Comprehensive Botnet Detection by Mitigating Adversarial Attacks, Navigating the Subtleties of Perturbation Distances and Fortifying Predictions with Conformal Layers

Rahul Yumlembam[a], Biju Issac[a,*], Seibu Mary Jacob[b], Longzhi Yang[a]

[a]*Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne, NE1 8ST, England, UK*

[b]*School of Computing, Engineering & Digital Technologies, Teesside University, Middlesbrough, TS1 3BX, England, UK*

Abstract

Botnets are computer networks controlled by malicious actors that present significant cybersecurity challenges. They autonomously infect, propagate, and coordinate to conduct cybercrimes, necessitating robust detection methods. This research addresses the sophisticated adversarial manipulations posed by attackers, aiming to undermine machine learning-based botnet detection systems. We introduce a flow-based detection approach, leveraging machine learning and deep learning algorithms trained on the ISCX and ISOT datasets. The detection algorithms are optimized using the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) to obtain a baseline detection method. The Carlini & Wagner(C&W) attack and Generative Adversarial Network (GAN) generate deceptive data with subtle perturbations, targeting each feature used for classification while preserving their semantic and syntactic relationships, which ensures that the adversarial samples retain meaningfulness and realism. An in-depth analysis of the required L2 distance from the original sample for the malware sample to misclassify is performed across various iteration checkpoints, showing different levels of misclassification at different L2 distances of the Pertrub sample from the original sample. Our work delves into the vulnerability of various models, examining the transferability of adversarial examples from a Neural Network surrogate model to Tree-based algorithms. Subsequently, models that initially misclassified the perturbed samples are retrained, enhancing their resilience and detection capabilities. In the final phase, a conformal prediction layer is integrated, significantly rejecting incorrect predictions — 58.20% in the ISCX dataset and 98.94% in the ISOT dataset.

Keywords: NIDS, C&W attack, Botnet detection, Machine learning, Conformal Prediction

*Corresponding author

Email address: bissac@ieee.org (Biju Issac)

1. Introduction

Cybercriminals can infect an organization's computing device or machine with a 'bot binary' executable using traditional attack techniques such as viruses and worms distributed through user downloads and email links. The bot binary runs silently in the background of a user machine, turning it into a 'zombie' waiting for commands from a Command and Control (C&C) server controlled by the botmaster or another bot. A botmaster controls a group of zombies and forms a botnet to perform distributed computing tasks [1]. Cybercriminals have shifted in favour of botnet usage, with a recent report recording over 10,000 C&C servers were added to the blocking lists [4]. For instance, in 2016, a DDoS attack carried out on DNS provider Dyn using the Mirai Internet of Things (IoT) Botnet caused users of the DNS service to have issues resolving domain names, ultimately causing many well-known sites to become inaccessible [5]. Cybercriminals frequently resort to botnets to earn profits, mainly through Cybercrime-as-a-Service (CaaS), where they rent out parts of a botnet to clients [7]. Botmasters must ensure that their bot evades the Intrusion Detection Systems (IDSs) to maintain operational persistence, allowing them to gain buyers' trust. Doing so increases the potential for botnet growth, attracting higher profits from buyers requiring a large amount of cumulative bandwidth and processing power.

Historically, the first line of defence against such threats has been Network Intrusion Detection Systems (NIDS). Traditional signature-based NIDS has revealed shortcomings, particularly in identifying novel or modified botnets. Consequently, a recent shift has been towards leveraging machine learning algorithms within NIDS. Machine learning systems promise to detect known threats and unearth previously unknown or zero-day threats. However, as with most advancements, this shift has vulnerabilities. Recent research underscores the susceptibility of machine learning models to adversarial attacks, especially evasion attacks. These manipulations create deceptive instances that could pass undetected, presenting a significant challenge. While there has been extensive research on adversarial attacks in fields like computer vision, the implications for NIDS still need to be explored more in detail. The features used for training the machine learning-based NIDS system are central to its effectiveness. Therefore, understanding and assessing the vulnerabilities that arise when adversaries manipulate these features is paramount. However, the extent to which adversaries can exploit these features and the magnitude of feature-based perturbations needed to compromise NIDS remains unexplored and, therefore, raises pressing questions: How susceptible are these features to adversarial alterations? Can subtle feature manipulations successfully bypass a sophisticated machine-learning detector? In this research, we delve into these questions, aiming to discover the vulnerabilities of each feature used in the training. A common approach to generating adversarial examples involves using a surrogate model. When we choose a neural network as the surrogate model, an essential question arises: Will these adversarial examples maintain their deceptive potency against different architectures? In this research, we aim to find out the extent of transferability of the adversarial samples.

First, we train both our machine learning and deep learning algorithms. To ensure the optimal performance of these models, we employ Genetic Algorithms (GA) and Particle Swarm Optimisation (PSO) to fine-tune the hyperparameters meticulously. This process

not only aids in establishing a solid foundation but also in achieving the best predictive accuracy possible for our models.

In the next step, we focus on the individual features we use to train the machine learning model. To alter these features, we utilize the Carlini & Wagner (C&W) and Generative Adversarial Network (GAN) attacks. However, we maintain semantic and syntactic relationships during this attack by adjusting dependent features concerning the primary attack feature. By doing this, we seek to identify and understand potential vulnerabilities that might emerge when these features undergo manipulation. Moreover, having crafted these adversarial examples with a neural network as the surrogate model, we probe their efficacy against other machine learning architectures. Specifically, we test the transferability of these adversarial samples to different architectures, such as decision trees and random forests. This step is pivotal, as it helps us gauge the breadth of the threat, determining whether adversarial samples designed for one model can compromise another. Upon identifying potential vulnerabilities, we collect the adversarial samples. We then integrate these samples into our training process, further refining our models and enhancing their resilience against adversarial inputs.

Finally, this work introduces conformal prediction grounded in robust statistical principles. It offers a framework that makes predictions and provides a valid measure of certainty for each prediction. We can reject uncertain instances and accept only the classifications of confident ones, thereby offering an opportunity to investigate the rejected instances instead of making an incorrect prediction. The need for reliable prediction is paramount in domains such as NIDS. However, with evolving cyber threats and sophisticated adversarial tactics, traditional predictive models occasionally grapple with instances of uncertainty. The repercussions of false positives or negatives in a high-stakes environment like cybersecurity can be dire. Hence, there is an urgent need for methods that predict and quantify the confidence in these predictions.

In summary, the regular and novel contributions of this paper are as follows:

- Optimised Model Training: We have taken a standard yet Discriminatoral approach in employing Genetic Algorithms (GA) and Particle Swarm Optimisation (PSO) to fine-tune the hyperparameters of our machine learning and deep learning models, which ensures we achieve competitive predictive accuracy with other works.

- In-depth Feature Vulnerability Analysis: Our research thoroughly examines the individual features utilized in training the machine learning model using C&W and GAN attacks. We meticulously adjust dependent features with the primary attack feature to uncover potential vulnerabilities in the training features, making a novel contribution to feature analysis within machine learning models.

- Transferability Examination: In a novel exploration, we extend our work's impact by investigating how adversarial examples, designed using neural networks, might perform against different architectural paradigms, such as decision trees and random forests. This inquiry provides new insights and an understanding of model robustness and transferability.

- Introduction of Conformal Prediction in NIDS: We make a novel contribution by integrating and analyzing conformal prediction within Network Intrusion Detection Systems (NIDS). Utilizing this robust statistical principle-based approach, we offer a predictive framework that provides valid measures of prediction certainty and can reject uncertain instances.

2. Related Ideas and Work

Botnets often communicate with Command and Control (C&C) servers or other bots through network interactions. In order to identify such communications, Network Intrusion Detection Systems (NIDS) like Snort are deployed to identify them. These systems use signature-based detection to check each network packet against predefined signatures. A signature-based detection system matches predefined signatures to each packet's signature and generates an alert if a match is found, deeming it as a malicious behaviour [8]. However, this requires analysis of every network packet, which is computationally intensive [9] and shown in research to suffer from a large proportion of packet drops when saturated at higher network speeds [10]. Furthermore, experimentation shows that Snort's false positive rate (FPR) can be high, with the default rule sets, rendering it challenging to analyze or trust the alerts [10]. On the other hand, as these detection methods mature, bot developers innovate, too, crafting ways to dodge detection and ensure their bots remain stealthily active. Techniques include encrypting communication payloads and fragmenting packets. In sophisticated techniques like the polymorphic blending attack, a bot first understands the typical traffic profile of a network, it then mimics this profile when communicating, making activities blend seamlessly with regular traffic. In response to these challenges, machine learning is increasingly employed in NIDS. The advantage of machine learning-based approaches is their ability to learn and recognize complex patterns, often surpassing traditional signature-based methods. Such systems can adapt to evolving threats, reducing false positives and increasing detection rates. They are especially effective when previously unknown or zero-day threats emerge, as they can detect anomalies without relying on predefined signatures.

Recent work such as Chen et al. explored conversational features within the CTU-13 botnet dataset scenarios [12]. They utilized multiple classifiers, such as Decision Tree, BayesNet, and the Random Forest Classifier. Velasco-Mata et al. delved into feature selection by employing the Information Gain and Gini Importance techniques [13]. Their efforts resulted in three pre-selected subsets containing five to seven features. On evaluating these subsets with Decision Tree, Random Forest, and k-Nearest Neighbors models, the Decision Trees with a five-feature set emerged as the top performers, achieving an impressive mean F1 score of 85%. Dollah et al. focused their investigation on the detection of HTTP-based botnets [14]. They curated a labelled dataset by merging botnet traffic with genuine web browsing traffic. Their evaluation metrics spanned Accuracy, Precision, Recall, and FPR while utilizing classifiers like Decision Tree, k-nearest Neighbour, Naïve Bayes, and Random Forest. Remarkably, the k-Nearest Neighbours classifier stood out, registering an average accuracy of 92.93%. Decision Tree's performance was superior to the Random Forest for the HTTP datasets they employed. Haddadi et al. Discriminatorally evaluated the per-

formance of renowned IDS tools, Snort and BotHunter, against subsets from the CTU-13 dataset scenarios [15]. Incorporating the C4.5 Decision Tree algorithm as their machine learning classifier, they experimented with feature sets derived from Tranalyzer and Argus network flow extraction tools. The Argus tool's features achieve an average DR of up to 99.45%. Multilayer Transformer encoder and deep neural network is utilized in [38]. The Transformer encoder is utilized for encoding the implicit semantic relationship between the traffic bytes of the botnet and the deep learning network to capture the spatial relationship between the traffic bytes of the botnet, achieving 91.92% detection accuracy on the ISCX-2014 dataset. BotMark [39] utilizes 15 statistical flow-based features and three graph-based features, applying k-means clustering to assess C-flows' similarity and leveraging least-square techniques and Local Outlier Factor (LOF) to evaluate graph-based anomalies. The model was tested with simulated network traffic from five recent botnets, including Mirai and Zeus, in a real computing environment, achieving an impressive 99.94% detection accuracy. Shahhosseini et al. [31] proposes to extract only the header parts of packets, specifically from the transport (TCP/UDP) and network (IP) layers, avoiding the payload to ensure privacy and to handle the increasing use of encrypted traffic which makes the payload-based analysis less effective. The feature extraction is done using a Long Short Term Memory (LSTM) and uses a Random Forrest classifier to classify the botnets. Negative Sampling Algorithm Based on an Artificial Immune System is used in [40] to reduce dimensionality by focusing on features that signify abnormal behaviour, effectively filtering out normal data, after which the data are scaled and using CNN and LSTM to classify the network traffic into Botnet or Normal.The Self-Attention mechanism is introduced in [30] where it uses Convolutional Block Attention Module (CBAM)-ResNet for spatial features and aids in understanding the sequence and temporal context. DCNNBiLSTM [42] utilizes a hybrid deep learning model combining Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) networks to detect botnet in Edge IoT devices. CNN layers first extract spatial features from the input data, which are then passed through BiLSTM layers that analyze temporal dependencies and sequences within the data. Similar work in HDLNIDS [43] employs a deep-learning framework combining convolutional neural networks (CNNs) and recurrent neural networks (RNNs) where CNN is used to extract spatial features and RNN to analyze temporal features. Gunupdi et al. utilize [44] Deep Residual Convolutional Neural Network (DCRNN) with Novel Binary Grasshopper Optimization Algorithm (NBGOA) as feature selector and Improved Gazelle Optimization Algorithm to search hyperparameter of the algorithm to reduce high false alarm rate of NIDS.

Machine learning and deep learning solutions have accurately detected malicious network flows. However, these algorithms have vulnerabilities, especially when faced with carefully crafted adversarial samples. This vulnerability is well-acknowledged in the field. Although there is a high output of scholarly activities from 2018 onwards on the topic of Intrusion Detection [3], a systematic review that explains the landscape of data poisoning attacks shows there is a lack of work that explicitly focuses on data poisoning attack [2]. The challenge is even more pronounced in the context of Network Intrusion Detection Systems (NIDS). Unlike images, where adversarial perturbations might be straightforward, NIDS requires preserving syntactic and semantic relationships in the data. For instance, if one

were to alter the 'duration' feature in a feature vector, a corresponding change in the 'rate' feature would be necessary to maintain coherence. Recent research has explored methods to generate adversarial features. Notably, the studies in [19] and [22] have employed Generative Adversarial Networks (GANs) to craft adversarial samples. Following the generation of these samples, the works in [19], [22], and [23] have utilized Particle Swarm Optimization (PSO) to automate the search for optimal traffic mutants leveraging the capabilities of PSO. It has been observed in a recent study [46] that even small perturbations in network traffic can bypass NIDS protections if strategically introduced. The study highlights that this can be achieved without compromising the core logic of the botnet attack. To ensure that the semantic relationship remains intact while modifying the feature vector, the approach taken in [46] involves using a mask. Additionally, they employ a sign-based adversarial attack, commonly called the 'sign attack'. Debicha et al. [45] leverages multiple strategically positioned adversarial detectors to improve detection rates over traditional single detector setups. Various transfer learning techniques, such as Feature Extraction and Fine-Tuning, as well as fusion rules like majority voting, Bayesian averaging, and Dempster-Shafer theory, are utilized, showcasing that a parallel IDS design with multiple detectors can better manage adversarial attacks. The practical feasibility of adversarial evasion attacks against machine learning-based NIDS is explored in Adv-Bot [46], demonstrating the potential to mislead these systems with strategically crafted adversarial botnet traffic. In their 2024 study, Roshan et al. [47] assess the vulnerability of machine learning-based NIDS to white-box adversarial attacks and suggest heuristic defence strategies to increase system robustness. Employing advanced adversarial attack techniques like FGSM, JSMA, PGD, and C&W, the study evaluates NIDS robustness. Meanwhile, Mohammadian et al. (2023) [49] focus on the susceptibility of deep learning-based NIDS to adversarial attacks, introducing a gradient-based method that identifies crucial features for creating adversarial perturbations using the Jacobian Saliency Map. This technique reduces the number of features needed, optimizing attack efficiency. Furthering this research, Roshan et al. (2023) [50] deliver a detailed analysis of the efficacy of deep learning models in defending against adversarial attacks on network intrusion detection systems by employing robust adversarial attack algorithms and adversarial training as a defence strategy. Kumar et al. (2024) [51] address vulnerabilities due to adversarial examples in DL/ML-based NIDS, introducing a variational autoencoder to create adversarial examples that not only evade detection but also comply with network security constraints. This approach achieves a 64.8% success rate in evading DL/ML-based IDS. Lastly, a novel framework that integrates a Weighted Conditional Stepwise Adversarial Network (WCSAN) with Particle Swarm Optimization (PSO) is proposed by Barik et al. (2024) [52]. This framework enhances IDS by employing advanced feature selection techniques such as PCA and LASSO, improving detection accuracy through optimized adversarial training.

The framework across these studies introduces perturbations while preserving the network traffic's semantic and syntactic relationships. The extent of perturbation required to achieve desired outcomes has yet to be exhaustively analyzed in prior research. We aim to bridge this gap by delving deep into the nuances of perturbation levels in malware samples and evaluating their corresponding evasion capabilities. Furthermore, many adversarial

samples are crafted using the gradient information from surrogate models. However, the transferability of these samples to non-gradient-based methods, such as Random Forests or Decision Trees, remains largely uncharted territory. We aim to shed light on this aspect by analyzing the transferability of generated adversarial samples to models without gradient information. The need for reliable and trustworthy prediction systems in malware detection systems is becoming increasingly important. Recent work in the field of Android malware classification, such as the study by Barbero et al. [24], has begun to explore the integration of conformal prediction to discard uncertain predictions, showcasing the potential of this methodology in enhancing prediction reliability. However, there is a notable gap in the current literature and application regarding integrating conformal prediction within the classification of network flow data. Our research aims to bridge this gap, shedding light on the potential of conformal prediction.

3. Preliminaries

This section describes the dataset, the feature extraction technique used, and the process of classification of network flow into Malware and Benign.

3.1. Datasets

In this research, we employed two distinct datasets: ISOT [36] and ISCX[37]. The ISOT dataset is a merged dataset of malicious and normal traffic datasets [17]. The malicious traffic from the French chapter of the Honeynet Project comprises activity from the Storm, Waledac and Zeus botnets. The normal background traffic datasets are the product of merged datasets from Ericsson Research and the Lawrence Berkeley National Laboratory. This data includes traffic from HTTP web browsing, gaming and P2P clients. The ISCX Botnet dataset includes network traffic from a range of botnets such as Neris, Rbot, Virut, NSIS, SMTP, Zeus and normal activity traffic, which are captured by replaying over a network testbed topology[18]. The ISOT dataset has been divided into two parts, with a 70/30 split for training and testing purposes, respectively. From the training portion, we further reserve 5% as a validation set to optimize the performance of the machine learning models. Similarly, for the ISCX dataset, although dedicated training and testing files were provided in the original publication, we also reserve 10% of the training set as a validation set. We tune all our model's hyperparameters using the validation set. In our experiment, the random seed is set to 42 to ensure the experiments are reproducible.

3.2. Feature Extraction and Feature Selection

In order to work with the ISOT and ISCX datasets, which are only available as .pcap files, feature extraction into network flow is necessary. Following the method used in [25], we extracted 32 features and their corresponding labels. The extracted features are in Table 1. The complete descriptions of all the features for both the base and extended datasets can be found in the 'ra' man page [26].

We employed the Information Gain (IG) metric to determine the most relevant features. Information Gain (IG) is a statistical measure used in machine learning and information

Table 1: Extracted features [26]

Features	Features	Features	Features
SrcAddr	dTtl	DstAddr	TcpRtt
Proto	SynAck	Sport	AckDat
Dport	SrcPkts	State	DstPkts
sTos	SAppBytes	dTos	DAppBytes
SrcWin	Dur	DstWin	TotPkts
sHops	TotBytes	dHops	Rate
LastTime	SrcRate	sTtl	Label

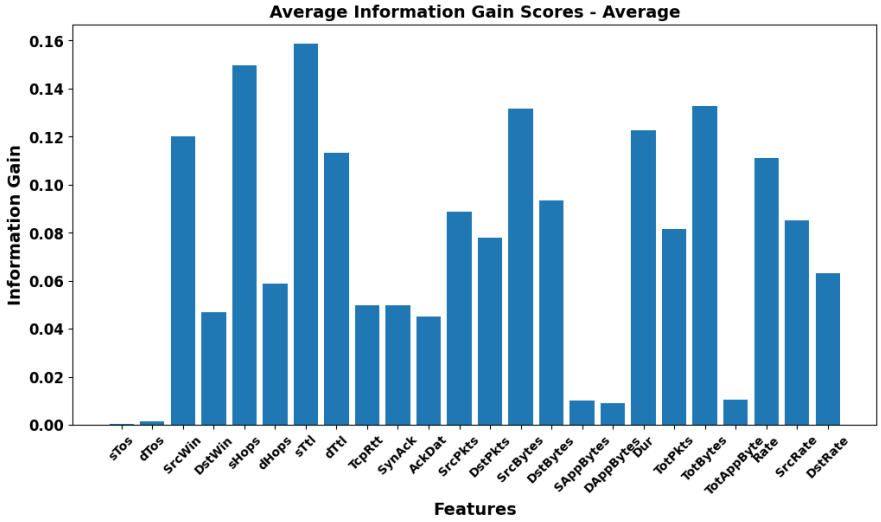

Figure 1: The average information gain of each feature, calculated from both the ISCX and ISOT datasets.

theory to determine how valuable a feature (or attribute) is in predicting the outcome or class of a given dataset. It measures the reduction in uncertainty or entropy achieved by partitioning a dataset based on that feature. In simpler terms, it quantifies the "gain" in our knowledge about the target variable after observing a feature.

Mathematically, the Information Gain for a feature A is defined as:

$$IG(A) = H(D) - H(D|A) \tag{1}$$

Where $H(D)$ is the entropy of the dataset D before the split. $H(D|A)$ is the expected entropy of the dataset D after it has been partitioned based on the feature A. The average Information gain across the two datasets for each feature is shown in Figure 1. We then use a Random Forest classifier with default hyperparameters to get the F1 score against different Information Gain (IG) thresholds as shown in Figure 2. We aimed to identify the optimal IG threshold that maximizes the classification performance. From Figure 2, we select 0.09 as the threshold as it gives the best F1 score of 88 and 65 across the two datasets. The features selected using the threshold are SrcWin, sHops, sTtl, dTtl, SrcBytes, DstBytes, Dur, TotBytes and Rate. After feature extraction, we employ the Standard Scaler, which transforms the features to have a mean of 0 and a standard deviation of 1. This scaling

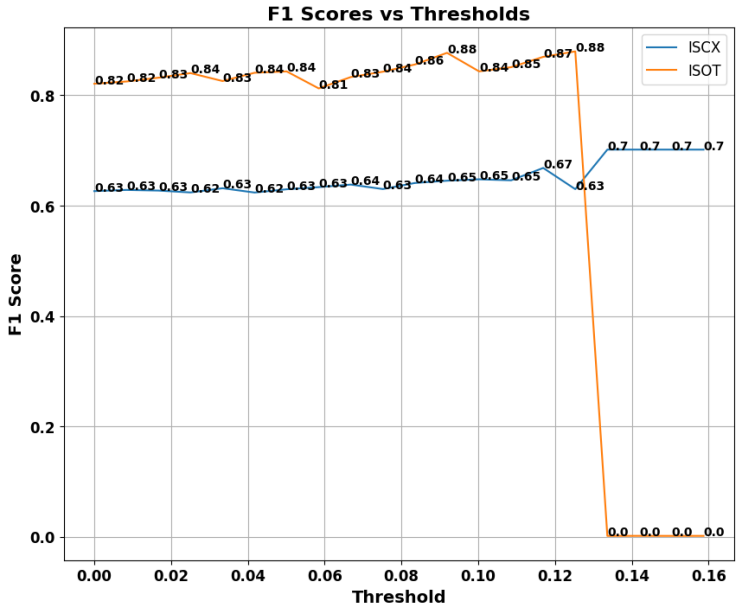

Figure 2: Comparative visualization of average F1 scores across various information gain thresholds for both ISCX and ISOT datasets

process enhances the stability and speed of convergence during the model training phase, ensuring that no particular feature dominates due to its scale.

3.3. Botnet Classification

To detect botnet network flow and compare the results, we experimented with three classifiers: Decision Tree (DT), Random Forest (RF), and Neural Network (NN). Based on the feature selection described in 3.2, we used the following features to train the classifier: 'SrcWin', 'sHops', 'sTtl', 'dTtl', 'SrcBytes', 'DstBytes', 'Dur', 'TotBytes', and 'Rate'. We initially employed default hyperparameter configurations for DT and RF. We constructed a sequential model for the NN classifier consisting of six layers with nine units each, using the ReLU activation function. Following these were two layers with six units and ReLU activations, culminating in a final layer with a single unit employing the sigmoid activation function. The Adam optimizer and the binary cross-entropy loss function were used to train the neural network. We conducted training for fifty epochs with a batch size of 120. Table 2 shows the initial results obtained using default hyperparameters. While these results provided valuable insights, we found that hyperparameter tuning was essential to enhance the model's predictive performance. Therefore, we employed three optimization algorithms, Random Search, Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), to further optimize the hyperparameters of the DT, RF, and NN classifiers.

3.3.1. Hyper Parameter Optimization

To optimize hyperparameters, we start with Random search, which operates by randomly selecting combinations of hyperparameters; we explore random search to establish a baseline

Table 2: **Classifiers performance with default hyper-parameters**

	ISCX				ISOT			
Classifier	**Acc**	**P**	**R**	**F1**	**Acc**	**P**	**R**	**F1**
DT	73.7	74.59	69.29	71.84	99.24	98.21	91.7	94.89
RF	86.57	92.25	78.86	85.03	99.24	98.27	91.75	94.9
NN	64.87	60.52	78.66	68.4	98.75	97.71	91.84	94.68

Acc-Accuracy, P-Precision, R-Recall.

performance level. We then explored nature-inspired algorithms, namely GA(Genetic Algorithm) and PSO (Particular Swarm Optimization). Nature-inspired algorithms are chosen for their robust exploration capabilities of the hyperparameter space through adaptive refinement. GA adaptively focuses the search on regions of the hyperparameter space that have shown promise while still exploring new regions. Over generations, the population converges toward an optimal or near-optimal set of hyperparameters. PSO refines the search space by allowing particles to gravitate towards the best solutions found by themselves and by others in the swarm. This movement is influenced by individual and collective experiences, leading to convergence on optimal solutions as particles iteratively adjust their positions. The mechanism of exploration (searching new areas) and exploitation (improving known good areas) dynamically is crucial for effectively navigating the irregular and high-dimensional search spaces of hyperparameters, which makes them an ideal candidate for hyperparameter search.

- **Random Search(RS)**: Random Search optimization is a more efficient version of exhaustive search. In random search, a selection of parameter values is given and then iterates through them, picking randomly between the parameter values. After the optimization process, the best-performing collection of parameters is selected. Since stochastic values are chosen for each hyperparameter type from a set for a given classifier, this method is inefficient and may not result in an optimal hyperparameter set in a reasonable amount of computational time for guaranteed improvements [11].

- **Genetic Algorithm (GA)** is a search heuristic inspired by Charles Darwin's theory of natural evolution. Natural biological evolution tends to favour the fitter individuals in a population. They survive longer and breed, wherein the genes in the offspring are crossed over from the parents, allowing mutation in the genetic makeup. This natural selection process allows the desirable fittest characteristics to be propagated through the future population generations. We initialized a population of potential solutions, each representing a unique set of hyperparameters for the classifier. For example, in the Decision Tree, the hyperparameters such as criterion, splitter, max depth, and min samples split, among others, were chosen randomly from predefined ranges or sets. To evaluate the performance of each individual, we trained the machine learning classifier using the hyperparameters it represented and then assessed its performance on the validation set. We determined each individual's fitness using two key metrics: the F1 score and accuracy, with higher values indicating superior performance. To evolve our

population, we employed a tournament selection mechanism, selecting individuals for crossover based on their fitness. We paired individuals to produce offspring using a uniform crossover method during the crossover phase. This method ensured that each hyperparameter from the parent individuals had an equal probability of being passed on to the offspring. We applied mutations after the crossover to introduce variability and maintain diversity in our population. In this mutation step, we randomly altered an individual's specific hyperparameters. Over multiple generations, we repeated the selection, crossover, and mutation processes, allowing the population to evolve and improve its overall fitness. Upon the GA's completion, we used this best individual to train our classifier on the entire training dataset. We then evaluated the classifier's performance on the test data.

More formally, we initialized a population of size P, where each individual I_i, for $i = 1, 2, ..., P$, is a vector of hyperparameters represented as $I_i = [h_1, h_2, ..., h_n]$, with n being the total number of hyperparameters. To evaluate the performance of each individual, we employed a fitness function ($f(I_i)$), which returns a tuple comprising the F1 score and accuracy, formulated as $f(I_i) = (F1(I_i), \text{Accuracy}(I_i))$. For the selection process, we used a tournament-based mechanism. A subset S of size T is randomly chosen from the population, and the individual I^* with the highest fitness in (S) is selected, denoted as $I^* = \text{argmax}_{I_i \in S} f(I_i)$. During the crossover phase, two-parent individuals, I_a and (I_b), produce an offspring I_c. Each hyperparameter (j) of the offspring is determined by $I_{c,j} = I_{a,j}$ with a probability of 0.5, or $I_{c,j} = I_{b,j}$ with a probability of 0.5. Mutation introduces variability. For each hyperparameter h_j in an individual I_i, the hyperparameter might be replaced with a random value from its domain with a mutation probability p_m, or it remains unchanged. This iterative selection process, crossover, and mutation continue for G generations, where G is the predefined number of generations.

- **Particle Swarm Optimization (PSO)** is a nature-inspired algorithm based on behaviour observed in swarms of fish, birds, etc. Each particle in the swarm represents a potential solution, encoded as a vector of hyperparameters, $P_i = [h_1, h_2, ..., h_n]$, where n is the total number of hyperparameters. The position of each particle is updated iteratively based on its own best-known position and the global best-known position of the swarm. The equation governs the update rule for each particle's position: $P_{i,t+1} = P_{i,t} + v_{i,t+1}$ where $v_{i,t+1}$ is the velocity of the particle at iteration $t + 1$ and is computed as:

$$v_{i,t+1} = w \cdot v_{i,t} + c_1 \cdot r_1 \cdot (p_{best} - P_{i,t}) + c_2 \cdot r_2 \cdot (g_{best} - P_{i,t}) \tag{2}$$

Here, w is the inertia weight, c_1 and c_2 are cognitive and social coefficients, respectively, r_1 and r_2 are random numbers between 0 and 1, p_{best} is the best-known position of the particle, and g_{best} is the best-known position among all particles in the swarm.

The fitness of each particle is evaluated using a combination of the F1 score and accuracy on the validation set. The PSO algorithm iteratively updates the positions of the

particles to search for the optimal hyperparameters that maximize the fitness function. After a predefined number of iterations, the particle with the highest fitness is selected as the solution, representing our machine classifier's optimal set of hyperparameters. The classifier is then trained on the entire training set using these hyperparameters and evaluated on the test set to obtain the final performance metrics.

The hyperparameters of Decision Tree, Random Forrest and Neural Network classifier were optimized using Random Search, Genetic Algorithm and Particle Swarm Optimization for comparison. The Decision Tree hyperparameter included the criterion, splitter, max depth, min samples per split, min samples per leaf, weight fraction leaf, max features, max-leaf nodes, impurity decrease, and the complexity parameter. We considered both "gini" and "entropy" for the criterion. We evaluated the splitter's "best" and "random" configurations. The max depth of the tree was varied from none to a maximum of 50, incrementing in steps of 3. We examined min samples per split from 2 up to 20 and min samples per leaf from 1 to 20. The weight fraction leaf was assessed from 0 up to 0.5, advancing in increments of 0.05. As for the max features, we explored "auto", "sqrt", "log2", and the possibility of it being none. The max-leaf nodes were checked in values ranging from none to 100, with a gap of 10 units between each. The impurity decrease hyperparameter was probed similarly to the weight fraction leaf, between 0 and 0.5 in 0.05 increments. Lastly, the complexity parameter, CCP alpha, was varied from 0 to 0.05, with 0.01 increments.

For Random Forrest, the hyperparameters include Number of estimators, Criterion, Maximum depth of the tree, Minimum number of samples required to split an internal node, Minimum number of samples required to be at a leaf node, Minimum weighted fraction of the sum total of weights required to be at a leaf node, number of features to consider when looking for the best split, Maximum number of leaf nodes, Threshold for early stopping in tree growth, Whether to use bootstrap samples and Complexity parameter. We set the number of estimators, or trees in the forest, by experimenting with values starting from 1 and going up to 191. For the criterion, the choices were "gini" and "entropy". The depth of individual trees, or max depth, was adjusted from 0 to a ceiling of 50, with intervals of 3. Samples required to split an internal node, or min samples split, were tested from 2 through 20. The minimal number of samples needed to be at a leaf node or min sample leaf, was explored from 1 to 20. The minimum weighted fraction of the total weights required to be at a leaf node was adjusted starting from 0 and reaching up to 0.5 in steps of 0.05. Regarding the number of features to consider when looking for the best split or max features, we incorporated options like "sqrt", "log2", none, and even specific counts from 2 through 9. The maximum number of leaf nodes, or max-leaf nodes, was observed starting from no constraint and going up to 100, increasing in tens. The Threshold for early stopping in tree growth, or min impurity decrease, was set to range between 0 and 1, moving in increments of 0.1. The forest's bootstrap samples, determining whether to use out-of-bag samples to estimate the generalization score or not, were toggled between True and False. Lastly, the complexity parameter, CCP alpha, was allowed to vary from 0 up to 0.05, marking every 0.01 increment. For the neural network, we considered several hyperparameters. We selected the activation functions from 'relu', 'sigmoid', and 'tanh'. We chose the optimizers from 'adam', 'sgd', and

'rmsprop'. We picked the loss functions from 'binary_crossentropy' and 'hinge'. We set the maximum number of hidden layers to 5 and allowed up to 50 nodes for each layer.

In the GA algorithm, the crossover mechanism was implemented using a uniform method, ensuring an even blend of genes from both parent entities. Regarding mutation, we randomly selected a hyperparameter and then altered its value. Selection was achieved through a tournament strategy, picking the finest out of five randomly chosen individuals based on their respective fitness scores. We employed a multi-objective approach to evaluate the fitness of each individual in the population. Specifically, we used the F1 score and accuracy of the classifier on the validation set as the fitness functions. The dual fitness measures ensured we did not just optimize for a single metric but considered the precision-recall balance (F1 score) and overall correct predictions (accuracy). We initiated the algorithm with a population of 100 individuals and ran it over 100 generations. After running the Genetic Algorithm, we identified the hyperparameters with the highest combined fitness score, and the best hyperparameter found is shown in Table 4. We then trained the Decision Tree classifier with these optimal hyperparameters on the entire training dataset and evaluated its performance on the test set. We reported key metrics such as accuracy, precision, recall, and F1 Score, which are shown in Table 7.

Table 3: **Optimal hyperparameter configurations for various classifiers on ISCX and ISOT datasets, through Random Search optimization**

Dataset	**Classifier**	**Hyperparameters**
ISCX	DT	criterion: entropy, splitter: best, max_depth: 24, min_samples_split: 16, min_samples_leaf: 20, min_weight_fraction_leaf: 0, max_features: log2, max_leaf_nodes: 100, min_impurity_decrease: 0.0, ccp_alpha: 0.05
	RF	n_estimators: 1, criterion: entropy, max_depth: 50, min_samples_split: 15, min_samples_leaf: 9, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, bootstrap: False
	NN	num_hidden: 4, nodes: 13, activation: tanh, optimizer: rmsprop, loss: binary_crossentropy
ISOT	DT	criterion: gini, splitter: best, max_depth: 45, min_samples_split: 15, min_samples_leaf: 12, min_weight_fraction_leaf: 0.05, max_features: auto, max_leaf_nodes: 40, min_impurity_decrease: 0.0, ccp_alpha: 0.02
	RF	n_estimators: 1, criterion: entropy, max_depth: 50, min_samples_split: 15, min_samples_leaf: 9, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, bootstrap: False
	NN	num_hidden: 5, nodes: 13, activation: tanh, optimizer: adam, loss: binary_crossentropy

The Particle Swarm Optimisation (PSO) used involved representing particles as vectors of indices that corresponded to classifier hyperparameters. The PSO dynamics were directed by cognitive ('c1'), social ('c2'), and inertia ('w') coefficients, with values set at 1.5, 2, and 0.9, respectively. These coefficients helped to direct the particles towards individual and global best positions. Each particle's position corresponds to a specific combination of hyperparameters. To determine the fitness of each particle, we used a multi-objective approach, which involved evaluating the F1 score and accuracy of the classifier on the test

Table 4: **Optimal hyperparameter configurations for various classifiers on ISCX and ISOT datasets, through GA optimization**

Dataset	Classifier	Hyperparameters
ISCX	DT	criterion: gini, splitter: random, max_depth: 18, min_samples_split: 6, min_samples_leaf: 6, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: 90, min_impurity_decrease: 0.0, ccp_alpha: 0.0
	RF	n_estimators: 2, criterion: gini, max_depth: 27, min_samples_split: 2, min_samples_leaf: 13, min_weight_fraction_leaf: 0.15, max_features: log2, max_leaf_nodes: 60, min_impurity_decrease: 0.0, bootstrap: True, ccp_alpha: 0.0
	NN	num_hidden: 5, nodes: 5, activation: tanh, optimizer: rmsprop, loss: hinge
ISOT	DT	criterion: entropy, splitter: best, max_depth: 15, min_samples_split: 2, min_samples_leaf: 1, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: None, min_impurity_decrease: 0.0, ccp_alpha: 0.0
	RF	n_estimators: 71, criterion: entropy, max_depth: 42, min_samples_split: 3, min_samples_leaf: 2, min_weight_fraction_leaf: 0, max_features: 7, max_leaf_nodes: None, min_impurity_decrease: 0.0, bootstrap: True, ccp_alpha: 0.0
	NN	num_hidden: 3, nodes: 29, activation: tanh, optimizer: rmsprop, loss: binary_crossentropy

Table 5: **Optimal hyperparameter configurations for various classifiers on ISCX and ISOT datasets, through PSO optimization**

Dataset	Classifier	Hyperparameters
ISCX	DT	criterion: entropy, splitter: random, max_depth: 39, min_samples_split: 2, min_samples_leaf: 20, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: 20, min_impurity_decrease: 0.0, ccp_alpha: 0.02
	RF	criterion: entropy, splitter: random, max_depth: 39, min_samples_split: 2, min_samples_leaf: 20, min_weight_fraction_leaf: 0, max_features: None, max_leaf_nodes: 20, min_impurity_decrease: 0.0, ccp_alpha: 0.02
	NN	activation: tanh, optimizer: sgd, loss: binary_crossentropy, hidden_layers: 1, layer_nodes: 30, input_nodes: 9
ISOT	DT	criterion: entropy, splitter: best, max_depth: 30, min_samples_split: 16, min_samples_leaf: 11, min_weight_fraction_leaf: 0, max_features: log2, max_leaf_nodes: 80, min_impurity_decrease: 0.0, ccp_alpha: 0.01
	RF	n_estimators: 61, criterion: entropy, max_depth: 12, min_samples_split: 6, min_samples_leaf: 15, min_weight_fraction_leaf: 0, max_features: 4, max_leaf_nodes: 20, min_impurity_decrease: 0.0, bootstrap: True, ccp_alpha: 0.0
	NN	activation: relu, optimizer: adam, loss: binary_crossentropy, hidden_layers: 3, layer_nodes: 40, input_nodes: 9

set. Then, we transformed these fitness values into a single objective by computing the weighted sum of the F1 score and accuracy. Each objective contributed equally with a weightage of 50%, ensuring that we optimised for a balanced trade-off between precision-recall (captured by the F1 score) and overall correct predictions (accuracy). We initialised the swarm with 100 particles and optimised over 50 iterations. Finally, we identified the

Table 6: **Classifiers performance with Random Search hyper-parameters**

	ISCX				ISOT			
Classifier	**Acc**	**P**	**R**	**F1**	**Acc**	**P**	**R**	**F1**
DT	61.17	85.69	43.82	57.99	95.68	77.39	61.70	68.66
RF	71.35	80.87	69.64	**74.83**	99.23	98.12	91.76	**94.84**
NN	76.39	90.73	68.39	77.99	98.48	94.35	85.34	89.62

Legend: Acc-Accuracy, P-Precision, R-Recall, F1-F1 score.

Table 7: **Classifiers performance with GA optimized hyper-parameters**

	ISCX				ISOT			
Classifier	**Acc**	**P**	**R**	**F1**	**Acc**	**P**	**R**	**F1**
DT	93.83	93.55	96.57	**95.04**	99.24	98.11	91.92	94.91
RF	84.02	89.57	83.61	86.49	99.27	98.16	92.3	**95.14**
NN	91.49	95.3	90.54	92.86	98.74	97.26	86.12	91.35

Legend: Acc-Accuracy, P-Precision, R-Recall, F1-F1 score

Table 8: **Classifiers performance with PSO optimized hyper-parameters**

	ISCX				ISOT			
Classifier	**Acc**	**P**	**R**	**F1**	**Acc**	**P**	**R**	**F1**
DT	87.77	88.99	91.31	**90.13**	96.73	86.08	68.98	76.59
RF	80.62	80.71	89.77	85	98.74	99.26	84.21	**91.12**
NN	79.09	76.77	94.35	84.66	98.54	94.08	86.54	90.15

Legend: Acc-Accuracy, P-Precision, R-Recall, F1-F1 score

global best particle position. We then used these optimal hyperparameters shown in Table 5 to train the classifier on the entire training dataset. The performance of this classifier was subsequently assessed on the test set, and it is shown in Table 8. Finally, to establish a baseline and to show that the GA and PSO makes improvement on the perfromance we show the result of Random search in 6.

4. Adversarial Attack on Classification Model

In order to evaluate the robustness of our machine learning-based Network Intrusion Detection Systems (NIDSs), we crafted adversarial samples. Adversarial samples are instances of data that are intentionally perturbed in such a way as to deceive the model, leading to incorrect predictions. The best model in Section 3 acts as a classifier with no architectural information about the classifier to the attacker. We adopted the C&W [27] and GAN[53] attack, both renowned adversarial attack techniques, to evaluate the robustness of our model against adversarial perturbations on individual features. This section outlines the threat model, constraints associated with crafting these adversarial samples, the background of the C&W and GAN attack and how we adapted it for our problem.

4.1. Threat Model

In our experimental setup, we operate under a grey-box attack scenario. Here, the attacker has complete knowledge of the dataset and features utilized by the model but lacks access to the model's parameters and architecture. This scenario is plausible, as in practical applications, features and datasets used in ML-based NIDSs are often disclosed through publications or documentation. Our primary objective is to scrutinize the vulnerabilities associated with specific features and investigate the transferability of adversarial samples across different models.

4.2. Problem Definition

Let $N = \{N_1, N_2, ..., N_m\}$ be a dataset consisting of network traffic samples, where each sample N_i is characterized by a set of features $\{F_1, F_2, ..., F_k\}$. The task at hand is to evaluate the robustness of a machine learning-based Network Intrusion Detection System (NIDS) model $M : N \rightarrow \{0, 1\}$, which classifies traffic as benign ($M(N_i) = 0$) or malicious ($M(N_i) = 1$).

The focus is specifically on adversarial attacks targeting malicious network traffic samples with the intention of transforming them into adversarial samples that can evade detection by the NIDS model. For a given malicious sample N_i where $M(N_i) = 1$, the objective is to generate an adversarial counterpart N_i' that not only deceives the NIDS model into making an incorrect prediction, $M(N_i') = 0$, but also maintains plausible and coherent feature relationships to resemble legitimate network traffic, adhering to a set of constraints. To accomplish this, we utilize the Carlini & Wagner (C&W)[27] method and GAN[53] method, adapting it to ensure that the perturbations applied to the malicious samples are subtle yet effective in evading detection while preserving the essential characteristics of legitimate network traffic.

4.3. C&W Attack on the Classification model

The primary goal of the C&W attack is to find a perturbation that minimizes the distance between the original and perturbed input, while ensuring that the perturbed input is misclassified by the model. The objective function used in the C&W attack is defined as:

$$Objective = \|x' - x\|_2^2 + c \times f(x') \tag{3}$$

x' is the adversarial example, x is the original input, and c is a constant that balances the trade-off between the perturbation magnitude and the misclassification. The function $f(x')$ is designed to capture the misclassification condition and is defined as:

$$f(x') = \max\left(\max_{i \neq t}(Z(x')_i) - Z(x')_t, -k\right) \tag{4}$$

Here, $Z(x')$ represents the logits or scores produced by the model for input x', t is the target class, and i iterates over all classes except the target class. The parameter k plays a pivotal role in determining the confidence with which the adversarial example should be misclassified. The C&W attack makes use of gradient-based optimization to craft the

adversarial sample. Specifically, the gradient of the objective function to the input x' is computed. This gradient indicates how the value of the objective function will change for a small change in x'. Mathematically:

$$\nabla_{x'}\text{Objective} \tag{5}$$

This gradient tells us the direction in which we should adjust (x') to decrease the value of the objective function most efficiently. The input (x') is adjusted iteratively using the computed gradient. An optimizer is often used for this purpose. The following equation gives the update rule:

$$x'_{new} = x'_{old} - \alpha \times \nabla_{x'} Objective \tag{6}$$

Where α is the learning rate, which controls the step size during the optimization process, the gradient computation and the update rule are repeated until the Pre-defined maximum number of iterations is reached. The C&W loss, which is the objective to be minimized during the attack, combines the L2 distance between the original and perturbed input with the misclassification condition. The L2 distance ensures that the adversarial perturbation is as tiny as possible, making it imperceptible to human observers. On the other hand, the term $c \times f(x')$ ensures that the perturbed input x' is misclassified into the desired target class with a confidence determined by k. The constant c is crucial as it determines the trade-off between these two objectives. The value of k is essentially a margin or threshold that defines the confidence with which we want the adversarial example to be misclassified. If k were set to a positive value, the adversarial example would need to be misclassified and misclassified with a k margin of confidence. Positive k value could lead to more significant perturbations, making the adversarial example further from the decision boundary and potentially more detectable. In our experiment, we set k to 0, which means $f(x')$ becomes positive (indicating a successful attack) as soon as the score for a wrong class exceeds the score for the correct class, even if just by a tiny margin. In our experiment, we have only benign and malware network flow, and we want malware network flow classified as benign; therefore, equation 3 can be written as shown in 7 and *Algorithm 1* outlines the computation of this objective function.

$$f(x') = max(Z(x')_{benign} - Z(x')_{malware}, 0) \tag{7}$$

Algorithm 1 Carlini-Wagner Adversarial Loss Calculation

Input: $x, x', \text{model}, \text{target\_class}, c, \kappa$
Output: loss
Description: This function computes the Carlini-Wagner adversarial loss, which is a combination of the L2 distance between the original and perturbed inputs, and a confidence margin-based loss.
function $f(x', \text{model}, \text{target\_class}, \kappa)$
    $Z \leftarrow \text{model}(x')$ ▷ Get model logits for perturbed input
    if $\text{target\_class} = 0$ **then**
        $Z_{\text{target}} \leftarrow 1 - Z[:, 0]$
        $Z_{\text{other}} \leftarrow Z[:, 0]$
    else
        $Z_{\text{target}} \leftarrow Z[:, 0]$
        $Z_{\text{other}} \leftarrow 1 - Z_{\text{target}}$
    end if
    return $\max(Z_{\text{other}} - Z_{\text{target}}, -\kappa)$ ▷ Compute confidence margin
end function
function CW_LOSS$(x, x', \text{model}, \text{target_class}, c, \kappa)$
    $\text{l2\_dist} \leftarrow \text{reduce\_sum}((x - x')^2)$ ▷ Calculate L2 distance
    return $\text{l2\_dist} + c \cdot f(x', \text{model}, \text{target\_class}, \kappa)$ ▷ Combine losses with balance parameter c
end function
$\text{loss} \leftarrow \text{cw\_loss}(x, x', \text{model}, \text{target\_class}, c, \kappa)$ ▷ Compute CW loss
return loss

In our experimental scenario, the classifier correctly identifies a malware network flow as malicious. As a result, the value of $f(x')$ is zero, and its gradient for the objective function is negligible or absent. This condition presents a significant challenge in generating meaningful adversarial perturbations using gradient-based optimization techniques. When $f(x')$ is zero, the optimization process becomes predominantly focused on minimizing the L_2 distance between the perturbed input x' and the original input x. However, this approach might not yield an effective adversarial example, as it does not guarantee that the perturbed input will be misclassified. To address the challenge of generating adversarial samples when the gradient of the objective function is negligible or zero, we introduce noise to the input, explicitly targeting the features we wish to attack. We generate random values between -1 and 1 and apply this noise element-wise to the attacked features. The noise is scaled relative to the original values of the attacked features, controlled by a specified magnitude. This scaling ensures that the perturbations are proportional to the original feature values, maintaining a balance that avoids excessive distortion. Algorithm 2 outlines the process of generating and applying this relative noise.

Algorithm 2 Generate Relative Noise

Input: x, feature_mask, magnitude
Output: relative_noise
Description: This function generates noise relative to the input x, scaled by the given magnitude, and masked by the feature_mask, ensuring that only specific features are perturbed.
noise ← random_values_in_range(-1, 1, shape_of(x)) ▷ Generate random values
relative_noise ← noise × x × magnitude ▷ Scale noise relative to x
relative_noise ← elementwise_multiply(relative_noise, feature_mask) ▷ Apply feature mask
return relative_noise

The introduction of noise serves to nudge the input x' out of regions where the function $f(x')$ is zero, facilitating a change in the model's classification. As a result, $f(x')$ becomes positive, providing a gradient that can guide the optimization process effectively. With this gradient information, the optimization can minimize the L_2 distance between the original and perturbed input while ensuring that the input is misclassified. This approach is crucial for generating adversarial examples that are both effective (in terms of causing misclassification) and subtle (in terms of minimal perturbations). By targeting specific features for noise addition and scaling the noise relative to the original feature values, we ensure that the perturbations remain constrained and do not lead to overtly distorted adversarial examples.

4.4. Generating Adversarial Network (GAN) attack:

The general architecture of GAN consists of a generator and a discriminator. In the context of fooling the NIDS, it also includes a substitute detector. The generator's (G) function creates synthetic data samples that mimic real network traffic. It starts from a random noise vector z and transforms it into a data sample x'. Mathematically, the generator is represented by the following equation:

$$G(z; \theta_g) = x' \tag{8}$$

Here, θ_g is the parameters of the generator. In our experiment, the generator is a neural network.

The Discriminator (D) role is to differentiate between real (x) and synthetic (x') data samples. It outputs a probability score indicating the likelihood of a sample being real. The Discriminator is represented by the following equation:

$$D(x; \theta_d) = p \tag{9}$$

Here, θ_d are the parameters of the Discriminator, and p is the probability score. In our experiment, the Discriminator is a neural network.

The Substitute Detector (Classifier, C) is a surrogate model for the actual NIDS model, classifying traffic as benign or malicious. It evaluates the synthetic samples x' and is represented as:

$$C(x'; \theta_c) = y' \tag{10}$$

Here, θ_c are the classifier's parameters, and y' is the predicted class. The generator aims to produce samples that the Discriminator should classify as real, and the substitute detector

should misclassify as malware. The loss function for the generator can be formulated as a combination of the Discriminator and the substitute detector's feedback. To train the generator, we use WGAN loss and a misclassification loss given by the substitute detector. It is formulated as

$$L_G = -\text{mean}(D(G(z))) + \text{mean}(C(G(z))) \tag{11}$$

The first term, $-\text{mean}(D(G(z)))$, is the standard WGAN generator loss, encouraging the generator to create samples that the Discriminator evaluates as real. The second term, $\text{mean}(C(G(z)))$, is the substitute loss, encouraging the generator to create samples that are classified as benign by the substitute detector. Here, minimizing the substitute loss means that the generator effectively tries to produce samples that the substitute detector scores as benign (closer to 0). The discriminator loss is designed to maximize the difference in evaluating real and synthetic samples. The loss is given by the following equation:

$$L_D = \text{mean}(D(G(z))) - \text{mean}(D(x)) \tag{12}$$

The first Term $\text{mean}(D(G(z)))$ represents the average score assigned by the discriminatory to the generated (fake) samples. A higher score indicates that the Discriminator considers the fake sample to be more like real data. The second Term $\text{mean}(D(x))$ term represents the average score assigned by the Discriminator to the real data samples. x denotes the actual data. A higher score indicates that the Discriminator correctly recognizes the sample as real. - Minimizing $\text{mean}(D(G(z)))$ pushes the Discriminator to assign lower scores to fake samples. Maximizing $\text{mean}(D(x))$ pushes the Discriminator to assign higher scores to real samples.

4.5. Targeting Individual Feature

Our attack focuses on attacking individual features without altering the entire data instance. A feature mask is employed to achieve these selective perturbations. A feature mask is a binary vector where each entry corresponds to a feature in the data. An entry with a value of 1 indicates that the corresponding feature can be modified, while 0 ensures that the feature remains untouched. This mask guides the adversarial crafting process, ensuring that only the desired features are susceptible to change. For instance, if the goal is to perturb the "Dur" (Duration) feature, the feature mask would have a one at the position corresponding to the "Dur" feature and 0 for all other positions. When the loss gradient to the input data is computed, this feature mask is applied, effectively zeroing out the gradients for all features except "Dur", which ensures that only the "Dur" feature is updated during the optimization process while all other features remain constant. The application of the feature mask is straightforward. After computing the gradients, they are element-wise multiplied with the feature mask. This operation ensures that gradients for untargeted features are nullified, preventing any changes during the optimization step.

4.6. Constraint on Generating Adversarial sample

After the optimization, it is essential to perturb a specific feature and ensure that related features are adjusted accordingly to maintain the coherence and plausibility of the traffic

data. For instance, when the feature "Dur" (Duration) is perturbed, it is vital to adjust the "Rate" feature accordingly. The "Rate" is intrinsically linked to the "Duration" as it represents the number of bytes transferred per unit of time. If the "Duration" of a network session increases or decreases, the "Rate" at which data is transferred would naturally be affected. Specifically, if the "Duration" is shortened, the "Rate" would increase, indicating a faster data transfer, and vice versa. Similarly, when the "SrcBytes" (Source Bytes) feature is manipulated, it directly impacts the "TotBytes" (Total Bytes) feature. "TotBytes" is the sum of "SrcBytes" and "DstBytes" (Destination Bytes). Therefore, any change in "SrcBytes" necessitates a corresponding adjustment in "TotBytes". Additionally, to keep the "Rate" constant, the "Duration" also need to be adjusted based on the new "SrcBytes" value. The features "sHops" and "dHops" represent the number of hop packets taken from the source to the destination. When these are altered, the Time-to-Live (TTL) values, represented by "sTtl" and "dTtl", must be adjusted. The TTL is typically initialized to a value (e.g., 255) and decremented at each hop. Thus, if "sHops" increases, "sTtl" would decrease, indicating that the packet has traversed more routers or switches. The relationship between the feature to be attacked and the necessary adjustment is shown in the following Table 9.

Table 9: Inter-feature relationships and adjustments post-perturbation

Feature Perturbed	**Related Adjustments**
Dur	Rate $= \frac{TotBytes}{Dur+\epsilon}$
SrcBytes	$TotBytes = SrcBytes + DstBytes; Dur = \frac{TotBytes}{Rate}$
DstBytes	$TotBytes = SrcBytes + DstBytes; Dur = \frac{TotBytes}{Rate}$
TotBytes	$SrcBytes = TotBytes - DstBytes;$ $DstBytes = TotBytes - SrcBytes;$ $Dur = \frac{TotBytes}{Rate+\epsilon}$
sHops	$sTtl =$ INITIAL TTL $- sHops$
sTtl or dTtl	$sHops = INITIALTTL - sTtl$
Rate	$Dur = \frac{TotBytes}{Rate+\epsilon}$
SrcWin	No adjustment needed since it is not dependent on other features

4.7. Defense Against Adversarial Sample

To protect against the Adversarial sample generated, we use Adversarial retraining. Adversarial retraining is a defence strategy adopted to enhance the robustness of machine learning models against adversarial attacks. At its core, adversarial retraining involves incorporating adversarial samples into the training dataset and retraining the model. By doing this, the model learns from genuine data and adversarial perturbations. The hope is that this dual exposure during training will equip the model to better recognize and thwart adversarial attempts during actual deployment.

5. Conformal Prediction in Network Flow Detection

Reliable and confident prediction is an essential aspect of network intrusion detection systems (NIDS), where the stakes are high and the margin for error is minimal. In this

context, the conformal prediction framework, as introduced by Vovk et al. [28], offers a robust and principled approach to not only classify network traffic but also quantify the certainty of each prediction. This section delves into the conformal prediction method, which has been adapted for application in network flow detection. For a comprehensive explanation, refer to Vovk et al.'s original paper [28].

When analyzing a dataset of network flows, we divide it into training, calibration, and test sets. Using the training set, we train a classifier, represented as $\hat{f}$. For each input instance x (symbolizing the features of a network flow), the classifier $\hat{f}$ yields two probabilities: $\hat{f}_0(x)$ and $\hat{f}_1(x)$. The former indicates the likelihood that the instance x represents normal traffic, while the latter signifies the probability of it being malicious. These probabilities, confined within the [0,1] interval, always sum to 1. We employ the calibration set to derive conformal scores $s_i = 1 - \hat{f}\left(X_i\right)_{Y_i}$, where s_i stands for one minus the predicted probabilities of the original class, with Y_i representing the true label of the instance (0 for normal and 1 for malicious). When $\hat{f}$ correctly predicts the class, s_i remains low. In contrast, incorrect predictions result in a high s_i value. Subsequently, we determine a threshold $\hat{q}$ from the s_i values in the calibration set. This threshold represents the $\lceil (n+1)(1-\alpha) \rceil / n$ empirical quantile of the scores from s_1 through s_n. For fresh instances in the test set, we formulate a prediction set $C(X_{test}) = \left\{ y : \hat{f}(X_{test})_y \geq 1 - \hat{q} \right\}$. This prediction set satisfies the following condition, given adequate trials [28]:

$$1 - \alpha \leq P(Y_{test} \in C(X_{test})) \tag{13}$$

Here, X_{test} and Y_{test} denote a test data point from a similar distribution, while the user specifies α as the desired error rate within [0,1]. The likelihood of the prediction set containing the correct label is at least $1 - \alpha$—a measure known as *coverage*. For instance, with α set at 0.5, the prediction set has a minimum 95% Upon evaluating a network flow instance from the dataset, the function C outputs a set based on the classifier's probability estimates. In the binary classification context of network traffic as normal or malicious, C can yield one of the following outputs: (a) $\{\emptyset\}$ (b) $\{Normal\}$ (c) $\{Malicious\}$ or (d) $\{Normal, Malicious\}$. For instance, considering $\hat{q}$ as 0.08, and given an input x_i that produces probabilities of 0.95 for malicious and 0.05 for normal traffic, the function C will return $\{Malicious\}$. If the probabilities were 0.4 for malicious and 0.6 for normal, with $\hat{q}$ still at 0.08, then C would yield $\{\emptyset\}$. In binary classifications, the eventuality of C producing an output of $\{Normal, Malicious\}$ is improbable due to the sum of probabilities being 1 and a $\hat{q}$ value of 0.5. A $\hat{q}$ of 0.5 indicates a poorly trained classifier, limiting the possible outputs of C to $\{\emptyset\}$, $\{Normal\}$, or $\{Malicious\}$. If a network flow instance receives an output of $\{\emptyset\}$, we should reject the classifier's classification, potentially marking it for further review. Conversely, outputs of $\{Normal\}$ or $\{Malicious\}$ indicate accepted classifications, backed by the coverage assurance from equation 13. For an in-depth tutorial, please refer [29].

6. Evaluation

6.1. Experimental Setup and Results for Crafting Adversarial Sample

To ensure a comprehensive evaluation of the robustness of our models against adversarial attacks, we meticulously crafted our experimental procedure. Our approach utilizes a surrogate model, a neural network we have pre-trained and discussed in the preliminaries section. This surrogate model serves as a substitute for the target model during the adversarial crafting process, providing gradient information that guides the generation of adversarial samples. Initially, we identify instances correctly predicted as malware by our surrogate model for each dataset ISCX and ISOT. By filtering the dataset based on these criteria, we concentrated our adversarial efforts on the samples most confidently identified by our model, thereby offering a test for our model's resilience. The attack was performed using the C&W and GAN adversarial attack method described in Section 4. We employed Algorithm 3 to iteratively generate adversarial examples using C&W attack and Algorithm 4 for GAN attack. This technique was iteratively applied to each of our selected features, namely 'SrcWin', 'sHops', 'sTtl', 'dTtl', 'SrcBytes', 'DstBytes', 'Dur', 'TotBytes', and 'Rate'. This algorithm ensures that only specific features are perturbed, maintaining the semantic integrity of the data. The process involves initializing with relative noise in C&W and generation of sample from random noise in GAN, applying a feature mask, and then optimizing the objective function to craft the adversarial examples. The primary objective was to subtly manipulate these samples so our model would misclassify them benign using the objective function shown in equation 3 and 11.

Algorithm 3 Generate CW Adversary

Input: model, x, target_class, feature_mask, c, epsilon, iterations, clip_min, clip_max
Output: x_prime
Description: This function generates an adversarial example by iteratively applying perturbations to the input x. The perturbations are guided by the CW loss and are masked to affect only certain features.
noise ← GenerateRelativeNoise(x, feature_mask) ▷ Initialize with relative noise
x_prime ← x + noise ▷ Apply initial perturbation
learning_rate ← epsilon ▷ Set learning rate
for iteration ← 1 to iterations **do**
    loss ← CalculateCwLoss(x, x_prime, model, target_class, c) ▷ Calculate CW loss
    gradient ← compute_gradient(loss, x_prime) ▷ Compute gradient of loss w.r.t x_prime
    masked_gradient ← elementwise_multiply(gradient, feature_mask) ▷ Apply feature mask to gradient
    x_prime ← x_prime - learning_rate × masked_gradient ▷ Update x_prime
    x_prime[feature_mask] ← clip_values(x_prime[feature_mask], clip_min, clip_max) ▷ Clip values to stay within bounds
end for
return x_prime

Algorithm 4 Training Generative Adversarial Network for NIDS Evasion

Input: real_data, batch_size, latent_dim, Discriminator_iterations, generator_iterations, feature_mask, scaler, feature_min, feature_max
Output: trained_generator, trained_Discriminator
Initialize: generator, Discriminator
for each epoch **do**
    for each batch in real_data **do**
        $X_{real} \leftarrow$ Sample real data
        $z \leftarrow$ Sample random noise of shape (batch_size, latent_dim)
        $X_{fake\_raw} \leftarrow$ generator(z) ▷ Generate fake data
        $X_{fake} \leftarrow$ Apply feature mask to X_{fake_raw}, clip to [feature_min, feature_max]
        for c_iter from 1 to Discriminator_iterations **do**
            $fake\_pred \leftarrow$ Discriminator(X_{fake})
            $real\_pred \leftarrow$ Discriminator(X_{real})
            $c\_loss \leftarrow$ Discriminator_loss($real\_pred$, $fake\_pred$)
            Update Discriminator weights to minimize c_loss
        end for
        for g_iter from 1 to generator_iterations **do**
            $z \leftarrow$ Sample random noise of shape (batch_size, latent_dim)
            $X_{fake\_raw} \leftarrow$ generator(z) ▷ Generate new fake data
            $X_{fake} \leftarrow$ Apply feature mask to X_{fake_raw}, adjust with scaler, clip to [feature_min, feature_max]
            $fake\_pred \leftarrow$ Discriminator(X_{fake})
            $substitute\_pred \leftarrow$ substitute_detector(X_{fake})
            $g\_loss \leftarrow$ generator_loss($fake\_pred$, $substitute\_pred$)
            Update generator weights to minimize g_loss
        end for
    end for
end for
return generator, Discriminator

For C&W attack, given the size of malware samples identified by our surrogate model initially, generating adversarial samples for the entire malware samples at once could be computationally intensive. Hence, we adopted a batching approach for the generation. We defined batches with a BATCH_SIZE of 10,000, dividing our dataset into manageable chunks. To commence the attack, we selected a feature and added some noise. To generate the noise, we generate random values within the range of -1 to 1 for each element in the batch. The rationale behind this randomness is to provide an unpredictable but constrained starting point for adversarial perturbations. Instead of relying solely on the random values, the noise was made 'relative' by multiplying it with the original data values, the relative value 0.1(10%) and the noise to ensure that features with larger magnitudes have proportionally larger noise values, making the noise relative to the magnitude of the feature it perturbs and also that the adversarial examples start close in the feature space to the genuine data points. The final step in the noise generation was the application of a feature mask. This mask, essentially a binary vector, was employed to selectively activate the perturbation on specific features of interest while keeping other features untouched. By multiplying the relative noise with this feature mask, we effectively zeroed out the noise for features we intended to

preserve. This ensured that the adversarial crafting only impacted the features we wanted to target. We then optimize iteratively the objective in equation 3 to generate adversarial examples using Adam optimizer with a learning rate set to 0.0001 and the constant c of the objective function, which determines the trade-off between the perturbation's magnitude and the classification error set to 0.01. In each iteration, we computed the gradient of this objective function concerning the perturbed input. However, this gradient is masked so that updates only occur in the desired feature direction, ensuring the semantic and syntactic relationships between features are maintained. Post gradient computation, we updated our adversarial example in the direction that would increase the classification error. It is worth noting that after every iteration, we imposed constraints on the feature modified in our adversarial example to the min and max of the feature in the dataset to ensure its values did not breach the min and max of that particular feature in the dataset. This was crucial for retaining the realism and legitimacy of the crafted examples. The process mentioned above was iteratively repeated until either the adversarial example was crafted satisfactorily or a set threshold of iterations was reached, which is set to 2000 in our experiment.

Algorithm 5 CW Batch

Input: model, scaler, input_samples, target_class,
feature_name, feature_min, feature_max,
it_value, c
Output: perturbed_samples
Description: This algorithm generates a batch of adversarial examples using the Carlini-Wagner method. It adjusts features according to a given feature mask, and ensures that the dependencies between features are maintained.
target_labels ← zeros_like((input_samples)) ▷ Initialize target labels as zeros
original ← inverse_transform(scaler, input_samples) ▷ Transform inputs back to original scale(since the input is in standard sclae)
feature_index ← index(feature_name in feature_list) ▷ Get index of the feature to be perturbed
feature_mask ← [1 if i == feature_index else 0 for i in range(9)] ▷ Create mask for selected feature
perturbed_samples ← GenerateCwAdversary(model, input_samples,
target_class, feature_mask, c, 0.001, it_value,
feature_min, feature_max) ▷ Generate adversarial examples
perturbed_samples_original ← inverse_transform(scaler, perturbed_samples) ▷ Transform perturbed samples back to original scale
perturbed_samples_original[:, feature_index] ← clip_values(
perturbed_samples_original[:, feature_index], feature_min, feature_max) ▷ Clip the feature values to be within valid range of the train dataset
AdjustDependencies(perturbed_samples_original, feature_name, feature_index) ▷ Adjust dependencies between features
perturbed_samples ← transform(scaler, perturbed_samples_original) ▷ Scale perturbed samples back
return perturbed_samples ▷ Return the generated adversarial examples

As mentioned before, generating these adversarial examples was not just a random tweaking of feature values. Instead, we maintained the semantic and syntactic relationship between the features. We began by isolating the specific feature we intended to perturb. Once identified, we then applied the C&W method. After generating the adversarial samples, the subsequent step ensured that these perturbations maintained the inter-feature relationships.

For instance, if the 'Dur' feature was modified, adjustments were made to the 'Rate' feature to preserve their natural relationship. Likewise, changes to the 'SrcBytes' would cascade to both the 'TotBytes' and 'Dur' features to uphold the integrity of the data. Similarly, any perturbation to features like 'sHops' would subsequently impact 'sTtl' and vice versa. The relationship is detailed in Table 9. Algorithm 5 outlines the entire process of generating adversarial samples in batches

In GAN attack, the process of adversarial sample uses a generator-discriminator architecture along with a surrogate model. The generator's role was to create synthetic data samples while the discriminator evaluated these samples' authenticity. The substitute detector, a neural network, served to simulate the target model's response to the adversarial inputs. Unlike the C&W, the adversarial generation begins with the creation of random latent vectors drawn from a normal distribution. The dimension of the latent variable is set to 100, from which the generator learns to map to the data space.

The generator is a neural network with two layers. The input layer is a dense layer with 128 units. It uses the ReLU (Rectified Linear Unit) activation function. The output layer is a dense layer with ten units, which is the number of features that we selected and uses the tanh activation function. To ensure that the output of the generator matches the scale of the original data, especially after standard normalization processes like StandardScaler, we apply a scaling transformation. This transformation involves multiplying the output by a scaling factor, which is the range of the data divided by 2, and then adding an offset, which is the average of the data's minimum and maximum values. This step is crucial for maintaining the realism of the generated data. The discriminator is another neural network that evaluates the authenticity of both real and synthetic data samples. The discriminator input is a dense layer with 128 units and a ReLU activation function. The final layer of the discriminator is a single-unit dense layer, which outputs a score representing the discriminator's assessment of how real or fake the input data appears. In standard GANs, the discriminator's output logit is often passed through a sigmoid activation function during the training phase to calculate the binary cross-entropy loss. However, the raw logit is used directly in WGANs and certain other GAN variants. This approach is based on the principle that using the raw score can lead to more stable training dynamics and better convergence properties, as it avoids potential issues related to vanishing or exploding gradients that can occur with bounded activation functions. The discriminator's loss is shown in equation 12.

The surrogate model, which is a neural network train in our preliminary section, acts as a proxy for the actual target model that the adversarial examples are intended to deceive. In the adversarial crafting process, the generator is tasked with producing samples that can fool both the discriminator and the surrogate model. The generator's training involves optimizing its ability to produce samples that are both realistic (as judged by the discriminator) and deceptive (as judged by the surrogate model). The loss is given in equation 11, which is designed to achieve a balance between producing samples that appear authentic to the discriminator and deceiving the surrogate model. During the optimization process, we employed the Adam optimizer with a learning rate of 0.0001. Our approach to adversarial example generation began with the selection of a target feature from the set 'TotAppByte', 'SAppBytes', 'DAppBytes', 'DstBytes', 'TotBytes', 'SrcBytes'. Similar to C&W, we use a

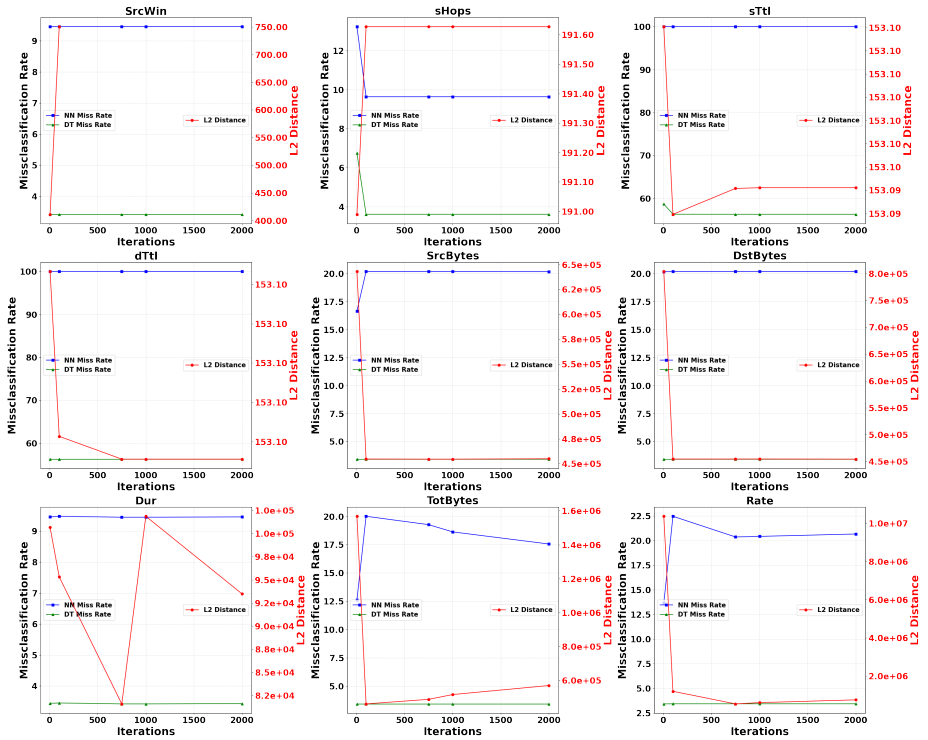

Figure 3: Average L2 distance and misclassification rate in ISCX dataset using C&W attack

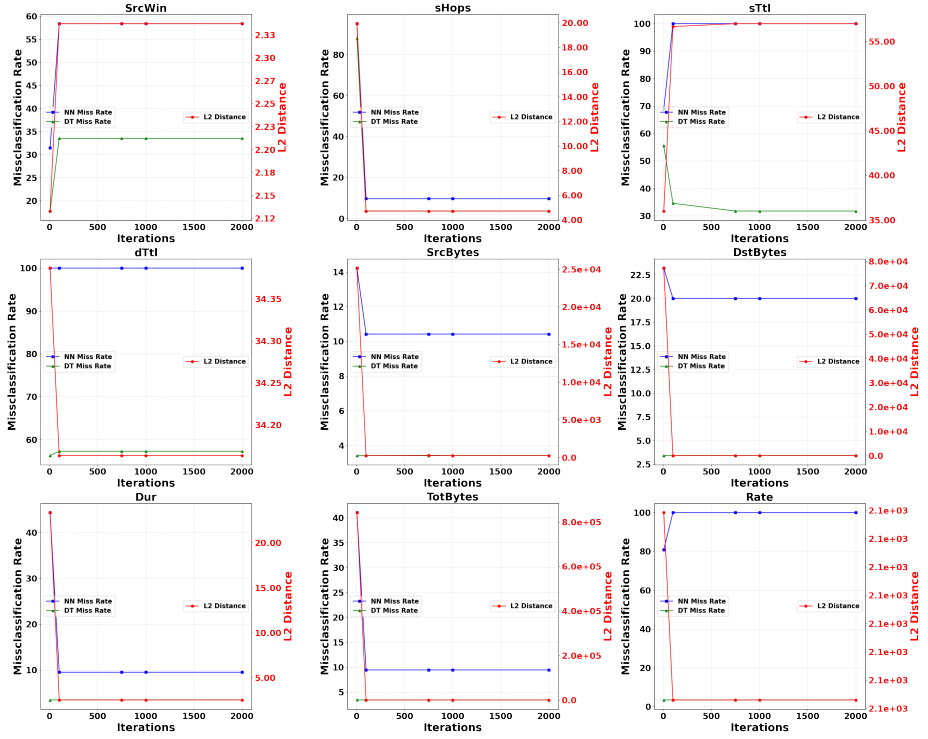

Figure 4: Average L2 distance and misclassification rate in ISCX dataset using GAN attack

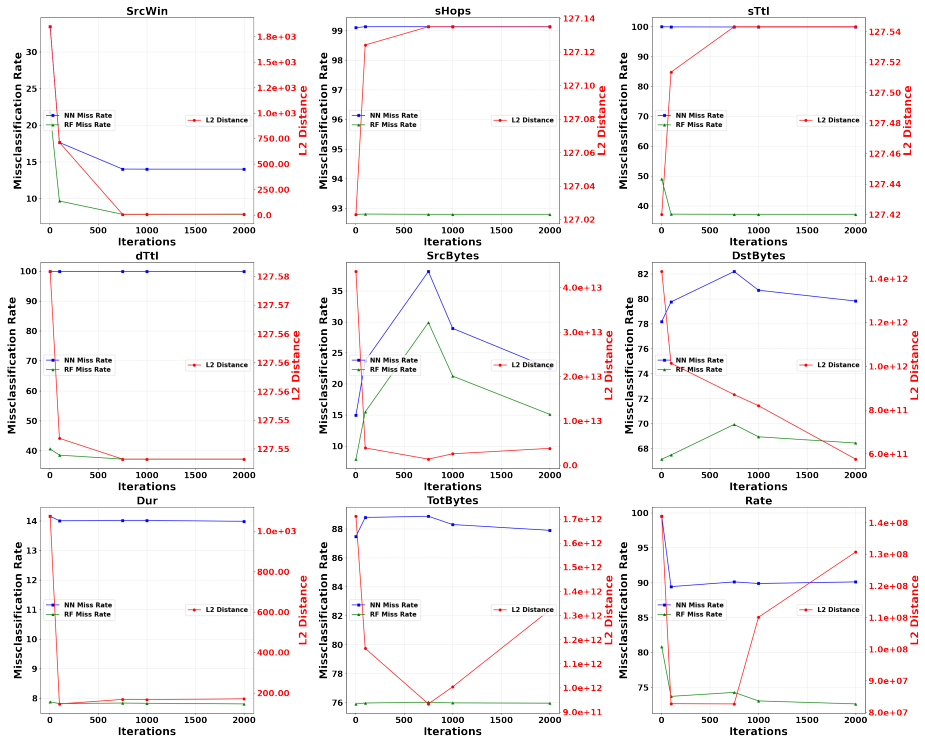

Figure 5: Average L2 distance and misclassification rate in ISOT dataset using C&W attack

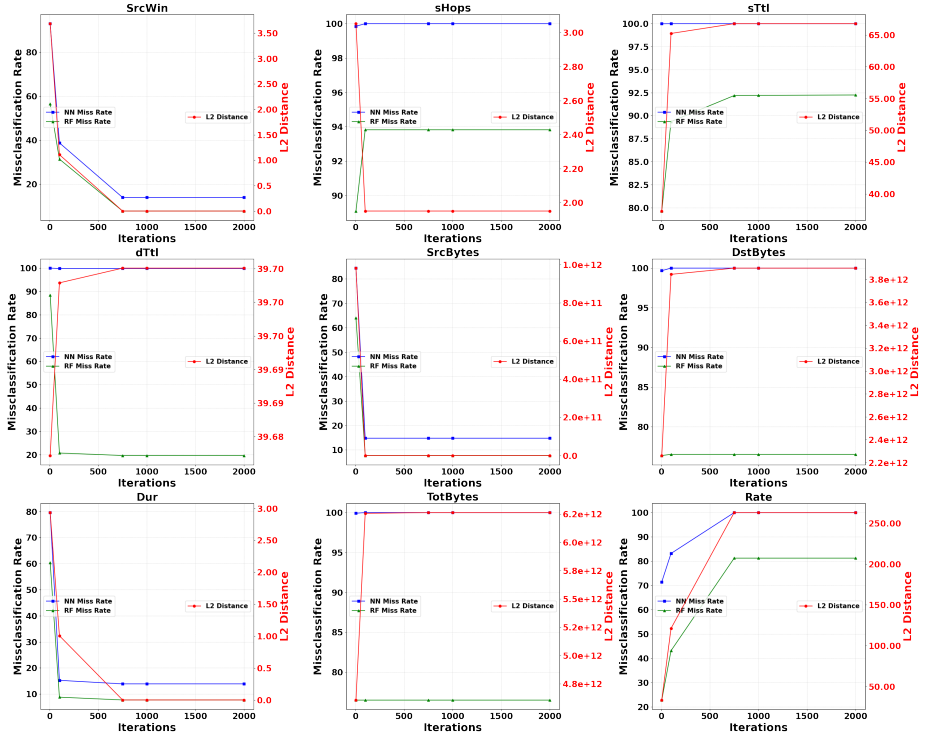

Figure 6: Average L2 distance and misclassification rate in ISOT dataset using GAN attack

binary vector to zero out the gradient of other features and only change the target feature and finally adjust related features using the relationship detailed in Table 9.

To further our understanding regarding the change in the original value compared to the adversarial sample in different points of iteration, we considered multiple iteration checkpoints, which are 5,100,750,1000,2000. For every targeted feature and iteration value, we collected the adversarial samples. Considering the nuances of adversarial attacks, it is not just about how often the model is deceived but also about the magnitude or intensity of the changes made to the original data. For this reason, we determined the average L2 distance difference between the original and adversarial data points, offering a quantitative measure of these adversarial perturbations. This relationship, detailing the trade-off between the difference in L2 distance of adversarial modifications with the original sample and their effectiveness, is captured in Figures 3, 4,5 and 6 for the ISCX and ISOT datasets, respectively.

While we primarily used a surrogate model to generate adversarial samples, we also tested these samples on the 'best' models to verify the transferability of the adversarial attacks. A direct comparison of the misclassification rates between the surrogate models and the most effective models for each dataset (decision tree for ISCX and Random Forest for ISOT) was needed to verify transferability. Figures 3,4, 5 and 6, show the misclassification rate of the surrogate model on the adversarial sample generated and the misclassification of the best model, providing insights on how transferable the attacks are.

6.2. Experimental setup for Adversarial Retraining and Conformal Prediction

In our study, after applying the C&W and GAN adversarial attack on the ISCX and ISOT datasets, we collected a significant number of adversarial samples from 5,100,750,1000,2000 iterations, amounting to a total of 874,036 samples while using C&W and 1,072,827 while using GAN in ISCX dataset whereas for ISOT dataset 60,5915 while using C&W samples and 714,006 while using GAN. This structured approach allowed us to incorporate varying levels of adversarial samples. Given the vulnerability observed in the Decision Tree and Random Forrest Model – which otherwise was the best performer for the ISCX and ISOT datasets- we retrain both models. We began by integrating these adversarial samples into the original training set. However, rather than using the same model parameters, we took an extra step to re-optimize the hyperparameters. For this, we employed a genetic algorithm (GA). Using GA ensures that our model is not just learning from the adversarial samples but also being tuned in the best possible manner to accommodate this new, augmented dataset and the best hyperparameter found is shown in Table 10. Upon retraining with the enhanced dataset and the optimized hyperparameters, we again tested the model against the test sample and the result is shown in 10. We also make sure the adversarial samples are correctly identified, and the result is shown in Table 11.

To evaluate the performance and robustness of conformal prediction in network flow, we allocated 10% of our training data to serve as the calibration set for the conformal prediction process. Utilizing the calibration set, we calculated the conformal scores s_i. With these scores, we were able to determine the threshold $\hat{q}$ which is the empirical quantile of the scores from s_1 through s_n given by $\lceil (n+1)(1-\alpha) \rceil / n$. To identify the optimal threshold α for

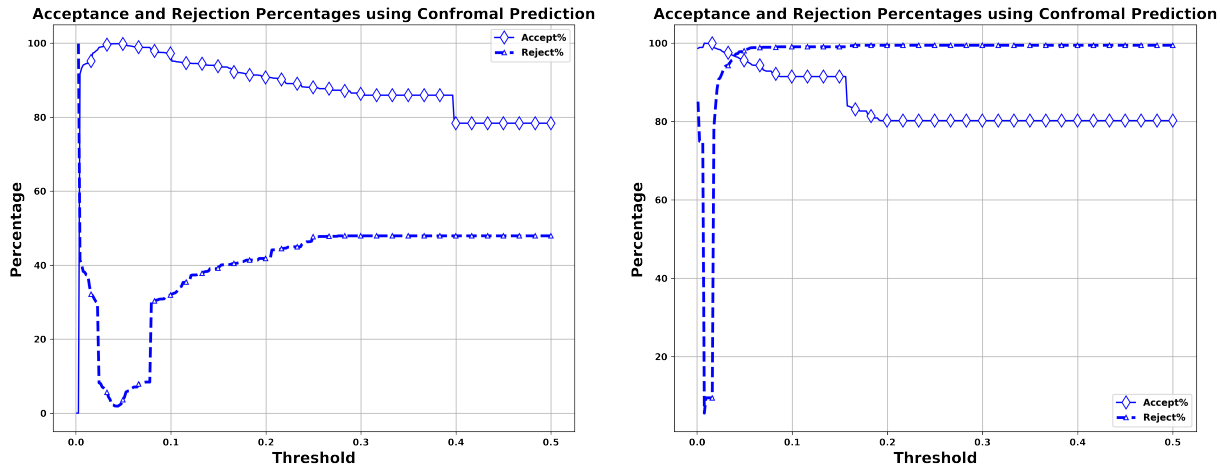

Figure 7: Correctly Accepted and Correctly Rejected Trade-off Graph using a range of α threshold for conformal prediction on ISCX(left) and ISOT(right)

Table 10: Best hyperparameters using GA optimization on ISCX and ISOT dataset with adversarial sample included in training

Dataset	**Classifier**	**Details**	**Type**
ISCX	DT	criterion: gini, splitter: best, max_depth: 18, min_samples_split: 11, min_samples_leaf: 3, min_weight_fraction_leaf: 0, max_features: auto, max_leaf_nodes: None, min_impurity_decrease: 0.0, ccp_alpha: 0.0	Hyperparameters
	DT	Accuracy: 90.93%, Precision: 89.84%, Recall: 96.03%, F1 score: 92.83%	Performance Metrics
ISOT	RF	n_estimators: 200, criterion: entropy, max_depth: 42, min_samples_split: 2, min_samples_leaf: 1, min_weight_fraction_leaf: 0, max_features: 7, max_leaf_nodes: None, min_impurity_decrease: 0.0, bootstrap: True, ccp_alpha: 0.0	Hyperparameters
	RF	Accuracy: 99.23%, Precision: 97.97%, Recall: 91.91%, F1 score: 94.85%	Performance Metrics

Table 11: Adversarial Samples Classifications Before and After Retraining

	Before Retraining	
Dataset	**Benign**	**Malware**
ISCX	GAN:1,072,827, C&W:874,036, Total:1,946,863	0
ISOT	GAN:714,006, C&W:605,916, Total:1,319,922	0
	After Retraining	
Dataset	**Benign**	**Malware**
ISCX	121	1,946,742
ISOT	7	1,319,915

Table 12: Conformal Prediction Performance for ISCX and ISOT

Type	Instances	Accept	Accept(%)	Reject(%)
ISCX - DT+Conformal				
Correctly Predicted				
Benign	85953	71429	83.10	16.90
Malware	156808	150154	95.76	4.24
Total	242761	221583	91.28	8.72
Incorrectly Predicted				
Benign	17726	11053	62.35	37.65
Malware	6470	3028	46.80	53.20
Total	24196	14081	58.20	41.80
ISOT - RF+Conformal				
Correctly Predicted				
Benign	407482	387035	94.98	5.02
Malware	31184	21324	68.38	31.62
Total	438666	408359	93.09	6.91
Incorrectly Predicted				
Benign	643	21	3.27	96.73
Malware	2742	15	0.55	99.45
Total	3385	36	1.06	98.94

accepting or rejecting predictions, we iterate over a spectrum from conservative (high $\hat{q}$) 0.5 to permissive (low $\hat{q}$) 0.001 by generating 200 equally spaced points between the two value and Figure 7 shows the Correctly accepted and Correctly rejected percentage obtained. To maximize the correctly accepted percentage and correctly rejected percentage, we maximize the harmonic mean of the Correctly Accepted Percentage and Correctly Rejected Percentage given in equation 14. The best thresholds found are 0.076 in the ISOT dataset and 0.17 in the ISCX dataset. In our conformal prediction framework, an instance would either be accepted if the prediction set is non-empty, indicating a reliable classification or rejected if the prediction set is empty, highlighting uncertainty and necessitating further inspection. We carefully recorded and analyzed the instances categorized as accepted or rejected, simultaneously distinguishing between those correctly and incorrectly predicted. The results, presented in Table 12, offer a comprehensive breakdown of these categories: correctly predicted and accepted, correctly predicted and rejected, incorrectly predicted and accepted, and incorrectly predicted and rejected.

$$argmax(\alpha) = 2 \times \frac{CA\%(\theta) \times CR\%(\theta)}{CA\%(\theta) + CR\%(\theta)} \tag{14}$$

In this formulation, $CA\%(\theta)$ represents the percentage of Correctly Accepted predictions, calculated as $\frac{CA(\theta)}{\sum 1\{y_i=\hat{y}_i\}} \times 100$, while $CR\%(\theta)$ signifies the percentage of Correctly Rejected predictions, computed as $\frac{CR(\theta)}{\sum 1\{y_i\neq\hat{y}_i\}} \times 100$.

6.3. Experimental Analysis

6.3.1. Evaluating Feature Sensitivity: Average L2 distance and Misclassification Rates

In order to comprehensively evaluate the subtlety and effectiveness of adversarial perturbations, we focused on two key metrics: the change in average L2 distances between original and adversarial samples and the misclassification rate of the surrogate model. The average L2 distance offers insights into the magnitude of perturbations; lower values signify more imperceptible changes, while higher values may hint at overt alterations. On the other hand, the misclassification rate gauges the success of these perturbations in misleading the model.

For the ISCX dataset,the baseline misclassification rate of the surrogate model was identified as 9.45%. During the C&W attack, Starting with the 'SrcWin' feature, a gradual increase in misclassification rates was observed, moving from the baseline of 9.45% and stabilizing at 60%. This uptrend coincided with an increase in the L2 distance, suggesting a direct correlation between the magnitude of perturbation and the likelihood of misclassification. However, after 250 iterations, the generation of adversarial samples ceased, implying a threshold of robustness specific to the 'SrcWin' feature. The 'Shops' feature initially presented a misclassification rate slightly above the baseline at 14%, with an L2 distance starting at 191.0. As iterations progressed, the L2 distance increased marginally to 191.6, yet the misclassification rate intriguingly converged towards the baseline, settling at 9.5%. This pattern indicates the model's capacity to adapt to and mitigate the impact of increasing perturbations on the 'Shops' feature. Conversely, the 'sTtl' and 'dTtl' features exhibited a consistent vulnerability, with misclassification rates persistently at 100%. The L2 distance for these features showed a slight decline, suggesting that even minimal perturbations were sufficient to compromise the model's accuracy, highlighting a significant susceptibility in handling TTL-related attributes. In examining the 'SrcBytes' and 'DstBytes' features, a notable decrease in L2 distance was observed, from 650,000 to 450,000 and from 800,000 to 450,000, respectively. Despite the reduction in perturbation magnitude, the misclassification rates remained around 20%, showing the efficacy of the attack in deceiving the model even with subtler alterations. The 'Duration' feature demonstrated an initial decrease in L2 distance, indicating an attempt to refine the adversarial perturbations. However, the misclassification rate showed minimal variation, suggesting that the model's performance could be influenced by more pronounced perturbations, as evidenced by the subsequent increase in L2 distance. 'TotBytes' and 'Rate' features further elucidated the model's response spectrum to adversarial challenges. 'TotBytes' experienced a substantial reduction in L2 distance, which correlated with an improvement in classification accuracy, as the misclassification rate declined to 17.5%. Meanwhile, the 'Rate' feature underwent a significant drop in L2 distance but only saw a slight decrease in the misclassification rate, suggesting that certain features may require more nuanced adjustments to enhance the model's resilience. In summary, 'SrcWin' and 'Shops' showcase the model's capacity to withstand increasing perturbations; others, notably TTL-related features, highlight intrinsic vulnerabilities that adversarial attacks can exploit.

During the GAN attack of the ISCX dataset, the 'SrcWin' feature experienced an incremental rise in L2 distance, from an ideal baseline of zero to 2.33, which corresponded with an elevation in the misclassification rate from 30% to a stable 60%. This trend indicates

a saturation point in the classifier's vulnerability, beyond which additional perturbations ceased to increase the misclassification rate, suggesting a nuanced robustness of the classifier against adversarial manipulations targeting the 'SrcWin' feature. Conversely, the 'Shops' feature showcased a different pattern, where the L2 distance started at approximately 20 but decreased gradually to around 3 and with it, the misclassification decreased linearly. Significant findings were observed in the TTL-related features, 'sTtl' and 'dTtl', where the model exhibited a consistent vulnerability. For 'sTtl', the L2 distance escalated from 35 to 58, accompanying a surge in misclassification rate to a consistent 100%. Similarly, 'dTtl' maintained a 100% misclassification rate throughout, with the L2 distance showing a minor ascent from 34.5 to 34.1. These outcomes emphasize the classifier's susceptibility to adversarial attacks on TTL attributes, where even slight perturbations significantly compromise classification accuracy. For 'SrcBytes', a noteworthy reduction in L2 distance from 25,000 to 0 was observed, and with it, the misclassification decreased from 14 to 11. A similar pattern emerged for 'DstBytes', where the L2 distance saw a substantial reduction from 80,000 to 0, yet the misclassification rate persistently hovered around 20%. These instances demonstrate the classifier's consistent challenge in accurately discerning adversarial samples from genuine ones, even as perturbations became less pronounced. The 'Duration' feature revealed an initial decline in L2 distance from 20 to approximately 0, which was met with a decrease in misclassification rate from 40% to 11%. Similar patterns were observed in TotBytes' features. Conversely, the 'Rate' feature didn't change its L2 distance much, staying at around 2100, and the misclassification remains consistently around 100%.

Upon analyzing the ISCX dataset subjected to both C&W and GAN adversarial attacks, it's evident that each method exhibits distinct characteristics in terms of perturbation efficiency and the resulting misclassification rates across various network features. The nuanced examination reveals that the GAN attack generally produces adversarial samples with lower L2 distances compared to those generated by the C&W attack, suggesting a subtler approach to manipulating the data while still effectively deceiving the classifier. For instance, during the GAN attack, features like 'SrcBytes' and 'Duration' saw a notable decrease in L2 distance, approaching nearly zero, which did not significantly compromise the model's ability to classify correctly, as seen in the modest decrease in misclassification rates. This contrasts with the C&W attack, where, despite achieving high misclassification rates for features like 'sTtl' and 'dTtl', the approach necessitated larger perturbations, as indicated by the relatively higher L2 distances. Such differences highlight the GAN attack's capability to exploit the model's vulnerabilities with minimal deviation from the original dataset, making these perturbations less detectable and potentially more dangerous. Moreover, the effectiveness of the GAN attack in maintaining or even lowering L2 distances while achieving high misclassification rates, particularly for features such as 'SrcWin' and 'Shops', shows its effectiveness. The 'SrcWin' feature, under the GAN attack, exhibited an increase in misclassification rates to 60% with a moderate increase in L2 distance to 2.33, showcasing the attack's precision in targeting the model's weaknesses. Conversely, the C&W attack, though effective in increasing misclassification rates, often required more significant alterations to the data, as seen in the increased L2 distances for the same and other features like 'Shops'. Interestingly, TTL-related features ('sTtl' and 'dTtl') consistently showed vulnera-

bility under both attacks, with 100% misclassification rates, showing an area for enhancing the classifier's defences.

In the ISOT dataset, the baseline misclassification rate of the surrogate model was identified as 13.87%. During the C&W attack, starting with the 'SrcWin' feature, we observed an initial average L2 distance of approximately 1.75×10^3, which corresponded with a misclassification rate of 32.5%. Over the iterations, this L2 distance narrowed closer to the original data, culminating in a reduced misclassification rate of about 15% by the 750th iteration. This trajectory suggests that the model becomes more adept at identifying adversarial samples as they become subtler, indicating a growing challenge for the attack to deceive the model effectively. The 'Shops' feature exhibited a contrasting scenario with an astoundingly high initial misclassification rate of 99%, despite only a minimal increase in the average L2 distance from 1.2708×10^2 to 1.2713×10^2. 'sTtl' and 'dTtl' observe similar patterns with Shops where the misclassification rate is very high with little change in the Initial L2 distance. 'SrcBytes' saw an L2 difference starting at 1×10^{13}, halved over the iterations, accompanied by a fluctuating misclassification rate that peaked at 36% during the 750th iteration before dropping to 22%. This indicates a diminishing deception efficiency as adversarial examples more closely resemble genuine data. A similar trend was noted for 'DstBytes', with the L2 difference commencing at 1.4×10^{12} and reducing to 7×10^{11}, alongside a temporary spike in misclassification to 82%, which then slightly reduced to 79.5% by the end of the iterations. The 'Duration' feature, despite a significant reduction in L2 difference from 1.2×10^3 to 2×10^2, maintained a relatively stable misclassification rate around 14.1%, suggesting the model's inherent robustness to perturbations affecting this feature. The 'TotBytes' feature showed a nearly consistent misclassification rate of around 88% with the L2 distance fluctuating around 1.7×10^{112} to 9×10^{11}. Finally, the Rate feature shows a decrease in misclassification rate of 100% to 90% with its L2 distance fluctuating between 1.4×10^{08} to 8×10^{07}. In summary, features like 'Shops', 'sTtl', and 'dTtl' remain significantly vulnerable despite minimal perturbation changes. The consistent misclassification rate for the 'Duration' feature despite varying L2 distances underscores the feature resilience against adversarial perturbations.

During the GAN attack of the ISOT dataset, The 'SrcWin' feature showcased a remarkable improvement in the classifier's ability to discern between genuine and adversarial samples, as evidenced by the reduction in L2 distance from approximately 3.50 to near zero, accompanied by a decrease in misclassification rate from around 87% to 10%. This significant improvement indicates that the neural network became increasingly effective at classifying samples correctly as adversarial perturbations were refined to more closely resemble the original data. In stark contrast, the 'Shops' feature maintained a stubbornly high misclassification rate of 100%, despite a halving of the L2 distance from around 3 to 1.5. This persistence in high misclassification rates, regardless of a reduction in perturbation magnitude, highlights a particular vulnerability of the model to attacks targeting the 'Shops' feature, suggesting an area where the model's defences could be bolstered. The TTL-related features 'sTtl' and 'dTtl' presented a uniform challenge, with both features experiencing a consistent 100% misclassification rate. For 'sTtl', the L2 distance increased from about 40 to 65, and for 'dTtl', it showed a negligible rise from 39.68 to 39.70, underlining the

model's ongoing susceptibility to adversarial manipulations affecting TTL attributes despite varying degrees of perturbation. The 'SrcBytes' feature, with a substantial reduction in L2 distance from $1x10^12$ to nearly 0, saw a corresponding decrease in misclassification from approximately 85% to 15%. Similarly, the 'Duration' feature witnessed a decrease in L2 distance from around 3.0 to nearly 0, with misclassification rates dropping from about 80% to 15%, demonstrating the model's enhanced performance as adversarial examples became increasingly indistinguishable from genuine data. However, the 'DstBytes' and 'TotBytes' features displayed 100% with a very large L2 distance. Lastly, the 'Rate' feature underscored a direct correlation between increased perturbation magnitude and heightened vulnerability, with L2 distance expanding from 45 to 250 and misclassification rates climbing from 70% to 100%. In summary, While certain features like 'SrcWin' and 'Duration' evidenced the model's potential for improvement and resilience, others, notably 'Shops,' 'sit,' 'dTtl,' 'DstBytes,' and 'TotBytes,' underscored persistent vulnerabilities.

In comparing C&W and GAN attacks on the ISOT dataset, certain features like 'Shops', 'sTtl', 'dTtl', and 'DstBytes' displayed a consistent vulnerability under both attack types, with high misclassification rates that were relatively unaffected by the scale of L2 distance changes. This indicates that these features are inherently more susceptible to adversarial manipulations, regardless of the attack method. While the C&W attack demonstrates the potential for inducing high misclassification rates with larger perturbations, the GAN attack distinguishes itself by achieving similar or even higher rates of misclassification with subtler, less detectable changes in L2 distance.

6.3.2. Experimental Analysis of Adversarial Attack Transferability

To understand the transferability of the attack in the ISCX dataset, we tested adversarial examples that are successful against the surrogate model on a Decision Tree. The original Decision Tree had a misclassification rate of 3.42%. For features like 'SrcWin' and 'Duration', the Decision Tree held its ground with misclassification rates near this baseline for the C&W attack; however, during the GAN attack, the misclassification increased to 35%. For 'Shops', the rate increased to 7% before settling back to 3.4% during the C&W attack and during the GAN attack, the misclassification increased to around 85% but eventually dropped down to around 15% as the L2 distance became smaller noticeably, the surrogate model, and the Decision Tree displayed same misclassification rate implying the attacks are transferable. The 'sTtl' and 'dTtl' features significantly rose, with rates hovering at around 50% during both attacks, indicating the tree's vulnerability for these features. In contrast, 'TotBytes' and 'Rate' only saw minor increases in both attacks, with rates around 4%. This highlighted that while the surrogate model (NN) was susceptible to the adversarial attack, the Decision Tree largely resisted the attack on most of the features except for sTtl and dTtl in both attacks. Although SrcWin and SHops are hard to attack using C&W attack, with GAN, the two features saw an increase in the misclassification rate.

For the ISOT dataset, in the 'SrcWin' feature, the attack pushed the RF model's misclassification rate up to 20% for C&W and 60% for GAN. However, this heightened misclassification dipped back to the 7.7% baseline as we iterated. This swing back to the baseline suggests the C&W attack might lose some of its edge as its perturbed samples draw closer

to genuine data. The Shops feature consistently gets a misclassification rate of 93% during the C&W attack and 94% during the GAN attack. In the 'sTtl' feature, GAN achieves a better misclassification rate as compared to the C&W attack, staying constant at 90% for GAN, but for C&W, the misclassification drops down to 40%. For 'dTtl', it converged to around 35% for the C&W attack, but for GAN, the misclassification drops down from 90% to around 21% as the iteration progresses. The 'SrcBytes' feature saw its highest misclassification around the 750th iteration, after which it decreased to 15% during the C&W attack, whereas during GAN, the misclassification decreased from 60% to around 15% as the iteration progressed. During the C&W attack of the 'DstBytes' misclassification rate increased at 70% around the same 750th iteration mark and settled at 68% as the iteration progressed; however, for GAN, the misclassification remained constant at around 75%. Interestingly, the 'Duration' feature remained largely unfazed by the C&W attack, holding its misclassification rate steady at 7.7%; however, during the GAN attack, the misclassification drops down from 60% to the baseline as L2 distance decreases. The 'TotBytes' feature experienced a relatively similar misclassification rate of around 75% during both attacks. Finally, 'Rate' misclassification started at 81% and was reduced to 74% during the C&W attack, whereas the misclassification rate increased from 20% to around 80% for the GAN attack.

Our comprehensive examination across the ISCX and ISOT datasets has shown the intricate nature of adversarial sample transferability and its varying impact on classifier performance. Specific features such as 'sTtl', 'dTtl', and 'Shops' demonstrated a heightened vulnerability, with adversarial samples successfully compromising the model's integrity. Contrastingly, features like 'Duration' showcased a commendable resilience against C&W; however, GAN shows the feature is still vulnerable to its attack. Crucially, it is observed that the misclassification rates induced by adversarial samples are consistently lower when tested on the primary models (Decision Tree for ISCX and Random Forest for ISOT) in comparison to the surrogate Neural Network model. Confirming not all the samples generated by the surrogate model is transferable.

6.3.3. Experimental Analysis of Adversarial Retraining and Conformal Prediction

Upon the completion of the adversarial retraining process, the models displayed remarkable improvement in their capability to correctly identify adversarial samples, as can be seen in Table 11. In the initial state, prior to retraining, the models were unable to correctly classify any of the adversarial samples in both the ISCX and ISOT datasets, misclassifying them all as benign, which changed post-retraining. The Decision Tree model misclassified a mere 121 out of the 1,946,863 adversarial samples in the ISCX dataset, showcasing a significant leap in its ability to detect and correctly classify adversarial inputs. In the case of the ISOT dataset, the Random Forest model demonstrated even more impressive performance, only misclassifying 7 adversarial samples. In addition to this notable improvement in adversarial sample detection, the models also maintained robust performance metrics on the test set, as shown in Table 10.

Conformal prediction has played an indispensable role in enhancing the trustworthiness and reliability of our models, as demonstrated by the comprehensive results laid out in Table 12. By implementing this technique, we have been able to substantially increase our

confidence in the model's predictions, thus ensuring that the outputs are not only accurate but also dependable. In the context of the ISCX dataset, when integrated with the Decision Tree model, conformal prediction exhibited exceptional performance. A staggering 91.28% of the total instances were correctly accepted, signifying that the model, when combined with conformal prediction, is highly adept at recognizing and affirming correct predictions. This is a crucial aspect, especially in network security applications, where the cost of false negatives can be substantial. On the flip side, the model also showcased its proficiency in identifying potential misclassifications, successfully rejecting 58.20% of the incorrectly predicted instances. This indicates that the model is not only good at affirming what it knows but also at flagging what it does not, adding a layer of security and reliability. The effectiveness of conformal prediction was further shown in the ISOT dataset, where it was employed alongside the Random Forest model. Here, it correctly accepted 93.09% of the total instances, showcasing its ability to discern and uphold accurate predictions. More impressively, it rejected an overwhelming 98.94% of the incorrect predictions. This high rejection rate is particularly noteworthy, as it highlights the model's capacity to effectively eliminate unreliable outputs, ensuring that the predictions made are both precise and dependable.

By striking a delicate balance between accepting correct predictions and rejecting incorrect ones, conformal prediction has proved to be an invaluable asset in the classification process. It has not just bolstered the accuracy of our models but has also added a level of reliability and robustness that is paramount in network security contexts. This dual capability ensures that our models are not just performing well but also providing outputs that we can trust, making them indispensable tools in network security.

6.4. Performance Comparison with Other Works

In our research, we have primarily focused on identifying adversarial samples, assessing feature vulnerabilities, and analyzing the transferability of adversarial samples. Despite this, we have also evaluated the performance of classifiers in our method and compared them with other research papers.

In the ISCX dataset comparison shown in Table 13, the Decision Tree (DT) classifier shows competitive results compared to single classifiers from other works. Without the conformal prediction layer and adversarial retraining, our DT classifier achieves an F1 score of 95.04%, which is comparable to the results from Li, Y. & Yao (2022) [30] who used ResNet and CBAM-Resnet, achieving F1 scores of 93.0% and 95.73%, respectively. However, when we integrate the conformal prediction layer, our DT classifier's performance is enhanced significantly, reaching an F1 score of 95.52%. This surpasses the single classifiers' performance and is comparable to the Parallel CBAM-ResNet ensemble and Self-attention from Li, Y.& Yao (2022), which achieved an F1 score of 97.15%. The LSTM from Shah Hosseini et al. (2022) [31] achieved an F1 score of 94.5%, and the ensemble approach of LSTM-RF has a better result. However, it is crucial to highlight that these methods utilize multiple classifiers, and integrating a conformal prediction layer on top of these ensemble methods could yield even better performance. Guangali[38] utilize a multilayer Transformer and DNN, achieving an accuracy of 91.92%, which is comparable to the accuracy obtained by our work

Table 13: Comparison of Related Works on ISCX Dataset

Author	**Classifier**	**Performance**
Li, Y. & Yao 2022 [30]	ResNet	Acc=93.67%, Prec=92.2%, F1=93.0%
Li, Y. & Yao 2022 [30]	CBAM-Resnet	Acc=95.85%, Prec=95.26%, F1=95.73%
Hassan et al 2021[16].	Payload Embedding	Acc=80.9%, Prec=80.9%, Rec=80.9%, F1=80.9%
Li, Y. & Yao 2022 [30]	Parallel CBAM-ResNet and Self-attention	Acc=97.26%, Prec=96.94%, Rec=97.15%
Shahhosseini et al. 2022 [31]	LSTM	Prec=95%, Rec=94%, F1=94.5%
Guangli 2024 [38]	Bot-DM: Multilayer Transformer and DNN	Acc=91.92%, Prec=91.45%, Rec=91.45%
Meher Afroz 2024 [54]	DT	Acc=81.71%, Prec=83.92%
*Our work (without Adversarial Hardening and Conformal Prediction)	DT	Acc=93.83%, Prec=93.55%, Rec=96.57%, F1=95.04%
*Our work (with Adversarial Hardening and Conformal Prediction)	DT	Acc=94.02%, Prec=93.14%, Rec=98.02%, F1=95.52%

Legend: Acc-Accuracy, Prec-Precision, Rec-Recall, F1-F1-score

95.04% which can be further improved if conformal layers are implemented along with the model. Meher Afroz[55] utilize DT, but there is no hyperparameter optimization, which likely impacts their performance.

In the ISOT dataset comparison Table 14, our method with the Random Forest (RF) classifier and conformal prediction layer outperforms all other methods listed, achieving an F1 score of 99.92%. This is a significant improvement over other techniques, such as SoNSTAR by Debashi et al. (2018) [33], which achieved an F1 score of 98.3%, and the DT classifier by Khan et al. (2019) [34], which had an accuracy of 98.7%. Mehdi et al. [55] that using DNNSVMLib achieved a similar result with an accuracy of 99.64%, which is slightly lower than our accuracy of 99.99%, which might be just statistical noise, but they didn't provide other important metrics such as Precision, Recall and F1.

Table 14: Comparison of Related Works on ISOT Dataset

Author	**Classifier**	**Performance**
Mai & Park 2016 [32]	K-means	Detection=97.11%
Pektas & Acarman 2017 [35]	RF	F1=99.0%, Acc=99.5%, Rec=99.0%, Prec=99.0%
Debashi, M et al 2018[33]	SoNSTAR	Acc=99.2%, Prec=97.1%, Rec=99.5%, F1=98.3%,
Khan et al. 2019 [34]	DT	Acc=98.7%
Mehdi Asadi et al 2020[55]	DNNSVMLib-c4.5	Acc=99.64%
*Our work (without Adversarial Hardening and Conformal Prediction)	RF	Acc=99.27%, Prec=98.16%, Rec=92.3%, F1=95.14%
*Our work (with Adversarial Hardening and Conformal Prediction)	RF	Acc=99.99%, Prec=99.90%, Rec=99.93%, F1=99.92%

Legend: Acc-Accuracy, Prec-Precision, Rec-Recall, F1-F1-score

In addressing the challenges posed by adversarial attacks in network intrusion detection, our research introduces a novel analytical perspective by rigorously examining the vulnerability of model features through an in-depth analysis of the L2 distance and iteration counts required for successful misclassification. Unlike existing studies, such as those employing standard GAN and C&W attack [19, 22, 23], our approach provides a feature-wise evaluation of feature sensitivity under adversarial conditions. By comparing the efficacy of GAN and C&W attacks, we demonstrate that GANs are capable of achieving misclassification at significantly lower L2 distances, thus highlighting their potential for more efficient adversarial strategies. Additionally, our work explores the Discriminatory aspect of transferability in adversarial attacks, an area often overlooked in other studies. We investigate the application of adversarial examples across different models, revealing that while some attack samples deceive a surrogate model, they do not always generalize across different feature sets due to varying misclassification rates. This insight is visually supported by Figures 4 and 5, which illustrate the nuanced relationship between perturbed features' L2 distances and their corresponding classification rates of the surrogate model (Neural Network) and the actual model based on a tree-based algorithm across different datasets.

In a significant advancement over traditional methods, our study introduces the application of a conformal prediction layer to network intrusion detection systems (NIDS), focusing on mitigating the impact of uncertain predictions. Unlike other approaches that primarily aim to enhance overall model accuracy, our method leverages the robust framework of conformal prediction to specifically reject uncertain outcomes, thereby improving the re-

liability and trustworthiness of the predictions. This technique, which can be universally applied across any classifier, has demonstrated substantial improvements in model performance within our tests. Specifically, implementing the conformal prediction layer increased the F1 score from 93.75% to 95.32% on the ISCX dataset and from 95.12% to 99.79% on the ISOT dataset.

Our method ensures that the classifiers are high-performing in accuracy, precision, recall, and F1 score and resilient against adversarial manipulations. This dual focus provides a more comprehensive and reliable solution for network security. Our method, therefore, offers a balanced approach that does not sacrifice the ability to identify adversarial samples for high performance on clean data, ensuring robustness in real-world scenarios where adversarial attacks are a constant threat.

7. Limitations

The enhancement of F1 scores through the implementation of a conformal prediction layer has been demonstrated; however, it comes with a cost of rejecting correct predictions, and future studies can explore the reduction of rejection of correct predictions. The efficacy of both the classifiers and the conformal prediction layer is heavily contingent upon the quality and representativeness of the training data. In scenarios where the training data are not sufficiently diverse or fail to encapsulate the full spectrum of potential attack vectors and normal traffic patterns, the model's performance may not accurately reflect its effectiveness in real-world applications. Despite advancements aimed at enhancing robustness against adversarial attacks, the dynamic and evolving nature of these threats may still pose challenges. Newly developed or previously unseen adversarial strategies could potentially compromise the models. Ongoing adaptation and refinement of the models, informed by the latest adversarial tactics, are imperative to sustain high-security levels. Our investigation into the transferability of adversarial examples highlights significant complexities and dependencies on specific model characteristics. Although our findings provide valuable insights, they represent only a preliminary exploration within a complex, multifaceted research area. Further studies are needed to elaborate on these dynamics, potentially broadening the investigation to include diverse models and adversarial conditions.

8. Time Complexity Analysis

This section delves into the computational complexities associated with various components of our study, providing insights for understanding the computational demands of the employed methodologies.

The primary classifiers under consideration are the Decision Tree (DT) and Random Forest (RF), utilized respectively in the ISCX and ISOT datasets. The Decision Tree exhibits a time complexity of $O(N \cdot M_f \cdot \log(N))$, where N denotes the number of training examples, and M_f represents the number of features. On the other hand, the Random Forest, an ensemble of multiple decision trees, has a time complexity of $O(N \cdot M_f \cdot \log(N) \cdot T_{RF})$, with T_{RF} indicating the number of trees within the forest.

In hyperparameter optimization, the Genetic Algorithm (GA) was identified as the optimal choice through comparative analysis with Particle Swarm Optimization (PSO). The GA initializes populations with a complexity of $O(P \cdot n_{hp})$, where P stands for the population size and n_{hp} signifies the number of hyperparameters. Each population's evaluation demands $O(P \cdot E)$ time complexity, with E encapsulating the classifier training and fitness function computation. The tournament selection process in GA adds a complexity of $O(P \cdot T_{GA})$, and the crossover operation necessitates $O(P \cdot n_{hp})$ time. Aggregating these complexities, the overall per generation complexity of the GA algorithm sums up to $O(G \cdot P \cdot (3n_{hp} + E + T_{GA}))$, with G representing the number of generations. When considering the Decision Tree classifier inclusive of hyperparameter optimization, the resultant time complexity is $O(N \cdot M_f \cdot \log(N)) + O(G \cdot P \cdot (3n_{hp} + E_{DT} + T_{GA}))$. Similarly, for the Random Forest classifier, the overall time complexity is expressed as $O(N \cdot M_f \cdot \log(N) \cdot T_{RF}) + O(G \cdot P \cdot (3n_{hp} + E_{RF} + T_{GA}))$.

The C&W attack methodology involves multiple computational steps. The calculation of the L2 norm, with D representing the input space's dimensionality, incurs a time complexity of $O(D)$. Neural network output evaluation, denoted as $Z(x')$, and subsequent operations result in a complexity of $O(M_{NN} + C)$, where M_{NN} encapsulates the neural network evaluation complexity, and C represents the number of classes. The gradient computation through backpropagation, alongside additional operations for the perturbation term, introduces a complexity of $O(D + M_{NN})$. Factoring in the iterative nature of the Adam optimization algorithm, denoted by I iterations, the cumulative time complexity of the C&W attack approximates to $O(I \cdot (D + M_{NN} + C))$.

The GAN attack consists of training two neural network Generator and Discriminator. The time complexity of neural network is $O(t \cdot n \cdot \sum_{i=1}^{n-1} x_i \cdot x_{i+1})$.Let the generator G have n_G layers with $g_1, g_2, \ldots, g_{n_G}$ nodes in each respective layer and the discriminator D has n_D layers with $d_1, d_2, \ldots, d_{n_D}$ nodes in each respective layer. If t is the number of training examples, n is the number of epochs. Assuming the discriminator is updated k times for each update of the generator, the total complexity is $O\left(t \cdot n \cdot \left(k \cdot \sum_{i=1}^{n_D-1} d_i \cdot d_{i+1} + \sum_{i=1}^{n_G-1} g_i \cdot g_{i+1}\right)\right)$.

In conformal prediction, the calibration set, consisting of N_c instances, necessitates probability estimates and conformal scores computation, resulting in a time complexity of $O(N_c \cdot M_f)$. The subsequent threshold determination requires sorting the conformal scores, adding a $O(N_c \cdot \log(N_c))$ complexity. For the test set, comprising N_t instances, the formulation of prediction sets demands a time complexity of $O(N_t \cdot D_t)$ for Decision Trees and $O(N_t \cdot T_{RF} \cdot D_{RF})$ for Random Forests. This results in a total time complexity of $O(N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot D_t)$ for Conformal Prediction with Decision Trees and $O(N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot T_{RF} \cdot D_{RF})$ for Random Forests. The final time complexity of the whole process will be $O(N \cdot M_f \cdot \log(N) + G \cdot P \cdot (n_{hp} + E_{DT} + T_{GA}) + N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot D_t + I \cdot (D + M_{NN} + C))$ for Decision tree and $O(N \cdot M_f \cdot \log(N) \cdot T_{RF} + G \cdot P \cdot (n_{hp} + E_{RF} + T_{GA}) + N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot T_{RF} \cdot D_{RF} + I \cdot (D + M_{NN} + C))$ for Random Forrest.

In the case of the Decision Tree (DT) and Random Forest (RF) classifiers, the convergence of the algorithm where the model's performance stabilizes and does not significantly improve with further training is highly influenced by the tree depth, number of trees, and

the complexity of the data. For the DT classifier, it generally converges upon sufficiently partitioning the feature space. For RF, convergence is achieved when additional trees do not markedly improve the model's performance. These factors directly impact the time complexities of $O(N \cdot M_f \cdot \log(N))$ and $O(N \cdot M_f \cdot \log(N) \cdot T_{RF})$ for DT and RF respectively, where a faster convergence could potentially lead to reduced computation time.

In hyperparameter optimization using the Genetic Algorithm (GA), convergence occur when the values of the objective function (such as accuracy, F1 score, etc., for a classifier) stop showing significant changes over generations, the convergence rate is contingent on the number of generations G, population size P, and tournament size T_{GA}. These parameters influence the overall time complexity of $O(G \cdot P \cdot (n_{hp} + E + T_{GA}))$. The more the number of G and P larger the exploration space of the hyperparameter.

The C&W and GAN attack convergence is Discriminatoral, especially given its iterative nature. In C&W the associated time complexity is $O(I \cdot (D + M_{NN} + C))$ where, I represents the number of iterations required for convergence. Similarly for GAN the associated time complexity is $O\left(t \cdot n \cdot \left(k \cdot \sum_{i=1}^{n_D-1} d_i \cdot d_{i+1} + \sum_{i=1}^{n_G-1} g_i \cdot g_{i+1}\right)\right)$ where n represents the number of iteration. A higher number of iterations might yield a more precise adversarial example but at an increased computational cost, necessitating a careful calibration of I and n to ensure efficiency.

Conformal prediction, though not iterative, still requires careful consideration of the calibration set size N_c and the number of test instances N_t, as these parameters influence the time complexities of $O(N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot D_t)$ and $O(N_c \cdot M_f + N_c \cdot \log(N_c) + N_t \cdot T_{RF} \cdot D_{RF})$ for Decision Trees and Random Forests, respectively.

9. Conclusion

Our research has made significant strides in advancing network security defences, particularly in the realm of botnet detection and adversarial sample mitigation. By leveraging both machine learning and deep learning algorithms and fine-tuning their hyperparameters with Genetic Algorithms and Particle Swarm Optimization, we established a strong foundation and achieved optimal predictive accuracy. Our in-depth analysis of feature vulnerabilities using GAN and C&W attack method revealed crucial insights, allowing us to maintain meaningful semantic and syntactic relationships even when features were manipulated. This meticulous approach to adversarial example generation and our investigation into their transferability across different model architectures shows the breadth and complexity of the threat landscape. The introduction of conformal prediction to Network Intrusion Detection Systems marked a significant innovation in our research. This robust, statistically grounded method enhanced the reliability of our model's predictions by confidently accepting correct predictions and crucially rejecting incorrect ones. The impressive rejection rates of 58.20% for incorrect predictions in the ISCX dataset and 98.94% in the ISOT dataset speak volumes about the efficacy of this approach. In future, we plan to explore additional adversarial attack methods that could offer a broader understanding of potential vulnerabilities in Network Intrusion Detection Systems (NIDS). By exposing our models to a broader array of attack vectors, we can further strengthen their resilience and improve

their detection capabilities. While conformal prediction has shown promise in enhancing the reliability of NIDS, further research could focus on refining this approach.

10. Data availability

The datasets used in this study are essential for validating our proposed methodologies and are publicly available, ensuring transparency and reproducibility of our results. Specifically, we utilized the following datasets: ISOT Botnet Dataset [36] and ISCX 2014 Botnet Dataset [37]. The dataset can be downloaded after requesting the owner of the dataset.

References

[1] Thanh Vu SN, Stege M, El-Habr PI, Bang J, Dragoni N. A . Survey on Botnets: Incentives, Evolution, Detection and Current Trends. Future Internet; (2021) 13(8):198. https://doi.org/10.3390/fi13080198

[2] Aljanabi, M. & Ahmad, H. Navigating the Void: Uncovering Research Gaps in the Detection of Data Poisoning Attacks in Federated Learning-Based Big Data Processing: A Systematic Literature Review. *Mesopotamian Journal Of Big Data*. 2023 pp. 149-158 https://doi.org/10.58496/mjbd/2023/019

[3] Yaseen, M. & Albahri, A. Mapping the Evolution of Intrusion Detection in Big Data: A Bibliometric Analysis. *Mesopotamian Journal Of Big Data*. 2023 pp. 138-148 https://doi.org/10.58496/mjbd/2023/018

[4] Spamhaus. (2019). *Botnet Threat Report 2019*. Spamhaus.com [Online] Geneva: Spamhaus. Available at: https://www.deteque.com/app/uploads/2019/02/Spamhaus-Botnet-Threat-Report-2019.pdf [Accessed 06 December 2019].

[5] Dynstatus. "Update Regarding DDoS Event Against Dyn Managed DNS on October 21, 2016", Dynstatus.com. [Online]. Available: https://www.dynstatus.com/incidents/5r9mppc1kb77. (2020) [Accessed 06 December 2019].

[6] Cloudflare. (2019). Famous DDoS Attacks. [Online] Available at: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/ [Accessed 06 December 2019].

[7] Putman, C., Abhishta and Nieuwenhuis, L. J. M. (2018). "Business Model of a Botnet", *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018. https://doi.org/10.1109/pdp2018.2018.00077

[8] Baker, A. R., Caswell, B., & Poor. M. . Snort 2.1 Intrusion Detection, 2nd ed.: Syngress Publishing. (2004) https://doi.org/10.1016/b978-193183604-3/50015-1

[9] Roesch, M. "Snort - Lightweight Intrusion Detection for Networks," (1999) in LISA '99: https://doi.org/10.1016/b978-159749099-3/50015-x

[10] Shah, S. and Issac, B. . "Performance comparison of intrusion detection systems and application of machine learning to Snort system," *Future Generation Computer Systems*, vol. 80 (2018), pp. 157-170. https://doi.org/10.1016/j.future.2017.10.016

[11] Bergstra, J., & Bengio, Y. . Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, (2012) 13(10), 281–305.

[12] Chen, R., Niu, W., Zhang, X., Zhuo, Z. and Lv, F. "An effective conversation-based botnet detection method," *Mathematical Problems in Engineering*, vol. 2017. https://doi.org/10.1155/2017/4934082

[13] Velasco-Mata, J., González-Castro, V., Fernández, E. & Alegre, . Efficient Detection of Botnet Traffic by Features Selection and Decision Trees. *IEEE Access*. **9** (2021) pp. 120567-120579 https://doi.org/10.1109/access.2021.3108222

[14] Dollah, R. F. M., Faizal, M., Arif, F., Mas'ud, M. Z. and Xin, L. K. . "Machine learning for http botnet detection using classifier algorithms," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, (2018) vol. 10,no. 1-7, pp. 27–30.

[15] Haddadi, F., Phan, D.-T. and Zincir-Heywood, A. N. . "How to choose from different botnet detection systems?," *NOMS 2016-2016 IEEE/IFIP Net-work Operations and Management Symposium*, Istanbul, (2016) pp. 1079-1084. https://doi.org/10.1109/noms.2016.7502964

[16] Hassan, M., Haque, M., Tozal, M., Raghavan, V. & Agrawal, R. Intrusion Detection Using Payload Embeddings.(2021) *IEEE Access*. https://doi.org/10.1109/access.2021.3139835

[17] Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J. and Hakimian, P. "Detecting p2p botnets through network behavior analysis and machine learning," in *2011 Ninth annual international conference on privacy, security and trust*, 2011 pp. 174-180. https://doi.org/10.1109/pst.2011.5971980

[18] CIC. . Canadian Institute for Cybersecurity, Botnet 2014 - Datasets - Research - University of New Brunswick. [Online] Available at: https://www.unb.ca/cic/datasets/botnet.html, 2014 (Accessed 19 March 2020).

[19] Alhajjar, E., Maxwell, P. & Bastian, N. Adversarial machine learning in network intrusion detection systems. *Expert Systems With Applications*. **186** (2021) pp. 115782. https://doi.org/10.1016/j.eswa.2021.115782

[20] Hashemi, M., Cusack, G. & Keller, E. Towards evaluation of nidss in adversarial setting. *Proceedings Of The 3rd ACM CoNEXT Workshop On Big DAta, Machine Learning And Artificial Intelligence For Data Communication Networks*. (2019) pp. 14-21 https://doi.org/10.1145/3359992.3366642

[21] Sadeghzadeh, A., Shiravi, S. & Jalili, R. Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification. *IEEE Transactions On Network And Service Management*. **18** (2021), 1962-1976 https://doi.org/10.1109/tnsm.2021.3052888

[22] Han, D., Wang, Z., Zhong, Y., Chen, W., Yang, J., Lu, S., Shi, X. & Yin, X. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors. *IEEE Journal On Selected Areas In Communications*. **39** (2021) , 2632-2647. https://doi.org/10.1109/jsac.2021.3087242

[23] Chen, J., Gao, X., Deng, R., He, Y., Fang, C. & Cheng, P. Generating adversarial examples against machine learning-based intrusion detector in industrial control systems. *IEEE Transactions On Dependable And Secure Computing*. **19** (2020) , 1810-1825. https://doi.org/10.1109/tdsc.2020.3037500

[24] Barbero, F., Pendlebury, F., Pierazzi, F. & Cavallaro, L. Transcending transcend: Revisiting malware classification in the presence of concept drift. *2022 IEEE Symposium On Security And Privacy (SP)*. (2022) pp. 805-823. https://doi.org/10.1109/sp46214.2022.9833659

[25] Garcia, S., Grill, M., Stiborek, J. and Zunino, A. (2014). "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, https://doi.org/10.1016/j.cose.2014.05.011

[26] Bullard, C. (2020). "ra(1) — argus-client — Debian testing — Debian Manpages", *Manpages.debian.org*. [Online]. Available: https://manpages.debian.org/testing/argus-client/ra.1.en.html. (Accessed: 28-Sep- 2020).

[27] Carlini, N. & Wagner, D. Audio adversarial examples: Targeted attacks on speech-to-text. *2018 IEEE Security And Privacy Workshops (SPW)*. (2018) pp. 1-7. https://doi.org/10.1109/spw.2018.00009

[28] Vovk, V., Gammerman, A. & Shafer, G. Algorithmic learning in a random world. (Springer,2005). https://doi.org/10.1007/978-3-031-06649-8

[29] Angelopoulos, A. & Bates, S. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *ArXiv Preprint*. (2021), ArXiv:2107.07511. https://doi.org/10.1561/9781638281597

[30] Li, Y. & Yao, R. Botnet Detection Method Based on Parallel CBAM-ResNet and Self-attention. *2022 IEEE International Conference On Signal Processing, Communications And Computing (ICSPCC)*. (2022) pp. 1-6. https://doi.org/10.1109/icspcc55723.2022.9984230

[31] Shahhosseini, M., Mashayekhi, H. & Rezvani, M. A deep learning approach for botnet detection using raw network traffic data. *Journal Of Network And Systems Management*. **30** (2022), 44. https://doi.org/10.1007/s10922-022-09655-7

[32] Mai, L. and Park, M. "A comparison of clustering algorithms for botnet detection based on network flow," 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), Vienna, (2016) pp. 667-669, https://doi.org/10.1109/icufn.2016.7537117

[33] Debashi, M. & Vickers, P. Sonification of network traffic for detecting and learning about botnet behavior. *IEEE Access.* **6** (2018) pp. 33826-33839. https://doi.org/10.1109/access.2018.2847349

[34] Khan, R. U., Zhang, X., Kumar, R., Sharif, A., Golilarz, N. A. and Alazab, M. (2019). "An adaptive multi-layer botnet detection technique using machine learning classifiers," *Applied Sciences*, vol. 9, no. 11, Art no. 2375. https://doi.org/10.3390/app9112375

[35] Pektaş, A. & Acarman, T. Effective feature selection for botnet detection based on network flow analysis. *International Conference Automatics And Informatics.* (2017) pp. 1-4.

[36] Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J. & Hakimian, P. Detecting P2P botnets through network behavior analysis and machine learning. *Proceedings Of The 9th Annual Conference On Privacy, Security And Trust (PST2011).* (2011,7) pp. https://doi.org/10.1109/pst.2011.5971980

[37] Beigi, E., Jazi, H., Stakhanova, N. & Ghorbani, A. Towards effective feature selection in machine learning-based botnet detection approaches. *Communications And Network Security (CNS), 2014 IEEE Conference On.* (2014) pp. https://doi.org/10.1109/cns.2014.6997492

[38] Wu, G., Wang, X., Lu, Q. & Zhang, H. Bot-DM: A dual-modal botnet detection method based on the combination of implicit semantic expression and graphical expression. *Expert Systems With Applications.* (2024) pp. 123384 https://doi.org/10.1016/j.eswa.2024.123384

[39] Wang, W., Shang, Y., He, Y., Li, Y. & Liu, J. BotMark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors. *Information Sciences.* **511** (2020) pp. 284-296 https://doi.org/10.1016/j.ins.2019.09.024

[40] Hosseini, S., Nezhad, A. & Seilani, H. Botnet detection using negative selection algorithm, convolution neural network and classification methods. *Evolving Systems.* **13** (2022), 101-115. https://doi.org/10.1007/s12530-020-09362-1

[41] Layeghy, S., Baktashmotlagh, M. & Portmann, M. DI-NIDS: Domain invariant network intrusion detection system. *Knowledge-Based Systems.* 273 (2023) pp. 110626. https://doi.org/10.1016/j.knosys.2023.110626

[42] Hnamte, V. & Hussain, J. DCNNBiLSTM: An efficient hybrid deep learning-based intrusion detection system. *Telematics And Informatics Reports.* **10** (2023) pp. 100053. https://doi.org/10.1016/j.teler.2023.100053

[43] Qazi, E., Faheem, M. & Zia, T. HDLNIDS: hybrid deep-learning-based network intrusion detection system. *Applied Sciences.* 13 (2023) , 4921. https://doi.org/10.3390/app13084921

[44] Kumar, G., Kumar, R., Kumar, K., Sai, N. & Brahmaiah, M. Deep residual convolutional neural Network: An efficient technique for intrusion detection system. *Expert Systems With Applications.* 238 (2024) pp. 121912.

[45] Debicha, I., Bauwens, R., Debatty, T., Dricot, J., Kenaza, T. & Mees, W. TAD: Transfer learning-based multi-adversarial detection of evasion attacks against network intrusion detection systems. *Future Generation Computer Systems.* 138 (2023) pp. 185-197. https://doi.org/10.1016/j.future.2022.08.011

[46] Debicha, I., Cochez, B., Kenaza, T., Debatty, T., Dricot, J. & Mees, W. Adv-Bot: Realistic adversarial botnet attacks against network intrusion detection systems. *Computers & Security.* 129 (2023) pp. 103176. https://doi.org/10.1016/j.cose.2023.103176

[47] Roshan, K., Zafar, A. & Haque, S. Untargeted white-box adversarial attack with heuristic defence methods in real-time deep learning based network intrusion detection system. *Computer Communications.* 218 (2024) pp. 97-113. https://doi.org/10.1016/j.comcom.2023.09.030

[48] Sarıkaya, A., Kılıç, B. & Demirci, M. RAIDS: Robust autoencoder-based intrusion detection system model against adversarial attacks. *Computers & Security.* **135** (2023) pp. 103483. https://doi.org/10.1016/j.cose.2023.103483

[49] Mohammadian, H., Ghorbani, A. & Lashkari, A. A gradient-based approach for adversarial attack on deep learning-based network intrusion detection systems. *Applied Soft Computing.*137 (2023) pp. 110173. https://doi.org/10.1016/j.asoc.2023.110173

[50] Roshan, K., Zafar, A. & Haque, S. A novel deep learning based model to defend network intrusion detection system against adversarial attacks. *2023 10th International Conference On Computing For*

Sustainable Global Development (INDIACom). (2023) pp. 386-391.

[51] Kumar, V., Kumar, K. & Singh, M. Generating practical adversarial examples against learning-based network intrusion detection systems. *Annals Of Telecommunications*. (2024) pp. 1-18. https://doi.org/10.1007/s12243-024-01021-9

[52] Barik, K., Misra, S. & Fernandez-Sanz, L. Adversarial attack detection framework based on optimized weighted conditional stepwise adversarial network. *International Journal Of Information Security*. (2024) pp. 1-24. https://doi.org/10.1007/s10207-024-00844-w

[53] Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein generative adversarial networks. *International Conference On Machine Learning*.(2017) pp. 214-223.

[54] Afroz, M., Ibnath, M., Rahman, A., Sultana, J. & Rab, R. On feature selection algorithms for effective botnet detection. *Journal Of Network And Systems Management*. 32 (2024) , 43. https://doi.org/10.1007/s10922-024-09817-9

[55] Asadi, M., Jamali, M., Parsa, S. & Majidnezhad, V. Detecting botnet by using particle swarm optimization algorithm based on voting system. *Future Generation Computer Systems*. 107 (2020) pp. 95-111. https://doi.org/10.1016/j.future.2020.01.055