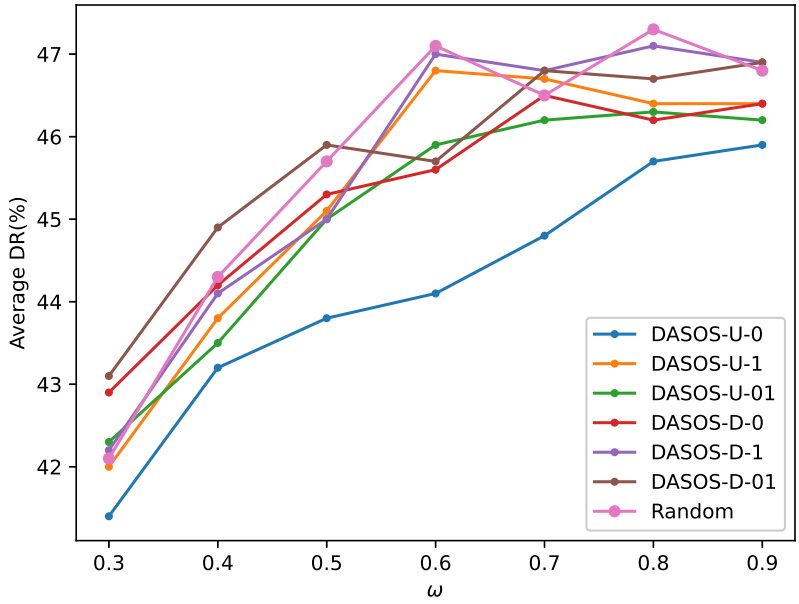

Figure 4.6: RNN(DASOS) - The average detection rate under diffrent ω using DASOS as the TV function in RNN models

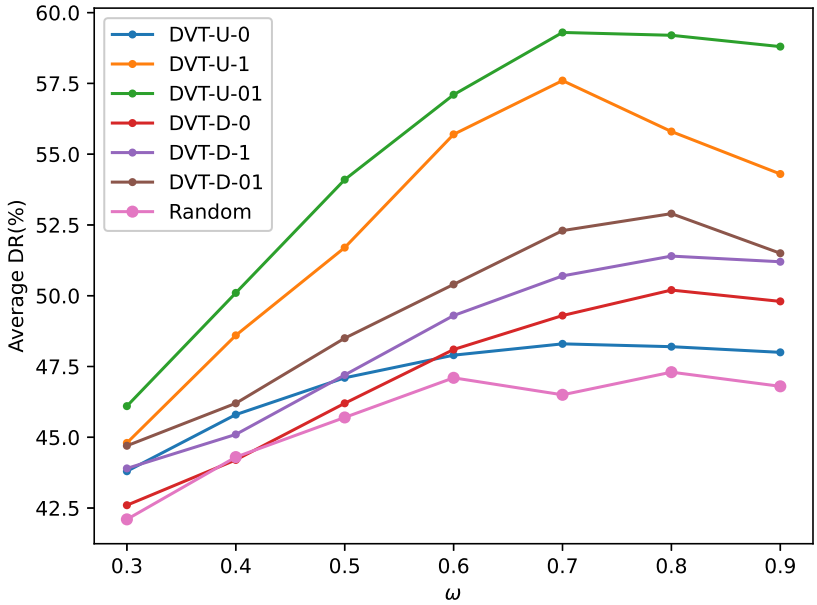

Figure 4.7: RNN(DVT) - The average detection rate under different ω using DVT as the TV function in RNN models

outperform the random selection. Among them, the 'DVT-U-01' strategy achieves the highest peak value of 59.4% at $\omega = 0.7$. On closer examination, we find that for the peak value, the U flag performs better than the D flag for s_1 and s_{01}, but shows the opposite trend for s_0. Furthermore, the S flag indicates that s_{01} consistently performs the best, followed by s_1.

Now we will compare the performance of these strategies under the MLP model. The results are shown in Figures 4.8 and 4.9.

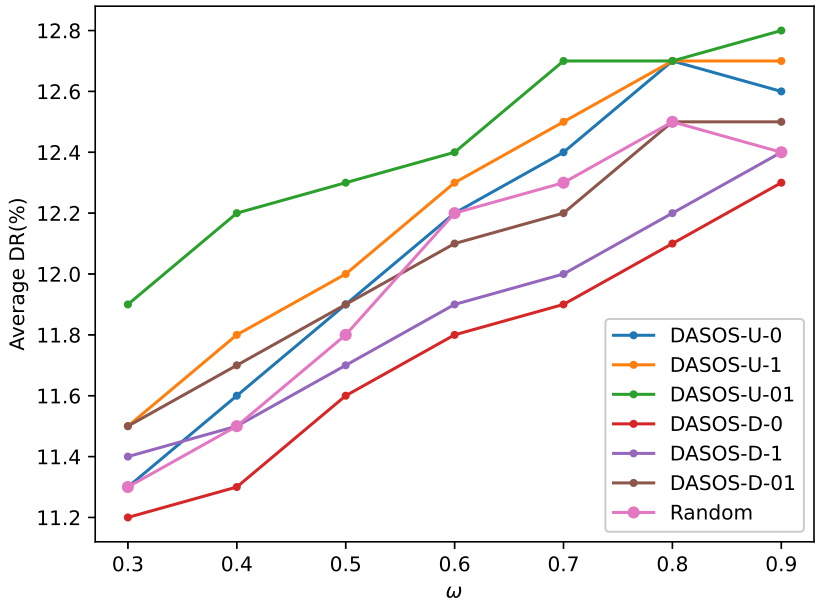

Figure 4.8: MLP(DASOS) - The average detection rate under diffrent ω using DASOS as the TV function in MLP models

In Figure 4.8, the results of the DASOS function show minimal variation from the random selection strategy. The largest difference is only about 1 percent, observed in the 'DASOS-U-01' strategy at $\omega = 0.9$. Similarly, in Figure 4.9, we find comparable findings, with the 'DVT-U-01' strategy achieving the best result, peaking at 16.5% when $\omega = 0.8$. The performance of the U flag is superior to the D flag, but the difference is not as pronounced as observed in the RNN models. The s_{01} flag remains the optimal choice.

Then, we also do comparison in LR models and the results are shown in Figures 4.10 and 4.11.

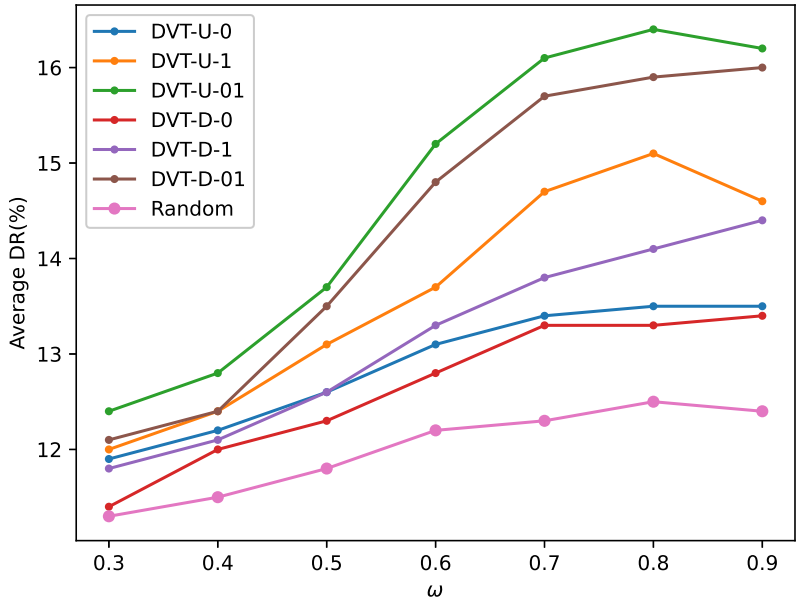

Figure 4.9: MLP(DVT) - The average detection rate under diffrent ω using DVT as the TV function in MLP models

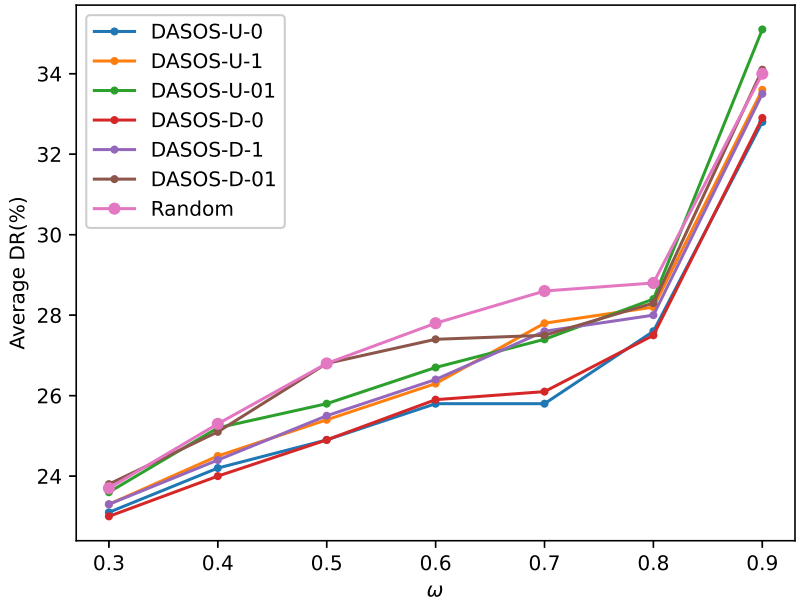

Figure 4.10: LR(DASOS) - The average detection rate under diffrent ω using DASOS as the TV function in LR models

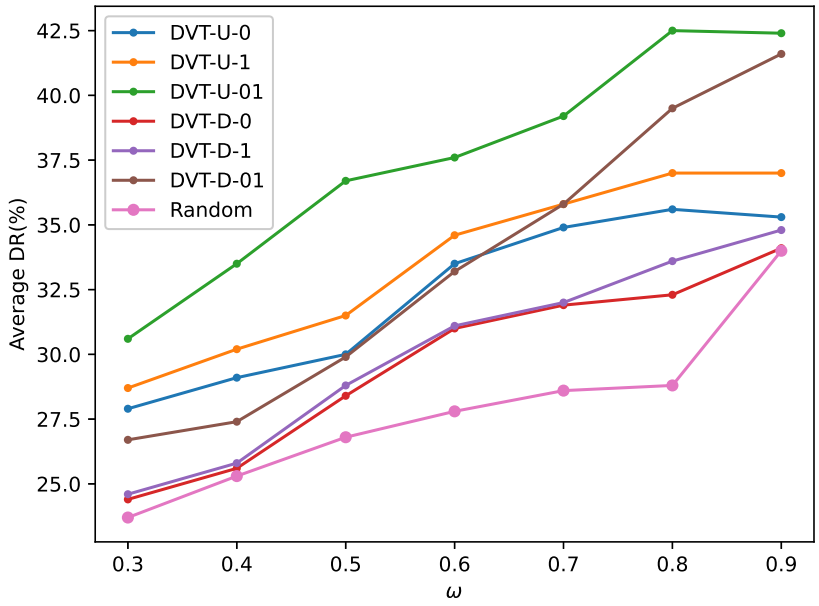

Figure 4.11: LR(DVT) - The average detection rate under diffrent ω using DVT as the TV function in LR models

Indeed, we can gather valuable insights regarding LR models from the results presented in Figure 4.10. The average detection rate consistently increases as ω increases. Unlike the other two models, there is a significant leap between $\omega = 0.8$ and $\omega = 0.9$, with the largest difference being about 5 percent, in contrast to the other leaps which are less than 1 percent. Moreover, similar to the MLP models, the DASOS function does not show significant improvements compared to random selection. Furthermore, the results exhibit the same trend as in MLP models. The highest average detection rate of 42.5% is achieved by the 'DVT-U-01' strategy at $\omega = 0.8$. To achieve good performance, it is essential to use the U flag and the s_{01} flag, with improvements of 3 percent and 5 percent, respectively.

Among all the results obtained from these three valid models, we can draw some conclusions. Firstly, the DASOS function does not demonstrate significant improvements in sample selection. Although the results are better than those without S-IDS, random selection can still compete with these strategies. On the other hand, the DVT function proves to be highly effective during the evaluation and consistently outperforms other approaches. It remains the best option, particularly when ω takes

on larger values, leading to the best results. Among the DVT function choices, the U flag and s_{01} are the optimal options. The U flag introduces more loss to the model for one sample, while s_{01} contributes more samples. Consequently, S-IDS built based on these choices exhibits the most significant difference from the IDS, thereby effectively interfering with the training of IDSGAN.

Furthermore, among these three models, the RNN-based S-IDS exhibits the most substantial increase in detection rate, ranging from 17.0% to a maximum of 59.3%. On the other hand, the LR-based and MLP-based S-IDS provide increases of up to 21.1% and 7.8%, respectively. This observation suggests that our framework appears to be particularly suitable for RNN models.

4.5 Performance of S-IDS Variants

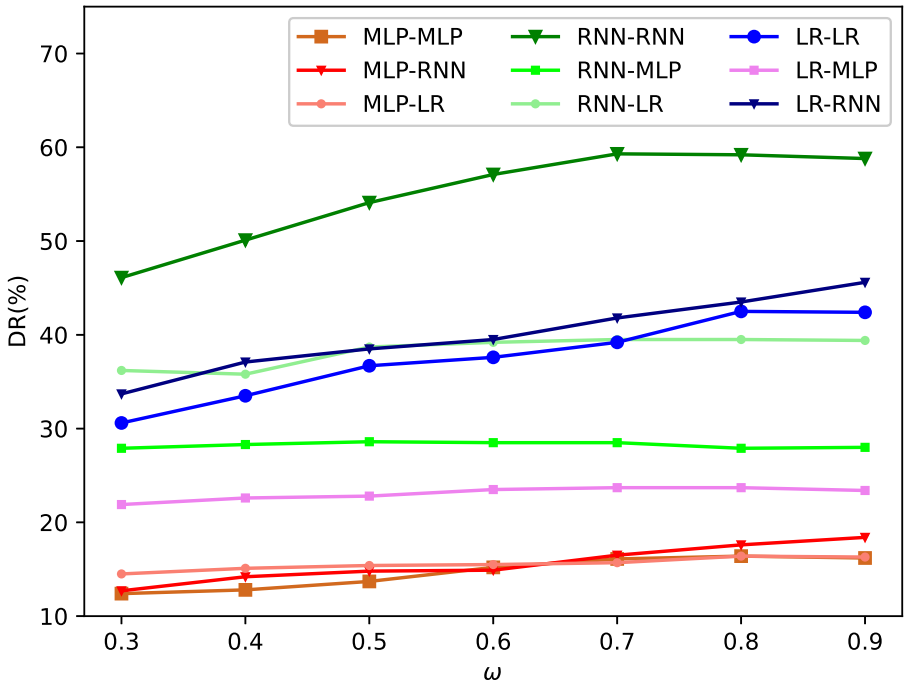

Figure 4.12: The detection rate of cross-model framework. A-B means using A as the IDS model and B as S-IDS model

In the last section, we made the assumption that the S-IDS and IDS share the same ML models. However, this may not always be the most optimal arrangement. To explore the best design, we employ the cross-model framework, where the IDS

and S-IDS use different models. For instance, we can build our IDS using the MLP model, while employing another RNN model to function as the S-IDS. In this part, we compare the LR, RNN, and MLP models as part of the cross-model evaluation, as depicted in Figure 4.12. Given the results obtained in the previous section, we select the trained S-IDS using the 'DVT-U-01' strategy and set $\omega \in \{0.3, \ldots, 0.9\}$ for this comparison.

Based on these results, we can draw some possible conclusions. For all types of IDS, the RNN model appears to be the best choice for the S-IDS. In the MLP-RNN framework, the detection rate reaches its highest value of 18.4% at $\omega = 0.9$. However, at lower values of ω, sometimes using LR models can yield better detection rates. In the LR-RNN framework, it consistently performs the best, achieving the highest detection rate of 45.6%, while the LR-LR framework only reaches 42.5%. Moreover, the RNN-RNN framework remains the best among all combinations, exhibiting the highest detection rate of 59.4%.

Chapter 5

Conclusion and Future Work

In this chapter, we will give our conclusions based on the results and analysis in the last chapter. Then, we will present more interesting findings during the whole evaluation and raise some possible future work.

5.1 Conclusion

In this paper, we introduce a new framework, using the Sophon IDS (S-IDS) to mislead the GAN training. In order to build the S-IDS, we present a simple strategy where we train the S-IDS by modifying the labels of some samples. Besides, we introduce some parameters in this strategy, including the distance definition, priority, selected labels and the noise percentage. By comparing the performances among them, we can choose the optimal parameters building the S-IDS. To evaluate it, we simulate the process of IDSGAN attacks on these S-IDSs and the original IDS. Three different ML models, Multilayer Perceptron (MLP), Recurrent Neural Network (RNN) and Logistic Regression (LR), are selected in the IDS and S-IDS training. The results demonstrate that the best S-IDS can significantly increase the detection rate in various scenarios: from 8.7% to 16.5% for the MLP model, 17.0% to 59.4% for the RNN model and 21.4% to 42.5% for the LR model. Furthermore, we investigate the performance of S-IDS when IDS and S-IDS adopt different ML models (e.g. RNN and MLP are adopted by IDS and S-IDS, respectively). Our experimental results indicate that S-IDS leads to the highest detection rate when RNN is employed by both IDS and S-IDS.

5.2 Future Work

Although we have shown that the S-IDS is able to increase the detection rate of the adversarial samples, there are also some existing issues not solved during our

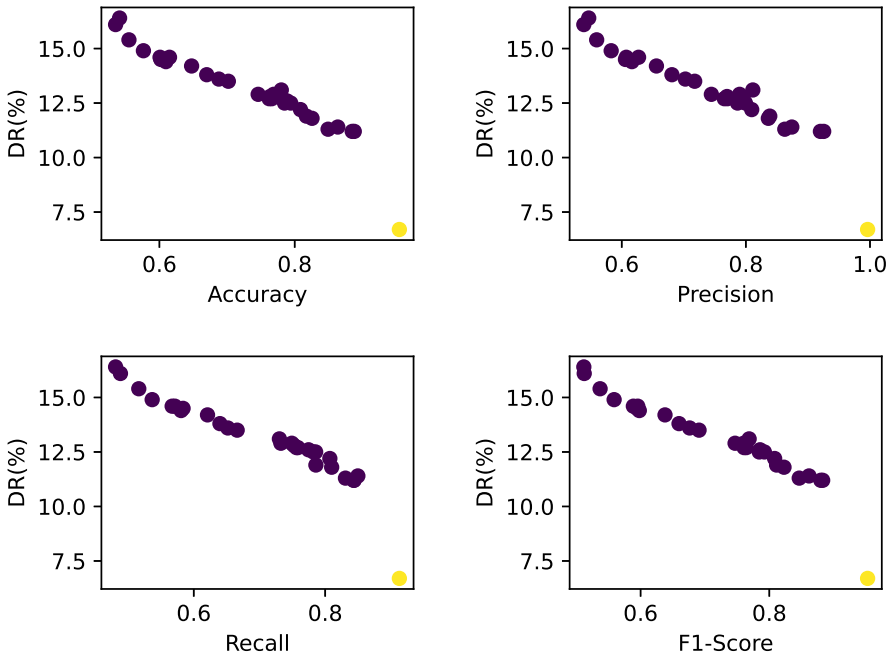

Figure 5.1: The relationship between detection rate and the statistics of S-IDS. The yellow node is the IDS as a comparison.

evaluation. We will discuss them in this part and give our new idea of the enhancement of either the S-IDS or IDSGAN.

5.2.1 S-IDS Statistics and Detection Rate

In the previous tests, we compared the relationship between the detection rate and the corresponding models and parameters. However, it is crucial to understand that the part that directly influences the GAN training is the S-IDS, even though it is trained based on these variables. Therefore, we need to investigate if there exists a functional relationship between the S-IDS statistics and the final detection rates. While we have mentioned that the S-IDS should be distinct from the IDS, it remains unclear which statistics would be the most promising. After conducting the evaluation, we select some of the trained S-IDS from the past tests and choose accuracy, precision, recall, and F1-score as the statistics of the S-IDS. We then plot these data points in Figures 5.1 and 5.2.

The results for the MLP-based S-IDS indicate that we need to minimize the four

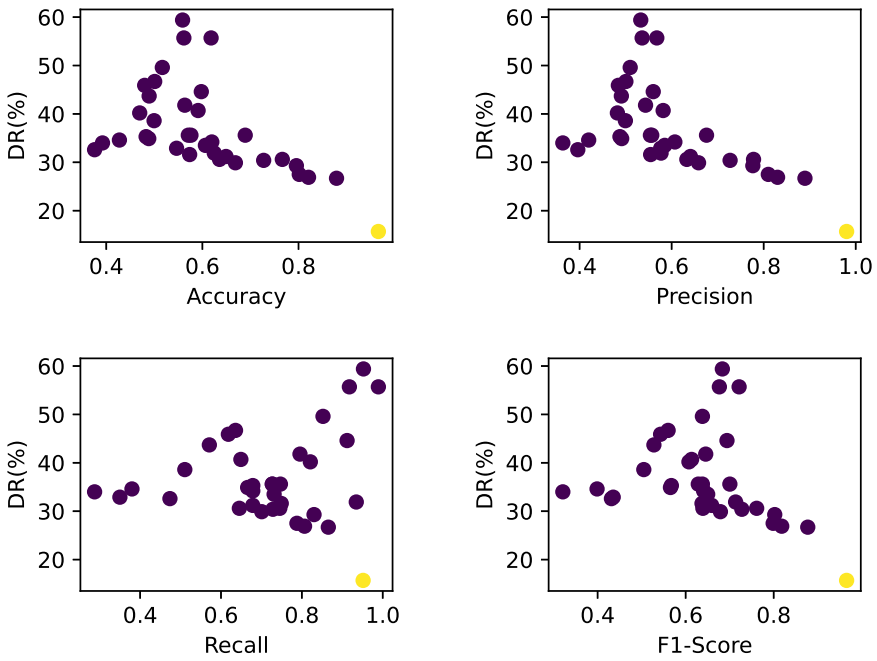

Figure 5.2: The relationship between detection rate and the statistics of S-IDS. The yellow node is the IDS as a comparison.

statistics as much as possible to achieve the best performance. However, the RNN-based S-IDS shows a different pattern. Here, the detection rate increases as the recall rises, while the accuracy and precision should be maintained around 0.5. At present, we do not have a clear understanding of why these two models exhibit such differences in behavior. However, understanding these differences is crucial if we aim to build an optimal framework.

Further investigation and analysis are necessary to uncover the underlying reasons for the contrasting behavior between MLP and RNN-based S-IDS. Once we understand these reasons, we can better design and optimize the framework for achieving the most effective performance.

5.2.2 Overtraining of IDSGAN

During the IDSGAN training, we employ a stop timing strategy based on the convergence of the detection rate. However, it is worth noting that the output of the

generator network, G, changes as the IDSGAN trains. As the training nears completion, the output becomes significantly smaller. To gain insights into the regulations during the IDSGAN training within the SophonIDS framework, built from the 'DVT-U-01' strategy with $\omega = 0.9$, we collect information on the loss, the detection rate of adversarial samples, and the index of G output as given in Eq. (5.1). Here, N represents the number of all continuous non-functional features, and n represents the number of samples. To handle cases where a feature has a value of 0, we replace it with 10^{-30} instead. We present the trends for the MLP, RNN, and LR models in Figures 5.3 and 5.4.

$$index = \frac{\sum_{i=1}^{n} \sum_{j=1}^{N} \log G(x_i^j)}{n * N} \tag{5.1}$$

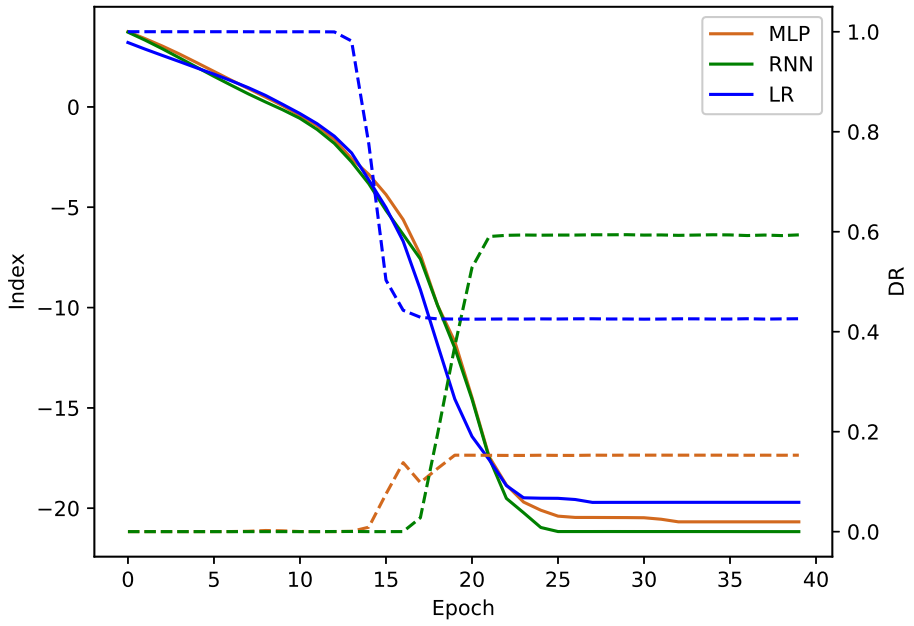

Figure 5.3: The trend of value index and detection rate as epoch increases in training dataset. The solid line is the index while the dashed line is the detection rate.

From the figures, we observe that the loss and the index of the output are both decreasing in all three models (MLP, RNN, and LR). Comparing the index of the original samples (-2) with the final G output, it becomes evident that the output is indeed very small. Although the flow is still valid, it is not the desired one. For attackers, the objective is to maintain traffic similar to the original, keeping functional features unchanged while controlling non-functional features within a common and

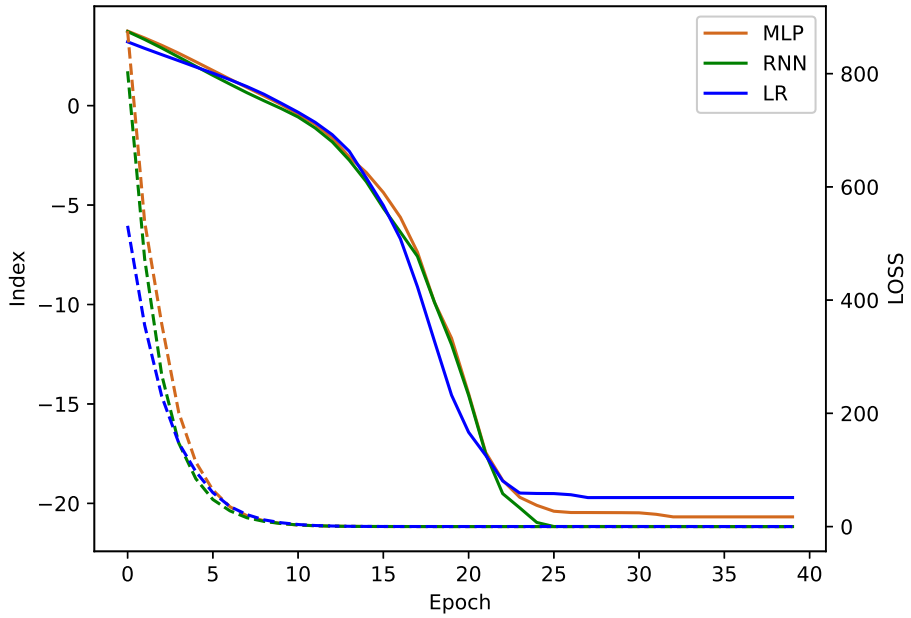

Figure 5.4: The trend of value index and loss as epoch increases in training dataset. The solid line is the index while the dashed line is the loss.

wide range. For instance, they might choose to stop the GAN training when $index \in [-5, -1]$. As indicated by the figure, when the epoch reaches 10, the loss is already low, and the value of the index is within a reasonable range, even though the detection rate has not yet converged. However, there is no standard criterion for judging the validity of adversarial samples, making it challenging to determine the exact timing.

Another option is to use the Adaptive Fast Gradient Sign Method (Ada-FGSM) as shown in Eq. (5.2). Instead of using the output of G directly, we multiply it by a small value ϵ and add it to the original sample, similar to how the FGSM works. However, this approach requires another loss function to update the parameters as well.

$$x^{adv} = x + \epsilon * G(x) \tag{5.2}$$

5.2.3 Further Variants of S-IDS

In this paper, we raise the concept of the S-IDS. To simplify the issue, we modify some labels of the dataset and train the model. Actually, this S-IDS is still a machine learning model while it does not have to be. To find out more possibilities, we analyse

the loss of generator and discriminator during the whole training process of IDSGAN. As the analysis is only a hypothesis, we put this part to the appendix.

In conclusion, the choices of S-IDS can be diverse, as long as the model can interfere B_{normal} and B_{mal}. We can use the S-IDS mentioned in the previous chapters, or the simple models from decision tree and random forest. Meanwhile, we can combine the advantages of different ML models to enhance the IDS design. For example, as MLP has the higher detecion rate of original malicious samples while RNN performs better in adversarial samples, we can update our IDS in Eq. (5.3). As long as either of the model predicts the flow as malicious, it will not be accepted. However, such design is in sacrifice of the error for the benign traffic.

$$x^{mal} \in \{x | \text{IDS}_{\text{RNN}}(x) == mal \quad or \quad \text{IDS}_{\text{MLP}}(x) == mal\} \tag{5.3}$$

In addition to those static models, we can also use dynamic models. The S-IDS can be updated using incremental learnings as it can learn more from the new samples [46]. Another option is that the fraud feedback is related to the number of each class in the previous predictions in order to control the percentage of B_{normal} and B_{mal} in one batch.

Meanwhile, the algorithm in the previous chapter is one example in the S-IDS framework. It is designed for the IDSGAN attacks as it can interfere the B_{normal} and B_{mal} in the loss function. However, many new attacking frameworks emerge and they have other special features, such as loss functions and training processes. In order to apply the concept of S-IDS to defend a specific attacking framework or in a different dataset, a new algorithm should be designed accordingly. Moreover, it is not guaranteed that S-IDS can be applied to every scenario. Possibly, no algorithm could have a reasonable performance for a certain task. In conclusion, we still need more research on the scenario and the algorithm before applying the S-IDS into the reality defence.

5.2.4 In-depth Theoretical and Practical Verification

In the methodology chapter, we provided an analysis of why S-IDS can help increase the detection rate for adversarial malicious samples, and we presented our label modification algorithm based on this analysis. However, we acknowledge that we cannot

claim with certainty that this is the optimal strategy. The complexity of machine learning algorithms makes it challenging to understand every detail inside them. To make it more practical and convincing, further research is needed.

One crucial area of research is the mathematical derivation of the modification rule. This could involve considering moments and samples for modification or exploring other factors not mentioned in this paper. By delving deeper into the mathematical aspects, we can gain a better understanding of the modification process and potentially enhance the framework's performance.

Furthermore, our evaluation was conducted on the CICIDS2017 dataset in this paper. It is essential to verify the framework's effectiveness on other datasets, such as NSL-KDD [12] and CSE-CIC-IDS2018 [13]. Additionally, exploring other domains, such as image identification datasets like MNIST-10 and ImageNet, could be valuable. By providing fraud feedback, the framework may pose challenges for attackers trying to generate adversarial samples and learn the data distribution. Our S-IDS framework can also be targeted for areas besides IDSGAN attacks in the anomaly-based IDS. It can be extended to the hybrid IDS, or other attacking strategies on IDS, or even not in the cybersecurity. It is believed to have a wide usage.

5.2.5 Measures Against Label Cleaning

In this framework, we use the S-IDS to interfere the IDSGAN attacks. A successful fraud information is to make the others believe what they get is the right one. Actually, we can always give the feedback of 'malicious' to the IDSGAN, it can also prevent the training. But that is easily detected by the attacker and they realize they are fooled. Therefore, we expect our S-IDS can give the feedbacks that still follow a possible data distribution and it would be better if the attacker is hard to know whether they are deceived. There are many papers that raise some algorithms cleaning the wrong labels in the dataset and most of them perform well [47] [48]. In our evaluation, we do not need to test whether our flipped dataset can be detected by these algorithms as the attackers actually do not know the existence of S-IDS. However, if this framework is already known, it is likely that they do a pre-attack. Based on the feedback, they can have a dataset and use label cleaning algorithms to find the wrong labels. After this, they could train their own IDS and train in a local

environment. In this way, our framework turns useless. Therefore, in order to avoid this, the S-IDS should also have the ability to resist the label cleaning.

Appendix A

An In-depth Analysis of the Loss of IDSGAN

Now, let us transform the loss function in IDSGAN. Given the black-box IDS model B, dataset X and the true labels y, we define the following two sets: $Y_{mal} = \{X_i | y_i = 1, i \in N\}$ and $Y_{normal} = \{X_i | y_i = 0, i \in N\}$. Also, $\bar{x}$ is the transformed generator input vector of sample x and $G_t(\bar{x})$ represents the adversarial sample for x. $\mathbb{E}$ represents the average function. According to the IDSGAN paper, we can have the generator (G) loss L_G and the adversarial samples A:

$$A = \{G(\bar{x}) | x \in Y_{mal}\}$$
$$L_G = \mathbb{E}_{x \in A} D(x)$$

During one batch, we can have a combination set C and two sets from IDS model B:

$$C = Y_{normal} \cup A$$
$$B_{mal} = \{x | x \in C, B(x) = 1\}$$
$$B_{normal} = \{x | x \in C, B(x) = 0\}$$

According to the loss function of discriminator D in IDSGAN, we can have two parts and the final loss L_D as:

$$L_D^N = \mathbb{E}_{x \in M_{normal}} D(x)$$
$$L_D^M = \mathbb{E}_{x \in M_{mal}} D(x)$$
$$L_D = L_D^N - L_D^M$$

Now, we define the following four sets:

$$P_a^n = \{x|x \in A, B(x) = 0\}$$
$$P_n^n = \{x|x \in Y_{normal}, B(x) = 0\}$$
$$P_a^m = \{x|x \in A, B(x) = 1\}$$
$$P_n^m = \{x|x \in Y_{normal}, B(x) = 1\}$$

Meanwhile, we introduce two statistics and the new loss:

$$R_a^n = \frac{len(P_a^n)}{len(P_n^n) + len(P_a^n)}$$
$$R_a^m = \frac{len(P_a^m)}{len(P_n^m) + len(P_a^m)}$$
$$L_D^O = \mathbb{E}_{x \in Y_{normal}} D(x)$$

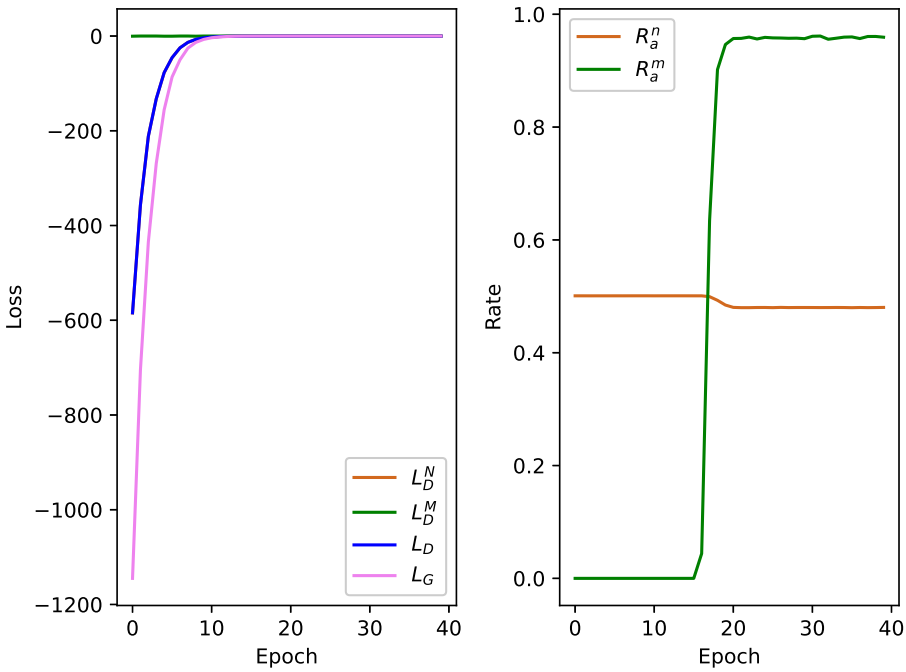

Figure A.1: (a) The curve of each loss during the training. (b) The curve of each rate during the training.

We take MLP-based IDS as the example. The curves for the above factors are shown in Figures A.1 and A.2. In the beginning of the training, L_D^M is almost at zero

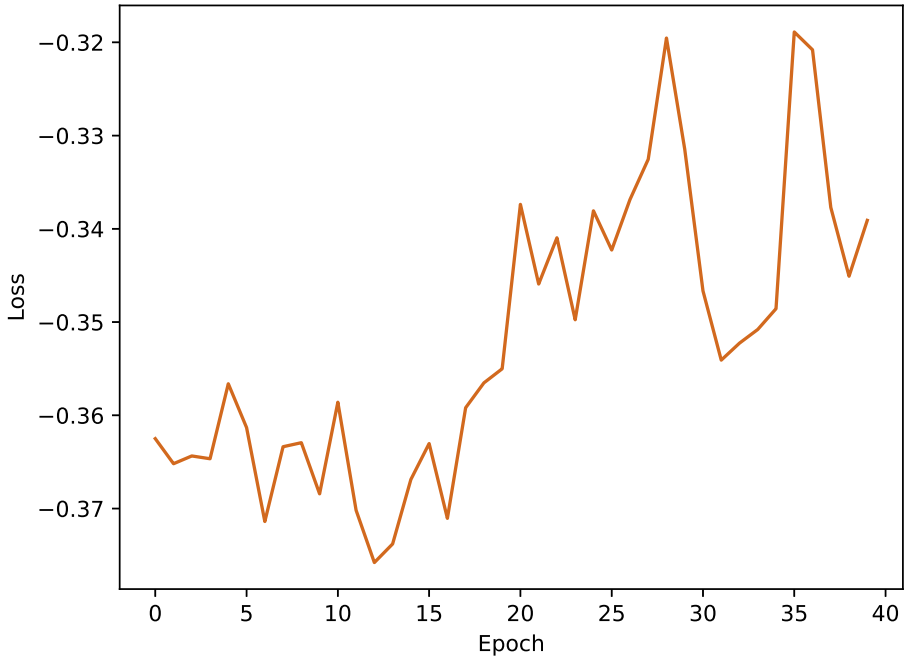

Figure A.2: The curve of L_D^O during the training.

while L_D^N has the large absolute value. Meanwhile, L_G is nearly twice the value of L_D^N. As we can see from the figure of rates, R_a^n is close to 0.5, which means half of the normal samples predicted by black-box IDS are from the original normal samples and the rest are from the adversarial samples. However, R_a^m is zero. That is to say, no adversarial samples are predicted as malicious. Then we analyse the curve for L_D^O shown in Figure A.2. The average outputs of original normal samples are at zero. That explains why L_D^M is almost at zero.

Now, we analyse how it helps us build the S-IDS. Firstly, We transform the definition of loss in D_t as:

$$L_D^N = \frac{len(P_a^n)}{len(P_n^n) + len(P_a^n)} * \mathbb{E}_{x \in P_a^n} D(x) + \frac{len(P_n^n)}{len(P_n^n) + len(P_a^n)} * \mathbb{E}_{x \in P_n^n} D(x)$$
$$L_D^M = \frac{len(P_a^m)}{len(P_n^m) + len(P_a^m)} * \mathbb{E}_{x \in P_a^m} D(x) + \frac{len(P_n^m)}{len(P_n^m) + len(P_a^m)} * \mathbb{E}_{x \in P_n^m} D(x)$$

By using the previous variables, we then simplify them into:

$$L_D^N = R_a^n * \mathbb{E}_{x \in P_a^n} D(x) + (1 - R_a^n) * \mathbb{E}_{x \in P_n^n} D(x)$$
$$L_D^M = R_a^m * \mathbb{E}_{x \in P_a^m} D(x) + (1 - R_a^m) * \mathbb{E}_{x \in P_n^m} D(x)$$

As shown in Figures A.1 and A.2, L_D^O is almost at zero while L_G has the larger absolute value, we can omit the effect of the normal samples in original dataset. Meanwhile, if we suppose $\mathbb{E}_{x \in P_a^m} D(x) \approx \mathbb{E}_{x \in P_a^n} D(x) \approx L_G$, then we can have:

$$L_D \approx (R_a^n - R_a^m) * L_G$$

Now we find out that the L_D is proportional to $R_a^n - R_a^m$. If we can try to keep $|R_a^n - R_a^m|$ in a small value for the beginning epochs, we can minimize the L_D as well. That is to say, D has less ability to learn from this epoch and improves itself. Even if G can update the parameters, we have already break their cross-learning process. D cannot give a updated output for the adversarial samples in the new epoch and G is not able to update in the proper way.

In our algorithm, we actually decrease the value of P_n^n / P_n^m and eventually decrease $|R_a^n - R_a^m|$. In fact, for the ML models, as long as we can decrease the accuracy, the detection rate of adversarial samples will rise. This also explains why MLP-based model has the best accuracy but the worst detection rate of adversarial samples from IDSGAN.

Ideally, if we can control $|R_a^n - R_a^m|$ around 0, the performance should be much better. We may need another strategy to build the S-IDS by collecting the characteristics of adversarial samples.

Bibliography

[1] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, oct 2020. ISSN 0001-0782. doi: 10.1145/3422622.

[4] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[5] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[8] Dorothy E Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987.

[9] Koral Ilgun and A Ustat. A real-time intrusion detection system for unix. *University of California Santa Barbara Master Thesis*, 1992.

[10] Jiahai Yang, Peng Ning, X Sean Wang, and Sushil Jajodia. Cards: A distributed system for detecting coordinated attacks. In *Information Security for Global Information Infrastructures: IFIP TC11 Sixteenth Annual Working Conference on Information Security August 22–24, 2000, Beijing, China 15*, pages 171–180. Springer, 2000.

[11] Wenjuan Li, Steven Tug, Weizhi Meng, and Yu Wang. Designing collaborative blockchained signature-based intrusion detection in iot environments. *Future Generation Computer Systems*, 96:481–489, 2019.

[12] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009. doi: 10.1109/CISDA.2009.5356528.

[13] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[14] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 12–26 vol.2, 2000. doi: 10.1109/DISCEX.2000.821506.

[15] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '11, page 29–36, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307680. doi: 10.1145/1978672.1978676.

[16] Benjamin Sangster, T. J. O'Connor, Thomas Cook, Robert Fanelli, Erik Dean, William J. Adams, Chris Morrell, and Gregory Conti. Toward instrumenting network warfare competitions to generate labeled datasets. In *Proceedings of the 2nd Conference on Cyber Security Experimentation and Test*, CSET'09, page 9, USA, 2009. USENIX Association.

[17] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of data mining in computer security*, pages 77–101, 2002.

[18] ShengYi Jiang, Xiaoyu Song, Hui Wang, Jian-Jun Han, and Qing-Hua Li. A clustering-based method for unsupervised intrusion detections. *Pattern Recognition Letters*, 27(7):802–810, 2006.

[19] Yu-Xin Meng. The practice on using machine learning for network anomaly intrusion detection. In *2011 International Conference on Machine Learning and Cybernetics*, volume 2, pages 576–581. IEEE, 2011.

[20] Sumouli Choudhury and Anirban Bhowal. Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. In *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pages 89–95. IEEE, 2015.

[21] Yalei Ding and Yuqing Zhai. Intrusion detection system for nsl-kdd dataset using convolutional neural networks. In *Proceedings of the 2018 2nd International conference on computer science and artificial intelligence*, pages 81–85, 2018.

[22] Tuan A Tang, Lotfi Mhamdi, Des McLernon, Syed Ali Raza Zaidi, and Mounir Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 202–206. IEEE, 2018.

[23] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[25] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[26] Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the automatic anime characters creation with generative adversarial networks. *arXiv preprint arXiv:1708.05509*, 2017.

[27] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[28] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[29] Zheng Wang. Deep learning-based intrusion detection with adversaries. *IEEE Access*, 6:38367–38384, 2018. doi: 10.1109/ACCESS.2018.2854599.

[30] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[31] Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. In *Pacific-asia conference on knowledge discovery and data mining*, pages 79–91. Springer, 2022.

[32] Ravi Chauhan and Shahram Shah Heydari. Polymorphic adversarial ddos attack on ids using gan. In *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, 2020. doi: 10.1109/ISNCC49221.2020.9297264.

[33] Vikash Kumar and Ditipriya Sinha. Synthetic attack data generation model applying generative adversarial network for intrusion detection. *Computers & Security*, 125:103054, 2023. ISSN 0167-4048. doi: 10.1016/j.cose.2022.103054.

[34] Simon Msika, Alejandro Quintero, and Foutse Khomh. Sigma: Strengthening ids with gan and metaheuristics attacks. *arXiv preprint arXiv:1912.09303*, 2019.

[35] JooHwa Lee and KeeHyun Park. Gan-based imbalanced data intrusion detection system. *Personal and Ubiquitous Computing*, 25(1):121–128, Feb 2021. ISSN 1617-4917. doi: 10.1007/s00779-019-01332-y.

[36] Paulo Freitas de Araujo-Filho, Mohamed Naili, Georges Kaddoum, Emmanuel Thepie Fapi, and Zhongwen Zhu. Unsupervised gan-based intrusion detection system using temporal convolutional networks and self-attention. *IEEE Transactions on Network and Service Management*, pages 1–1, 2023. doi: 10.1109/TNSM.2023.3260039.

[37] Xinying Guo, Chunhua Zhu, Jing Yang, and Yan Xiao. An anomaly detection model for ads-b systems based on improved gan and lstm networks. In *2021 IEEE 21st International Conference on Communication Technology (ICCT)*, pages 802–809, 2021. doi: 10.1109/ICCT52962.2021.9658039.

[38] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International conference on artificial neural networks*, pages 703–716. Springer, 2019.

[39] Cao Phan Xuan Qui, Dang Hong Quang, Phan The Duy, Do Thi Thu Hien, and Van-Hau Pham. 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, 2021. doi: 10.1109/RIVF51545.2021.9642111.

[40] Phan The Duy, Le Khac Tien, Nghi Hoang Khoa, Do Thi Thu Hien, Anh Gia-Tuan Nguyen, and Van-Hau Pham. Digfupas: Deceive ids with gan and function-preserving on adversarial samples in sdn-enabled networks. *Computers & Security*, 109:102367, 2021. ISSN 0167-4048. doi: 10.1016/j.cose.2021.102367.

[41] Iyatiti Mokube and Michele Adams. Honeypots: Concepts, approaches, and challenges. In *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, page 321–326, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936295. doi: 10.1145/1233341.1233399.

[42] Muhammet Baykara and Resul Das. A novel honeypot based security approach for real-time intrusion detection and prevention systems. *Journal of Information Security and Applications*, 41:103–116, 2018. ISSN 2214-2126. doi: 10.1016/j.jisa.2018.06.004.

[43] A Umamaheswari and B Kalaavathi. Honeypot tb-ids: trace back model based intrusion detection system using knowledge based honeypot construction model. *Cluster Computing*, 22:14027–14034, 2019.

[44] Liu Dongxia and Zhang Yongbo. An intrusion detection system based on honeypot technology. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 1, pages 451–454, March 2012. doi: 10.1109/ICCSEE.2012.158.

[45] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of tor traffic using time based features. In *International Conference on Information Systems Security and Privacy*, 2017.

[46] Gido M. van de Ven, Tinne Tuytelaars, and Andreas S. Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, Dec 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00568-3.

[47] Mélanie Bernhardt, Daniel C Castro, Ryutaro Tanno, Anton Schwaighofer, Kerem C Tezcan, Miguel Monteiro, Shruthi Bannur, Matthew P Lungren, Aditya Nori, Ben Glocker, et al. Active label cleaning for improved dataset quality under resource constraints. *Nature communications*, 13(1):1161, 2022.

[48] Stefano Teso, Andrea Bontempelli, Fausto Giunchiglia, and Andrea Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34:12966–12977, 2021.