MSc in Computer Science 2018-19

Project Dissertation

Project Dissertation title: PROMISE: Provably RObust Malware detectIon uSing diffErential privacy

Term and year of submission: Trinity Term 2019

Candidate Number: 1029012

Approximate Word Count: 26,611 (Main body text- math inline text)

Abstract

Network traffic analysis attempts to infer sensitive information from packet and flow communication patterns. One of the most important applications of this analysis is malware/malicious flow detection. Using network traffic analysis, experts have shown that botnets, Trojans, and other types of malware can be detected and then subsequently expunged from a network. Furthermore, as network traffic in greater quantities is encrypted and becomes more complex, machine learning and deep learning approaches increasingly are being used to detect malware. This inevitably comes with a price. Deep learning algorithms are critically vulnerable to adversarial examples. Using this vulnerability, carefully designed malware, by perturbing ordinarily inconsequentially aspects of their design, can effectively evade detection from these system. In this work, we firstly showcase the effectiveness of these types of these attacks, showing that a simple black-box attack can cause a 40% decrease in accuracy on the USTC-TFC2016 dataset and that on in a white-box setting, a similar attack can cause a 75% decrease in accuracy on the CSE-CIC-IDS2018 dataset.

In order to stop adversarial attacks a number approaches including adversarial training, defensive distillation, and feature squeezing. However, here we show that using Lecuyer et al.'s [47] approach of using differential privacy properties to provide provable robustness, we can provably provide *guarantees* on machine learning algorithms ability to separate malicious from benign flows. We show specifically that at a minimal loss of accuracy ($< 5\%$), for small attack vector sizes, we effectively and provably prevent adversarial examples from tricking our models. We further propose in this work a means of using the robustness metric that we glean from this approach as means setting priorities when investigating flows and thus of making malware detection systems more manageable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An attack in 2013 on the Target Corporation using malicious software or *malware* managed to compromise millions of customers' credit card information. Specifically, the malware installed on the company's registers managed to copy and send the information of whomever shopped at Target throughout the 2013 holiday season [19]. Following the attack, in addition to the leak of millions of individuals' private data, Target's market shares fell 11%.

Malware and *network attacks* can have devastating effects on a given system. Recent attacks like the Target malware attack can have devastating monetary effects and can undermine the privacy of millions of people. Other attacks like the 2016 botnet attack Mirai prevented millions of people from accessing Amazon.com, the BBC, Netflix, and the New York Times [12]. The GoldenEye ransomware caused blackouts in Ukraine when it targeted Ukrengergo, the state power distributor [8]. Being able to detect malware and their associated network flows, while challenging, is of vital importance.

Traditional approaches to malware and network attack detection focused on signature-based approaches. Signature-based approaches search for the specific binary hashes of malicious software. These types of approaches however, have begun to fall out of favour due the increased complexity of attacks. Signature-based approaches increasingly fail because they are only able to detect attacks that have previously been seen before. As a result of the increasing complexity of malware and malicious network activity, network operators and analyst have been looking towards a new option: machine learning.

Machine learning, particularly deep learning, attempts to learn deep underlying features in previous raw traffic data in order to detect new and different types of malicious network flows. The advantage of deep learning is that complex features can be learned automatically (at the expense of interpretability), from raw data and that it scales well with the amount of data that it must process as well [76]. Large strides were initially made within this area; an outstanding amount of information could be learned from malware packets and network flows.

This progress however was largely complicated with the increasing encryption of network traffic on the internet. All content-based network traffic analyses are rendered useless when the traffic payloads are all encrypted. Works like Wang et al.'s [72], which detects malware in HTTP flows by looking at payloads as a text document and then performing natural language processing on the flows, become defunct in an encrypted setting. Taking encryption into account is largely no longer an option. At the time of writing (August 2019), 91% of sites visited by US users were secured using encryption and 85% of sites visited by international users were encrypted. Similarly, over 75% of email traffic has become encrypted. This trend is thus here to stay, foiling many different types of machine learning malware detection techniques [5].

As a result of these changes in network traffic, many researchers have recently begun looking to using other features in machine learning algorithms to detect malicious traffic flows. These types of features have included packet inter-arrival times, packets sizes, number of bytes sent, Transport Layer Security (TLS) handshake information, among a host of other information. The benefit of using these features is that statistical features cannot be hidden as they are intrinsic to networking. Furthermore, other aspects of this information like the TLS handshake must be sent in the clear (unencrypted) in order to set up connections between a client and server. Although inherently less accurate than previous approaches, these methodologies have led to impressive results. Some networks have been able to achieve upwards of 94% [59], 98% [70], 99% accuracy [74] on their respective datasets.

However, despite this recent success in detecting malicious flows amongst benign ones, current methodologies come at a clear cost. Neural networks and associated machine learning techniques are vulnerable to adversarial examples. Adversarial examples are created when small perturbations cause a neural network to reclassify a given instance x to another completely different label.

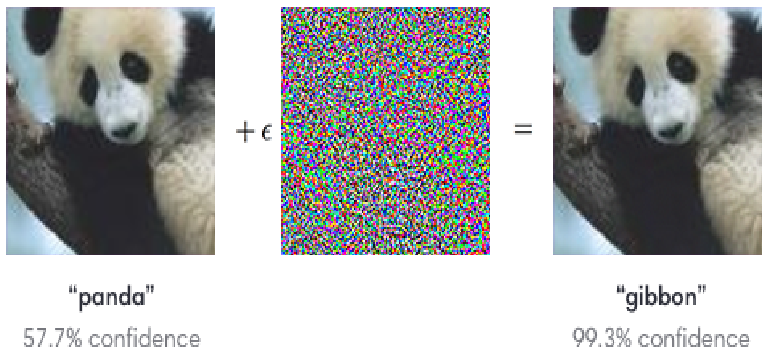


Figure 1.1: Example of Adversarial Example [1].

For example, as seen in Fig 1.1, a small perturbation introduced by noise can cause a given neural network to reclassify a panda as a gibbon. Likewise, in our domain, by adding a few more packets, or slightly changing the data rate, an adversary could cause their malicious traffic or malware to go undetected. As we will show in this work, this is not only possible, but it can have devastating effects on a system.

Making systems robust to adversarial examples has taken off within the last six years since Szegedy et al. [66] first described them in 2013. However, in quick succession to different proposed defence proposed, new attacks often immediately defeat them. As a result, creating a system that is provably robust to adversarial examples (especially in our context where people's private information and money is on the line) is of dire importance.

Despite many works finding that neural networks are inherently vulnerable to adversarial attacks [63], [67], recently a work by Lecuyer et al. [47] showed that by using differential privacy and an alternative prediction scheme, they could make a neural network ***provably robust*** to adversarial examples. Differential privacy is ordinarily a means of constraining the information disclosure of individual members of a database. However, in this context, Lecuyer et al. [47] showed that its properties could also be used to make models more robust. At the cost of some accuracy, Lecuyer et al. [47] showcased that a system can be made ***provably robust*** by incorporating differential privacy during training and by making multiple predictions when labelling. This is where our contributions come in; we seek to combine the problem of adversarial weakness to adversarial examples within the networking setting to the solution of differential privacy.

1.1 Our Contribution

Now that we have outlined the problem that we seek to address (neural networks that detect malicious traffic are vulnerable to adversarial examples) and its importance, we now give a succinct overview of our contributions.

1. We implement several different neural networks that differentiate malicious traffic from benign traffic for the UNSW-NB15 [53] and CSE-CIC-IDS2018 datasets [4] and that differentiate malware traffic from benign traffic for the USTC-TFC2016 [75] dataset. Using our methodology, we manage to achieve

98%, 99.89%, and 99.80% on the UNSW,USTC, and CSE-CIC datasets respectively.

2. We implement two different architectures to show that more fine-grained analysis of these datasets can be conducted in order to differentiate different types of malicious/malware flows.

3. We illustrate our two best models' vulnerability to both black-box and white-box types of attacks. We particularly show that an adversary could potentially transform over 40% of the benign flows in a test into appearing malicious within the USTC dataset using black box methods. This same adversary could also transform at least 10% of malicious flows into appearing benign using only black box methodologies in all considered datasets. For the CSE-CIC dataset we can transform over 75% of the malicious flows into appearing benign. We then further illustrate how a more knowledgeable adversary in a white-box setting could improve upon these results to transform over 30% of UNSW malicious flows into appearing benign, over 80% of USTC's benign flows into appearing malicious, and 5% of CSE-CIC benign flows into appearing malicious. We present attacks of this sort for both the l_1 and l_2 norms.

4. We illustrate targeted attacks on our networks to show that a knowledgeable adversary could attack single statistical features in order to create adversarial examples. We then correlated these changes in features to real malware.

5. We showcase how differential privacy can be used in order to make models robust to adversarial examples. Namely we illustrate that adding noise whilst training and using a multiple iteration prediction scheme can make our models more robust to adversarial examples. We illustrate this defence for both the l_1 and l_2 norms.

6. We show that other methods of adding differential privacy using Poisson subsampling to our models can lead to differing and sometimes better results.

7. We show that a metric used within our differential privacy approach can be used a form of prioritization for investigating malicious flows.

8. We finally show that using our approach, an adversary would be forced to change a given malware by a large amount in order to create an adversarial example against a given model. As a result, this largely becomes a defunct option for an adversary.

This is the first work that we are aware of that incorporates differential privacy for robustness to the area of malware detection. Now that we have outlined our contribution, we now give the background necessary in order to be able to understand our work.

Chapter 2

Background and Related Work

In this chapter, we give an overview of the background necessary for this project. This project encompasses the areas of network security, neural networks, adversarial examples, and differential privacy, and data-analysis algorithms. We shall go through each of these areas in turn.

2.1 Network Security: Malware, Network Attacks, and Network Traffic Analysis

Malware (malicious software) violates users' privacy, steals passwords, and even encrypts users' files for ransom. Network-based attacks like denial of service attacks, botnets, and worms have also wreaked havoc by preventing access to websites and shutting down services. These types of attacks often make use of online services, personal computers, and mobile devices to attack the integrity of systems. Furthermore, the technology for creating malware and network attacks have proliferated as they become accessible through the internet. This has all exacerbated the problem of finding a solution to detecting and preventing network and malware attacks. A lot of this research has focused on network traffic analysis.

By analysing network traffic, network operators can detect and minimize the damage perpetrated by malware and network attacks. As a result, many different types Intrusion Detection Systems (IDS) have been developed. Traditional methods have long focused on signature-based approaches [62]; however, this inherently means that this anti-virus software will be unable to detect malicious network flows if they change their behaviour. Increasingly, though anomaly-based approaches have supplemented signature-based attacks within IDS systems. These network-based and host-based approaches check, and match known patterns of types of malicious traffic patterns. The advantage of this approach is that these systems can detect and thus help prevent unknown attacks (this of course depends on collecting a large trove of appropriate data on which to train data).

2.1.1 Malware and Network Traffic analysis

Network traffic data analysis can be broken down into two main categories: packet-based and flow-based [39]. Flow-based analysis attempts to infer sensitive information from communication patterns such as statistical patterns, packet timings, and packet sizes. Packet based approaches attempt to learn sensitive information from headers and packet payloads. Flow-based approaches are generally the more scalable and efficient since they do not need to check the details of each packet. (Note for the rest of this work, we consider flows to be bidirectional. This means that flows that have the same interchange client and server IPs and the same client and server ports are part of the same flow)

Packet-based approaches are complicated by the encryption of traffic. As a result of this trend, most content-based network traffic analysis has been rendered infeasible.

Traditional pattern matching cannot be readily applied to this problem. Furthermore, given the scale of network traffic, much recent work has focused on the more efficient and feasible area of network flow analysis [65]. Increasingly, a large amount of work has gone into network-flow based approaches that use information like packet timings, packet sizes, Transport Layer Security (TLS) handshake information, domain names from Sever Name Indication (SNI), among a host of other information to characterize flows.

As a result of these changes, detecting malicious network flows in encrypted flows has risen in popularity. Recent works like Malalert [59] and Detection of Malicious Traffic Using Benford's Law [65] have made use of network-level traffic features in order to detect and classify different types of network traffic. Bartos et al.'s [29] approach also uses statistical features representations computed from network traffic to recognize malicious behaviour. They manage to achieve a representation invariant to the most common changes of malware behaviour by constructing a feature histogram for each group of HTTPS flows. TrafficAV [71] also proposes a server-side approach using decision trees to detect Android malware. In addition, to the above approaches, Drebin et al. [27] proposes a lightweight static analysis of network traffic for identifying malware traffic from Android mobile devices.

2.1.2 Feature Extraction

In creating a feature set to analyse network flows, several different approaches have been advocated. Manual feature involves expert defined features being extracted However, increasingly with the advent of deep neural networks, automatic feature extraction has been advocated as a means of avoiding this intensive and imperfect practice.

Manual Feature Extraction

Manual feature extraction depends on a list of expert-defined features extracted from network flow data in order to train models. Malalert [59] extracts statistical information from flows based solely on the number of bytes transmitted. Relying on only five statistical features they manage to achieve an F1 score of as high as 0.94 on a subset of their considered data. In their methodology they extraction information such as the minimum, maximum, mean, mean absolute deviation, kurtosis, skewness, standard deviation, variance, and various quantiles from byte information. Similarly, Kheir et al. [45] use high-level network features available in NetFlow. Malclassifier [25] extracts network flow behaviour during the malware's various infection stages, and classifies traffic based on that sequence behaviour. Lopez et al. [49] use packet timing information, source ports, destination ports in order to classify flows as either malicious or benign. To counteract the inability to read network packet information, Anderson et al. [26] propose a data omnia approach. In this approach, they extend flow records to contain all metadata about a flow, such as the unencrypted TLS handshake information and pointers to contextual flows. They also correlate DNS responses with TLS flows based on the destination addresses. Separately Anderson et al. also conduct a study of various machine learning algorithms to understand their underperformance on this task. They found that inaccurate ground truths and a highly non-stationary data distribution were the two major factors in underperformance. They also found that feature engineering by iterating on the initial feature set was key in increasing performance. Finally, Disclosure [31] extracts flows information, client access behaviour, and temporal patterns from NetFlow data to detect botnets.

Automatic Feature Extraction

Despite the large amount of work that has gone into detecting malicious network flows using predefined features, several works have focused on extracting network flow information directly from the encrypted network information. In this way, they do not have to work about losing information from only putting in these engineered feature sets. Wang et al. [76] use raw flow data from given flows in order to classify the flows as either malicious or benign. Prasse et al. [60] also used a LSTM neural network to detect malware in encrypted traffic, beating the previous state-of-the-art random-forest method.

Semi-Automated Feature Extraction

Semi-automated feature extraction involves making use of statistical features and automatic feature extraction. Vu et al. [68] make use of statistical data while also making use of header information for automatic feature learning. Bhat et. al. [30] also make use of spatial-temporal information in an 18-layer Resnet while also inputting seven different statistical features.

Conclusion

Detecting malware in encrypted network flows has a rich and developing background literature. We shall now explore the machine learning and deep learning algorithms that have been used in network traffic analysis, malware detection, and more specifically in this work.

2.2 Machine Learning, Neural Networks, and Adversarial Examples

In this section we give an overview of machine learning, neural networks, and adversarial examples. Neural networks are often used as classification tools and in this work, we use them as a means of differentiating different types of network flows. Thus, in this section, we firstly give background on the network that we used as well as some methodologies we used with neural networks to improve classification in this work. Secondly, we give an overview of adversarial examples. Adversarial examples arise from small perturbations in input that result in drastically different classifications.

2.2.1 Definition of Classification with Machine Learning

Machine learning is a means of analysis where systems learn, identify, and make decisions based on data. Machine learning furthermore is often used for classification. Here, we give a formal definition of the classification problem for machine learning.

In this problem, we are given a database $\mathcal{D}$ with N tuples each having a feature set $x \in [-1,1]^d$ and a ground truth label $y \in \mathcal{Z}_K$ with K possible categorical outcomes. Each y is a one-hot vector of K categories $y \in y_1, ..., y_K$. A single true class label $y_x \in y$ given $x \in D$ is assigned to only one of the K categories [11].

A machine learning model with parameters θ on a d-dimensional input x then is a function $f : R^d \rightarrow R^K$ that maps input to a vector of scores $f(x) = \{f_1(x), ...f_K(x)\} s.t. \forall k \in [1,K] : f_k(x) \in [0,1]$ and $\Sigma_{k=1}^{K} f(k) = 1$. The class with the highest score value is then selected as the predicted label for the input x, denoted as $y(x) = max_{k \in K} f_k(x)$ [11]. A model is said to correctly label a single instance x if $y(x) = y_x$ or that the predicted labels is equal to the ground truth label. Otherwise, the system is said to incorrectly classify the instance x. In a binary system that identifies malicious flows (port scans, botnets, backdoors, etc), this is such that benign flows are labelled as benign and that malicious flows are labelled as malicious. Similarly, in a multi-class system with K labels, this is such that a given instance x is assigned the correct label within all K labels.

2.2.2 Machine Learning: Random-Forests

Random forests are a combination of tree predictors such that each tree is initialized from the values of a randomized vector sampled independently from a given distribution [32]. Specifically, each tree predictor is a series of decisions that are used to perform finer and finer grained analysis (i.e. Is this object a fruit? Is this fruit round? Is this fruit red? Is this fruit an apple?). Random forests thus consist of many individual decision trees that act as ensembles in order to make a decision. Each decision tree reports a class prediction and the class with the most votes is the model's predictions. Each of these trees is initialized differently and acts independently, leading to robust predictions. For more information on random forests see [32].

2.2.3 Neural Networks: Multi-layer Perceptrons

Multi-layer perceptrons (MLP) are a class of feed forward neural networks. MLPs consist of input layer, one or more hidden layers that extract features, and an output layer. Each hidden layer within an MLP is composed of multiple nodes that consist of a linear function composed with a nonlinear activation function [70].

$$f(x) = \sigma(W \cdot x + b)$$

where $\sigma(\cdot)$ is the nonlinear activation function (i.e. RELU(x) or tanh(x)). The final layer outputs the result of the last hidden layer by usually putting the output through a *SoftMax* layer that gives a set of probabilities for each class.

2.2.4 Convolutional Neural Network

Convolutional neural networks (CNN) are particularly excellent at extracting 'strong signals' from data and thus are often used in classification [30]. CNNs utilize layers with convolving filters that are applied to local features and thus extract more complicated features from these local features, incorporating them in different ways. Convolutional layers accomplish this by making use of an input translation invariance (i.e. features appearing in multiple places). This allows locally connected feature convolve over the entire feature map. Pooling layers are often used after convolving layers within CNNs. Pooling layers combine adjacent outputs from feature maps and output the maximum. In this way, pooling layers down sample convolutional layers in order to the strongest signals.

Finally, fully-connected layers are the end of networks are dense layers that have every output neuron of the last layer as input to each neuron in the current layer. In most CNNs a SoftMax layer in then used as the final layer to output a probability distribution over the possible classes.

Dilated Causal Convolutions

While training CNNs, the receptive fields for the convolutional layers are often fairly small. For example, in the model used within this work, the receptive field for each convolutional layer is only 3. For long-term temporal understanding in CNNs, *dilated convolutions* are often used. Dilated convolutions are convolutions that skip inputs at a given dilation rate. Dilations thus allow networks to take a coarse and wider view of the network. For time-data specifically, this allows a better understanding of relationships amongst the data without increasing the number of parameters and the training cost.

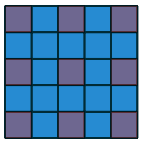


Figure 2.1: Example of dilated causal convolution with dilation rate of 1.

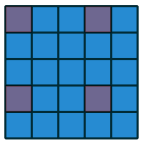


Figure 2.2: Example of dilated causal convolution with dilation rate of 2.

Dilated causal convolutions are often paired as in Bhat et al. [30] with causal convolutions and padding. Causal padding restricts every neuron to look only at the previous time-steps when performing the convolutions.

2.2.5 Adversarial Examples

A major issue with using neural networks within this domain is that they are weak to adversarial examples; an adversary trying to fool a neural network can easily do so. Adversarial examples are derived from small perturbations that are seemingly imperceptible, but that cause the classifier to predict a wrong label. Adversarial examples can be generated in a variety of ways. One of the most popular fast gradient sign method can achieve a 90%+ misclassification rate against many classifiers [41]. In this section, we first define adversarial examples formally, explain why they exist, and then explain the various sorts adversarial attacks.

Formal Definition: Adversarial Examples

We begin by formally defining adversarial examples. For adversarial attacks, for a target model f and inputs (x, y_x), the adversary's goal is to find an **adversarial example** $x_{adv} = x + \alpha$ where α is a perturbation induced by the adversary. The α chosen by the adversary is such that

1. x_{adv} are close according to a chosen norm

2. the model misclassifies x_{adv} s.t. $y(x_{adv}) \neq y(x)$.

The definition of adversarial examples requires our use of a distance metric to quantify the similarity between two sets of features. There are four widely used distance metrics that are used in generating adversarial examples we will consider in this work. Each of these four metrics are L_p norms.

L_p norms are written as $||x - x'||_p$, where the p-norm $||\cdot||_p$ is defined as follows:

$$||v||_p = \left(\Sigma_i^n |v_i|^p\right)^{\frac{1}{p}} \tag{2.1}$$

The four metrics are then as follows:

1. L_0: This distance metric measures the number of features x_i such that $x_i \neq x_i'$ in two images x and x'. This thus corresponds to the number of features that have been altered in a given image.

2. L_1: This distance metric measures the sum of the absolute value of difference between each feature x_i and x_i'.

3. L_2: This distance metric measures the Euclidean (root-mean squared) distance between two images x and x'. The L_2 norm can remain relatively small while changing many features

4. L_∞: This distance metric measures the maximum change to any of the features such that $L_\infty = \max(|x_1 - x_1'|, ..., x_n - x_n'|)$.

Within this work, we focus specifically on the L_1 and the L_2 metrics.

Why Do Adversarial Examples Exist?

Adversarial examples were first described by Szegedy et al. [66] in 2013. Since then there has been a plethora of research in generating and protecting against these attacks. In addition to this literature, in the last two years, there has been significantly work in determining why adversarial examples actually exist.

In the simplest argument, a neural network maps its input to a multi-dimensional space that it then divides with hyperplanes into regions that belong to given classes. More formally, a neural network gives m hyperplanes of the form $\Sigma_{j=1}^{n} a_i^k x_j + b_i$ for $i = 1, ...m$ which split $\mathbb{R}^n$ into *cells* [63]. If we denote by M the $m \times n$ matrix whose entries are the a_i^j coefficients and by B the column vector whose entries are the b_i constants, then each cell in the partition is defined by a particular vector S of $m\pm$ signs, and consists of all the points $\mathbf{x}$ in $\mathbb{R}^n$ for which $M\mathbf{x}+\mathrm{B}$ is a column vector of m size whose signs are as specified in S [63]. The maximal possible number of cells is then $\Sigma_{i=0}^{n} \binom{m}{i}$ [63]. The predictor is then able to associate labels (malicious vs benign to the cells). However, because even a small number of hyperplanes is enough to divide $\mathbb{R}^n$ into a huge number of cells, there exists a hodgepodge of cells surrounded by other different class cells. Given the complexity of neural network, a small perturbation is thus sometimes enough to move a cell in one class to another cell of another class. This is true then of every cell. For a clear picture of what this looks like see Fig. 2.3

Put in another way, due to the complexity of how neural networks differentiate different classes, it often learns insignificant details in data causing small perturbations of these details to drastically change classifications. The above explanation however, is not the full picture and only gives an intuitive understanding. Recently in 2019, Shamir et al. [63] showed that targeted attacks for the l_0- norm always exist due to the geometry of $\mathbb{R}^n$. For more details see Shamir et al [63].

We now give a brief overview of several different adversarial example algorithms that have been proposed in the last five years.

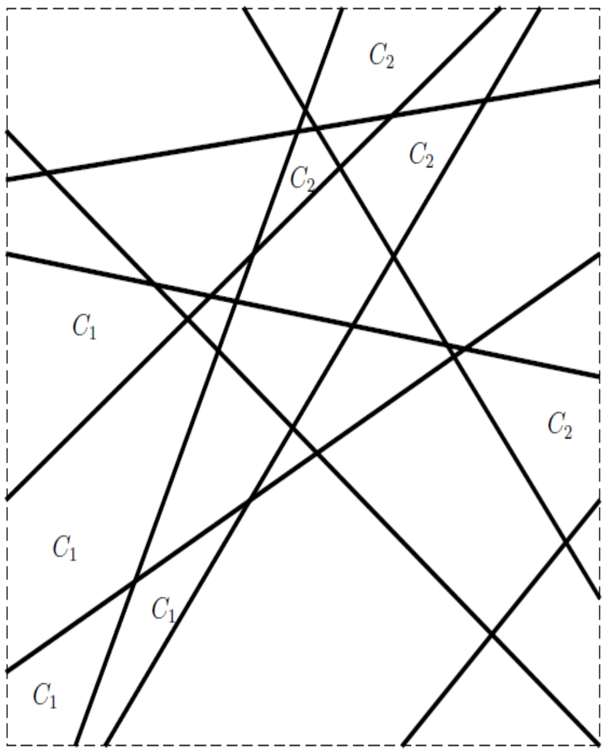


Figure 2.3: Example of classification cells that neural networks build [63].

Fast Gradient Sign Method

The fast gradient sign method proposed by Goodfellow et al. [41] is the most basic form of creating adversarial examples. Let θ be the parameters of a given model, x the input to the model, y the ground labels associated with x, and J(θ,x,y) the cost function used to the train the neural network. If the cost function is linearized around the current value of θ, the optimal max-norm constrained perturbation is given by $\eta = \epsilon sign(\nabla_x$ J(θ,x,y)) .The *epsilon* value refers to the magnitude of the change possible on each of the features within x. FGSM thus works by finding adversarial perturbations which increase the value of the loss function.

Projected Gradient Descent/Basic Iterative Method

The basic iterative method is an extension of FGSM [46]. It applies FGSM multiple times with small step sizes:

$$X_0^{adv} = X$$
$$\mathrm{X}_{N+1}^{adv} = Clip_{X,\epsilon}\big\{X_N^{adv} + \alpha sign(\nabla_X J(X_N^{adv}, y_{true}))\big\}$$

where $Clip_{X,\epsilon}(M)$ denotes the element wise clipping of M, with $M_{i,j}$ clipped to the range of $X_{i,j} - \epsilon, X_{i,j} + \epsilon$. The value of α is the value by which each feature is changed on each step. The number of iterations is a constant that can be changed for better accuracies.

This method is essentially what is also known as projected gradient descent (PGD). A variation of this type of attack that we consider in this work restarts the projected gradient from many points in a l_{inf}-ball surrounding individual attack points. This allows PGD to explore a larger part of the loss landscape [51].

One-step target class method

A different method than FGSM and the basic iterative method is the one-step target class method. This method seeks to maximize the probability $(y_{target}|X)$ of some specific target class y_{target} that is different that the ground label. In a neural network with cross-entropy loss, this leads to the following formulation of the one-step target class method

$$X^{adv} = X - \epsilon(\nabla_X J(X, y_{target}))$$

Kurkani et al. [46] recommend using the least likely class as the target when using this methodology in a multi-class setting.

Iterative least-likely class method

The iterative least-likely class method is an extension of the one-step class method and is noted for being generally highly effective [46]. It runs multiple iterations of the one step class method:

$$X_0^{adv} = X$$

$$\mathrm{X}_{N+1}^{adv} = Clip_{X,\epsilon}\big\{X_N^{adv} + \alpha sign(\nabla_X J(X_N^{adv}, y_{least_likely}))\big\}$$

where $Clip_{X,\epsilon}(M)$ denotes the element wise clipping of M, with $M_{i,j}$ clipped to the range of $X_{i,j} - \epsilon, X_{i,j} + \epsilon$.

Sparse l_1 Descent

The sparse l_1 descent attack was proposed by Tramer et al. [67] in 2019. While the projected gradient descent algorithms proposed above are efficient for l_1 attacks, they are highly inefficient in producing strong l_2-perturbations. This is because in the l_1-case, the steepest descent is only a unit vector for a given feature. As a result, each iteration of the attack only updates a single feature $\mathbf{r}$. In contrast the sparse l_1 attack has finer control over the sparsity of the update steps and updates multiple features at a time.

Tramer et al. propose for $q \in [0, 1]$, let $P_q(|\mathbf{g}|)$ by the q^{th} percentile of $|\mathbf{g}|$ where $|\mathbf{g}|$ is the gradient of the loss. Trammer et al. then set $e_i = sign(g_i)$ if $|g_i| > P_q(|\mathbf{g}|)$ and 0 otherwise. Tramer et al. then normalize $\mathbf{e} = [e_i, ...e_R]$ to the unit l_1-norm. For $q \gg 1/d$, this method thus updates many indices of $\mathbf{r}$ at once. Sparse l_1 descent also introduces an optimization for clipping by ignoring gradient components where the step cannot increase or decrease (i.e., the update would cause the feature to move beyond the input domain. By performing these optimizations, the sparse l_1 descent attack vastly outperforms projected gradient attacks and is competitive with the more expensive elastic net attacks (we do not discuss this attack here). For more details on the sparse l_1 attack see Trammer et al [67].

HopSkipJump Attack

The HopSkipJump based attack is a decision-based adversarial attack that generates adversarial examples based only on the output labels returned by the targeted model. This attack is optimized for the l_2 and the l_{inf} similarity metrics.

Decision based attacks work mainly using rejective sampling. In this setting, a feature vector firstly is initialized that lies in the target class. At each step a perturbation is sampled from a given distribution that reduces the distance of the perturbed image to the original feature vector. If the perturbation keeps the image in the target class, the perturbation is kept otherwise the perturbation is dropped [34].

The HopSkipJump Attack combines this decision with the problem of zero-order optimization. Zero-order optimization is the problem of optimizing a function f based only on access to function values $f(x)$.

We now give a brief overview of how hop-skip-jump's optimization framework for targeted attacks. For a more detailed view on the iterative algorithm for given l_p-metrics see Chen et al. [34]. The first component is a discriminative function $F : \mathbb{R}^K \to \mathbb{R}^m$ that accepts an input $x \in [0,1]^d$ and an output $y \in \Delta_m := \{y \in [0,1] | \Sigma_{c=1}^{K} = 1\}$. The output vector y $= F_1(x), ..., F_K(x)$ is then a probability distribution over the label set $[m] = 1, ..., K$. As usual, the class with the maximum probability is then the assigned class label. A targeted attack then seeks to change the original assigned label c to a pre-specified label $c^* \in [m] \setminus c$. More formally this can be defined using the function $S_{x^*} : \mathbb{R}^d \to \mathbb{R}^K$ s.t.:

$$S_{x^*}(x') := F_{c^*}(x') - max_{c \neq c^*} F_c(x')$$

A perturbed image x' is then a successful attack if and only if $S_{x'}(x') > 0$ and the boundary between successful and unsuccessful perturbed images is

$$bd(S_{x'}) := \{z \in [0,1]^d | S_{x'}(z) = 0\}$$

Finally, the HopSkipJump attack defines as an indicator a Boolean valued function $\Phi_{x'} : [0,1] \rightarrow \{-1,1\}$ where $\Phi_{x'}(x) = 1$ when the adversarial example is successful and $\Phi_{x'}(x) = -1$ when it is unsuccessful. The HopSkipJump attack can then formulate the attack as minimizing $min_{x'}(d(x,x')$ such that $\Phi_{x'}(x) = 1$. The HopSkipJump attack can then be formulated as generating a sample x' by only querying the classifier C alone.

Black-Box Attack Via Substitute Model Training through Data-Augmentation

We now give a brief overview of a means of performing black-box attacks via substitute model training on neural networks.

In this framework, an adversary, as in the Hop Skip Jump attack, seeks to craft inputs misclassified by the machine learning model by only accessing the labels assigned to any chosen input $\mathbf{x}$. The adversary seeks to do this by generating a fake dataset FD and then observing how the original model labels the data. A substitute model S is then trained using this synthetic data and the labels obtained from the original model. By then running an adversarial attack against the substitute model, a portion of adversarial examples are expected to be transferable between the architectures.

Substitute Model Training: In Papernot et al. [56], it was found that the substitute deep neural network architecture has 'relatively little impact on the success of the attack'. For this reason, here we focus on how the synthetic dataset is generated.

The heuristic outlined by Papernot et al. [56] involves identifying the input areas in which the model's output varies. Once these areas are identified, more input-output pairs are generated in this vicinity to capture how the output of the original model

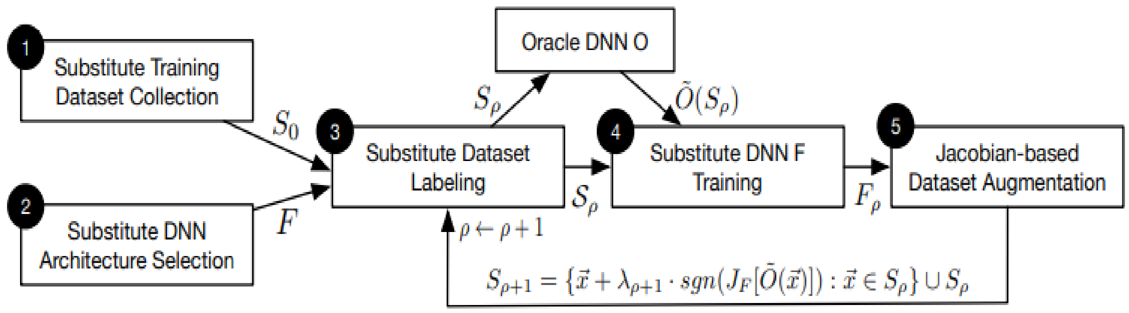

1
Substitute Training Dataset Collection
2
Substitute DNN Architecture Selection
S_0
F
3
Substitute Dataset Labeling
Oracle DNN O
S_ρ
$\tilde{O}(S_\rho)$
4
Substitute DNN F Training
S_ρ
F_ρ
5
Jacobian-based Dataset Augmentation
$\rho \leftarrow \rho+1$
$S_{\rho+1} = \{\vec{x} + \lambda_{\rho+1} \cdot sgn(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$

Figure 2.4: Flow of how black-box model is trained, and adversarial examples are developed. Figure taken from [56].

varies. These areas are identified using the substitute models' Jacobian matrix, which is evaluated at several input points x. More specifically, an adversary determines how the sign of the Jacobian matrix corresponds to the label assigned to the x. To obtain a new fake datapoint, a parameter λ is multiplied by the sign of the Jacobian and this is added to the original x. Papernot et al. [56] calls this methodology *Jacobian-based Dataset Augmentation*.

2.2.6 Defences against Adversarial Examples

Within the literature, there is a host of different protections that have been proposed to make models robust to adversarial examples. Unfortunately, each of these defence has been almost immediately defeated in turn. As previously described, adversarial examples are an inherent aspect of neural networks. Here, we give a brief overview of the different methods that have been proposed to make models more robust to adversarial examples.

Increasing the Capacity of the Model

Madry et al. [51] found that increasing the capacity (the amount of neural network parameters) can make a model more robust to adversarial examples (only for one-step perturbations).

Adversarial Training

Another method that has been proposed is using adversarially produced examples while training [46]. This is known as adversarial training. These adversarial examples are usually produced using the Fast Gradient Sign Method in combination with other adversarial attack methods. This methodology, however,, often causes the model to overfit to these adversarial examples. Kurkakin et al. labelled this as 'label leaking' [46].

Defensive Distillation

Defensive distillation is a method that seeks to increase adversarial robustness independently of any possible attack. Distillation works by first smoothing the SoftMax layer of a trained model that is used for classification by a constant T. The new model is then trained again but instead of using the original labels, the probability vectors from the original model are used as soft targets [57]. This allows for a smoother loss function. This type of approach can also be achieved by smoothing the labels.

Feature Squeezing

Another method that has been proposed is feature squeezing so that the features are not sensitive to perturbations [77]. Feature squeezing works by smoothing features so that many are mapped to the same value, making the model robust to noise. This however makes the system less accurate as a natural consequence.

Collaborative Multi-Task Training

Another method that has been proposed is based of collaborative multi-task training. This approach first encodes training labels into pairs and then counter black-box attacks by leveraging adversarial training using these pairs as the labels. The defence then constructs a detector to identify and reject high-confidence adversarial examples that can evade a black-box defence [69].

All the above methods do not provide ***provably robustness*** protection against adversarial examples. Namely there is no mathematical reason why they should be robust to attacks. We now however, present differential privacy, which can be used to provide ***provably robustness*** to adversarial examples.

2.3 Differential Privacy

Differential privacy is a means of constraining the information disclosure of individual members of a database when it is acted upon by an algorithm $\mathcal{A}$ [40]. In the formulation of differential privacy, this randomized algorithm takes as input database d, performs an algorithm $\mathcal{A}$ and outputs a value in the space O. This could be for example, taking in a database of heights of all University of Oxford students, performing the algorithm for averaging, and outputting this floating-point average. In order to prevent the information disclosure of the individual members, differential privacy mechanisms inject noise into the database algorithm $\mathcal{A}$ before or after the algorithm's operation in order to obfuscate private information of individual members. There are several different types of differential privacy. In the forthcoming section, we shall elaborate on the two types of differential privacy that are relevant to our project.

2.3.1 $\epsilon - DP$.

A randomized algorithm $\mathcal{A}$ fulfils $\epsilon - DP$ with respect to a metric ρ if for any two if for any two databases d and d' where $\rho(d, d') \leq 1$, all outputs $\mathcal{O} \subseteq \mathit{Range}(A)$ obey [40]:

$$Pr[A(\mathcal{D}) = \mathcal{O}] \leq e^{\epsilon} Pr[A(\mathcal{D}') = \mathcal{O}] \tag{2.2}$$

In the above equation, $\epsilon > 0$ controls the amount by which the distributions by $\mathcal{D}$ and $\mathcal{D}'$ may differ. The smaller the value of $\epsilon > 0$, the stronger the privacy guarantee. For small values of ϵ, differential privacy guarantees creating a small perturbation in the database cannot change the output of the algorithm $\mathcal{A}$ by a large value.

Note that differential privacy can be applied to general metrics $\rho(\mathcal{D}, \mathcal{D}') <= 1$ where ρ can be any l_p-norm [47].

2.3.2 $(\epsilon, \delta) - DP$.

A randomized algorithm $\mathcal{A}$ fulfils $(\epsilon, \delta) - DP$ with respect to a metric ρ if for any two if for any two databases d and d' where $\rho(d, d') \leq 1$, all outputs $\mathcal{O} \subseteq \mathit{Range}(A)$ obey [40]:

$$Pr[A(\mathcal{D}) = \mathcal{O}] \leq e^{e} Pr[A(\mathcal{D}') = \mathcal{O}] \tag{2.3}$$

Here $\epsilon > 0$ and $\delta \in [0, 1]$ and as with $\epsilon - DP$, for $(\epsilon, \delta) - DP$, the smaller the values of ϵ and δ, the greater the privacy guarantee. δ here is a broken probability.$(\epsilon, \delta) - DP$ is thus a relaxation of $\epsilon - DP$, allowing for more graceful increases in privacy loss, and thus is known as differential privacy with advanced composition.

2.3.3 Privacy amplification via subsampling

Subsampling is often used in the design of differential privacy mechanisms [28]. The 'privacy amplification by subsampling' principle ensures that a differentially private mechanism acting on a random subsample ensures higher privacy guarantees than for an entire population. Particularly in machine learning, where quantifying privacy loss is of the essence, subsampling has become the basis of several different algorithms. Each type of subsampling (i.e. sampling without replacement, sampling with replacement, Poisson subsampling) within differential privacy provides different privacy guarantees. In particular in this work, we focus on one of the of most well-known: Poisson subsampling. For more information for concerning the other types of subsampling see Balle et al. [28].

Poisson subsampling

Poisson subsampling with respect to remove/add-one relation is one of the most studied forms of subsampling within the differential privacy literature [28]. The Poisson subsampling mechanism $S^{po}_{\gamma} : 2^U \rightarrow P(2^U)$ takes a set x and outputs a probability distribution $w = S^{po}_{\gamma}(x)$ for a sample y. This distribution is supported on all set $y \subseteq x$ given by $w(y) = \gamma^{|y|}(1 - |\gamma|)^{|x|-|y|}$. This is equivalent to adding to y each element in x with probability γ [28].

Given a differentially private algorithm $\mathcal{A}$ with a given privacy value ϵ, the privacy profile of the subsampled algorithm can be bounded as follows:

Let $\mathcal{A}' = \mathcal{A}^{S^{po}_{\gamma}}$. For any $\epsilon \geq 0$, the new privacy value of the new algorithm is $\epsilon' = log(1 + \gamma(e^{\epsilon} - 1))$. For the values that we are considering, this value is nearly $\epsilon' = \gamma\epsilon$ We will not give the full proof for this result, but for more details see Li et al [48].

2.3.4 Differential Privacy Properties

Differential privacy has become increasingly popular not only due to the guarantees that it provides on the output of any algorithm $\mathcal{A}$ but also due several theoretical properties that have enabled a rich literature to evolve.

Post-processing property

The post-processing property states that any computation to the output of an (ϵ,δ)-DP algorithm remains (ϵ,δ)-DP [47]. Note that because this property holds for(ϵ,δ)-DP, it also holds for (ϵ,0)-DP /ϵ-DP. This property is explicit in the definition of differential privacy. It states that after a differentially private algorithm has been run on database, no other computation can recover the details hidden by the differentially private noise that was added. (Note that this a key property upon which this work rests.)

Expected Output Stability Bound

The expected value of an (ϵ,δ)-DP algorithm with a bounded output is not sensitive to small changes in the input [47]. Because the expected output stability bound is more implicit in the definition of differential privacy and was only formally provided by Lecuyer et al. [47], we provide the proof of this property here. (Note that this is the second property is the crux on which much of this work depends. We use this property in order to prove the robustness our neural networks to adversarial examples.)

Definition: *Suppose a randomized function A, with bounded output A(x) $\in$ [0,b], b $\in \mathbb{R}+$ satisfies (ϵ,δ)-DP. Then the expected value of its output meets the following property:*

$$\forall \alpha \in B_p(1) \cdot \mathbb{E}(A(x)) \leq e^{\epsilon}\mathbb{E}(\mathbb{A}(x+\alpha)) + b\delta$$

The expectation is taken over the randomness in $\mathcal{A}$ [47].

Proof: Consider any $\alpha \in B_p(1)$ and let $x := x + \alpha$. We write the expected output as:

$$\mathbb{E}(A(x)) = \int_0^b P(A(x) > t)dt$$

Applying the equation for (ϵ,δ)-DP we then see that:

$$\mathbb{E}(A(x)) \leq e^{\epsilon}(\int_0^b P(A(x) > t)dt) + \int_0^b(\delta\, dt)$$
$$= \mathrm{e}^{\epsilon}\mathbb{E}(A'(x)) + \int_0^b(\delta dt)$$

Since δ is a constant $\int_0^b(\delta dt) = b\delta$.

2.4 Differential Privacy in Machine Learning

Differential privacy has increasingly been used in machine learning in order to protect the sensitive information on which models are often trained. Instead of trying to learn the characteristics of individual members of a dataset, differential privacy forces machine learning algorithms to learn general characteristics about the dataset through the insertion of random noise [43].

Algorithm 1 Add Noise

```
1:  Training Data Input$(X, y)$
2:  #1 Add noise to training data X: Input Perturbation
3:  #1.5 Subsample training data (Must be conjunction with noise)
4:  Result Model parameters $\theta$
5:  $\theta \leftarrow Init(0)$
6:  $J(\theta) = \frac{1}{n}{}^{n}_{i=1} l(\theta, X_i, y_i) + \lambda R(\theta) + \beta$
7:  for epoch in epochs
8:      #2 Add noise to gradient: Gradient Perturbation
9:      $\theta = \theta - \eta(\nabla J(\theta) + \beta)$
10: end for
11: #3 Add noise to output: Output Perturbation
12: return $\theta + \beta$
```

As seen in Algorithm 1, there are several places where noise can be added to machine learning algorithm in order to add differential privacy to its calculation. Noise can be added directly to the input X to get an *Input perturbation*, to the noise gradient in order to get a *Gradient Perturbation*, and to the output of the algorithm in order to achieve an *Output Perturbation.* In addition to these three mechanisms, the input can be subsampled in order to minimize the privacy parameter used. This only gives limited examples of ways to achieve differential privacy, in machine learning; in addition to the above, other methods include the sample aggregate framework [54], the exponential mechanism [52], and the teacher ensemble mechanism [58].

2.4.1 Differential Privacy and Adversarial Examples

While differential privacy has been mainly used to protect sensitive training data from exploitation, in the recent work of Lecuyer et al. [47], differential privacy was used in a rather novel way. Instead of bounding the amount of privacy loss that individual training data members could anticipate, Lecuyer et al. use differential privacy in order to construct a lower limit on the amount of perturbation necessary in order to induce a change of classification for a neural network. In this way, by providing such a lower limit, Lecuyer et al. [47] manage to provide a means of certifiably evading adversarial examples on the neural network. In simple terms, there method ensures that small changes in the input of a given machine learning classifier (such as including a few more packets within the malware), will not change the prediction.

Lecuyer et al.'s work is completely different than how differential privacy is ordinarily used within machine learning network. Ordinarily, the goal of differential privacy within machine learning is to learn general characteristics while ensuring that public release of the model parameters is guaranteed to reveal non-significant information about the training set. In contrast Lecuyer et al. focus of creating a robust predictive model, where a small/insignificant change in the input does not result in a different classification. This is such that Lecuyer et al.'s training method is not differentially private, and it does not ensure the differential privacy of the training data.

2.4.2 Satisfying (ϵ,δ)-differential privacy for robustness

Here will elaborate on how in practice differentially-privacy for robustness is added to models.

Differential Privacy Noise Layer

A noise layer is inserted into a neural model using either the Gaussian or Laplace differential privacy mechanism. These both involve inputting either Gaussian or Laplace noise respectively at each one of the features inputted into the model. **Laplace Mechanism:** $noise(\Delta, L, \epsilon, \delta)$ is the Laplace mechanism with mean zero and standard deviation $\sqrt{2}\Delta_{p,1}L/\epsilon$; this gives $(\epsilon, 0)$-DP. L in above mechanism denotes the size of the p-norm attack. Note that in most calculations, the $\Delta = 1$, where Δ is the sensitivity of the model, after normalization.

Gaussian Mechanism: $noise(\Delta, L, \epsilon, \delta)$ has the Gaussian mechanism with mean zero and standard deviation $\sqrt{2\ln\frac{1.25}{\delta}}\Delta_{p,2}L/\epsilon$; this gives (ϵ, δ)-DP for $\epsilon \leq 1$. L in above mechanism denotes the size of the p-norm attack. Note that in most calculations, the $\Delta = 1$, where Δ is the sensitivity of the model, after normalization.

Depending on where noise is inserted in the model, the sensitivity Δ of the other layers will matter. This is because if noise is inserted following a neural dense calculation, adding noise only equivalent to slightly perturbing the weights of the model. As a result, the rest of the model would immediately compensate for this adjustment. For this reason, unless the noise is inputted on the image itself, the sensitivity of the pre-noise layer must be adjusted in order to prevent this from occurring.

As in Lecuyer et al. [47], the sensitivity of a function or layer f can be defined as the maximum change in the input given some l_p norm metric for the input and the output (the p-norm and the q-norm respectively).

$$\Delta_{p,q} = \Delta^f_{p,q} = max_{x,x':x \neq x'} \frac{||f(x)-f(x')||}{||x-x'||}$$

Computing the sensitivity of the pre-noise function f, as stated before, depends on its placement in the network. The placement of the layer has this flexibility because the post processing property of differential privacy guarantees that the output after this layer will remain differential private. We will now explain how to calculate the sensitivity for two places in the network that we considered in this work.

Option 1: Noise Directly on the Features: By placing noise directly on the features, the calculation of the sensitivity is fairly trivial. Here the pre-noise function f is just the identity function and as a result the sensitivity for all norm combinations is just 1.

Option 2: Noise after 1st Layer: Deep neural networks often start with a fully connected layer or a convolutional layer (in our work, a fully connected layer is used, see Chapter 4 for more details). We will now go into how the sensitivity of a fully connected layer is calculated.

A fully connected layer can be thought of as a matrix multiplication with values $W \in \mathbb{R}^{m,n}$. The sensitivity is then the matrix norm defined as: $||W||_{p,q} = \sup_{x:||x||_p \leq 1} ||Wx||_q$. By definition of linearity $\frac{||Wx||_q}{||x||_p} \leq ||W||_{p,q}$, which means that $\Delta_{p,q} = ||W||_{p,q}$. As a result of this, $||W||_{1,1}$ is the maximum 1-norm of W's columns, $||W||_{1,2}$ is the maximum 2-norm of W's columns, and $||W||_{2,2}$ is the maximum singular value of W. In the rest of this work, we only consider 1-norm and 2-norm attacks as inf-norm are difficult to bound, so we do not delve into how to solve for the sensitivity.

Option 3: Noise in Auto-encoder: Noise can also be placed before a deep neural network in a separately trained auto-encoder. Auto-encoders are trained to predict its own input. Often used to de-noise inputs, auto-encoders can be trained to incorporate differential privacy. Auto-encoders can use either of the previous two options whilst training. Once completed, sensitivity is calculated in the same way as the above options After training, this auto-encoder can be stacked before a predictive deep neural network for classification. Because of the post-processing property of differential privacy, the stacked network is also differentially private. This approach in particular allows for the separate training of the deep neural network and the auto-encoder, relieving a great deal of potential experimental work (see Chapter 4 for more details). The noise in the autoencoder can be placed within it using two previous options.

Once the sensitivity is calculated, the noise layers leverage the Laplace and Gaussian mechanism as follows. During training and prediction, the noise layer computes $f(x) + \mathbf{Z}$ where $\mathbf{Z} = (Z_1, ..., Z_m)$ are random variables form a noise distribution defined by $noise(\Delta, L, \epsilon, \delta)$ and f(x) is the pre-noise computation.

2.4.3 Differential privacy robustness

We now give an overview of how differential privacy relates to the robustness of training output. We show that a *DP scoring function* can be constructed such that given an input, the predictions made with respect to features of the input are differentially private and thus robust to perturbations in the input. By relying on the *Expected Output Stability Bound* property of differential privacy, we can give stability bounds on the expected output of the *DP scoring function.* This property can thus give rigorous conditions for robustness to adversarial examples.

More formally, we regard feature values of an input x as the records in the database and consider a randomized scoring function A that on an input x. We then say A maps x to a vector of scores $A = \{A_1(x), ...A_K(x)\} s.t. \forall k \in [1, K] : A_k(x) \in [0, 1]$ and $\Sigma_{k=1}^{K} A(k) = 1$ [47]. The model's original scoring function A can then transformed into a randomized $(\epsilon, \delta) - DP$ scoring function $\mathcal{A}(x)$ using a differential privacy mechanism. We then say that $\mathcal{A}$ is an (ϵ,δ)-feature level differentially private function if it satisfies (ϵ,δ)-DP for a given metric (i.e. l_1,l_2). This is functionally equivalent to saying that the algorithm $\mathcal{A}$ satisfies some form of differential privacy. The *Expected Output Stability Bound* then implies bounds on the expected outcome of an (ϵ,δ)-DP scoring function. This is such that:

Suppose a randomized function $\mathcal{A}$ satisfies (ϵ,δ)- feature differential privacy to a p-norm metric, and where $\mathcal{A}(x) = \{A_1(x), ...A_K(x)\} s.t. \forall k \in [1, K] : A_k(x) \in [0, 1]$ and $\Sigma_{k=1}^{K} A(k) = 1$ then:

$$\forall k \, \forall \alpha \, \in B_p(1) \cdot \mathbb{E}(A_k(x)) \leq e^{\epsilon} \mathbb{E}(A_k(x + \alpha)) + \delta \tag{2.4}$$

This derives directly from the *Expected Output Stability Bound.*

If a prediction procedure f then allows the expected value $\mathbb{E}(\mathcal{A}(x))$of $\mathcal{A}(x)$ is be calculated, then this can be used to provide robustness guarantees and labelling. This is because the expected value is the value x's true probability vector from which to pick the maximum argument and thus the classification [47].

Finally, we give the robustness condition: **Robustness Condition** *Suppose a satisfies (ϵ,δ)- feature differential privacy with respect to a p-norm. Then for any input x, if for some $k \in \mathcal{K}$* [47],

$$\mathbb{E}(A_k(x)) > e^{2\epsilon} max_{i:i \neq k} \mathbb{E}(A_i(x)) + (1 + e^{\epsilon})\delta \tag{2.5}$$

then the multiclass classification model based on label probability vector y(x) = $(\mathbb{E}(A_1(x)) ... \mathbb{E}(A_k(x)))$ is robust to attacks of α of size $||\alpha||_p \leq 1$ on input x.

We repeat the proof presented by Lecuyer et al. [47] here for clarity. For more information, see [47]. *Proof:* Consider any $\alpha \in B_p(1)$ and let $x' := x + \alpha$. From Equation 2.4.3, we have that:

$$\mathbb{E}(A_k(x)) \leq e^{\epsilon}\mathbb{E}((A_k(x')) + \delta \tag{2.6}$$

$$\mathbb{E}(A_i(x')) \leq e^{\epsilon}\mathbb{E}(A_k(x)) + \delta \;\; i \neq k \tag{2.7}$$

Equation 2.6 gives a lower bound on $\mathbb{E}((A_k(x'))$ while equation 2.7 gives an upper bound on $max_{i:i\neq k}\mathbb{E}((A_k(x'))$. Equation 2.10 implies that the lower of the expected sore is higher that the upper-bound for the expected score that for any other label. This then implies that the label is robust to perturbations. More completely [47]:

$$\begin{gathered}
\mathbb{E}(A_k(x')) \geq \frac{\mathbb{E}(A_k(x'))-\delta}{e^{\epsilon}} \\
\geq \frac{e^{2\epsilon}max_{i:i\neq k}\mathbb{E}(A_i(x))+(1+e^{\epsilon}\delta-\delta}{e^{\epsilon}} \\
= \mathrm{e}^{\epsilon}max_{i:i\neq k}\mathbb{E}(A_i(x)) + \delta \\
\geq e^{\epsilon}max_{i:i\neq k}\mathbb{E}(A_i(x')) \\
\mathrm{E}(\mathrm{A}_k(x')) > max_{i:i\neq k}\mathbb{E}(A_i(x+\alpha))\forall\alpha B_p(1)
\end{gathered}$$

2.4.4 Certified Accuracy and Robustness Procedure

We now give the full procedure to get certified accuracy on a particular example and prove robustness to adversarial examples. Ordinarily, for a given input x to a model with function A, the model predicts a label by picking the $arg\, max$ of the SoftMax layer or $f(x)$. As noted before for our labelling, we need access to $E(\mathcal{A}(x))$. As a result, in this work, the prediction procedure differs significantly. As in Lecuyer's et al. [47], the prediction is chosen by invoking the model function $\mathcal{A}(x)$ multiple to obtain an average. Each time that $\mathcal{A}(x)$ is invoked, it is with an independent draw for the noise layer. This allows a Monte Carlo estimation of the expected value for the prediction.

The Monte-Carlo method refers to algorithms that estimate a value based on sampling and simulation. Given an instance X of a function problem, let $f(X) \in \mathbb{R}$ denote the desired output. In our example, $X/(\mathcal{A}(x))$ is the output of the model for an input x and $f(X)$ the average SoftMax value for the correct label. The average $f(X)$ received for multiple calls of an algorithm for an output X can be bounded using Chernoff bounds. Let X be the output of the algorithm and letting X^* be the actual average SoftMax. Note that $\mathbb{E}[X] = X^*$ and that by the Chernoff bounds for $\epsilon \in (0,1)$ [16].

$$P(|X - X^*| \geq \epsilon X^*) = P(m|X - X^*| \geq \epsilon m X^*) \leq 2e^{\frac{\epsilon^2 m X^*}{3}} \tag{2.8}$$

Thus, a bound for how far away the expectation is from the actual expectation is can be reached. Note that probability that the expectation lies outside these bounds, can be made arbitrarily small depending on the number of invocations m of the algorithm $\mathcal{A}$ [16].

Once the $\mathbb{E}[X]$, is calculated using this Monte-Carlo method, new tighter η confidence intervals are also calculated as in [47] that hold with probability η. These η confidence bounds values can also be made arbitrarily small by increasing the number of invocations of the prediction algorithm. Within this work, the η-confidence error bounds are calculated using the Hoeffding inequality. Using the Hoeffding inequality, with probability η the following holds:

$$\mathbb{E}(A(x))^{lb} = \hat{\mathbb{E}}(A(x)) - \sqrt{\frac{1}{2n}\frac{2k}{1-\eta}} \leq \mathbb{E}(A(x)) \leq \hat{\mathbb{E}}(A(x)) + \sqrt{\frac{1}{2n}\frac{2k}{1-\eta}} = \mathbb{E}(A(x))^{ub} \tag{2.9}$$

where n is the number of invocations of the algorithm A, and k is the index of the label (starting from 1) [16], [47].

We can now give the general robustness condition for full prediction algorithm taking into account our use of a Monte-Carlo simulation for getting the value of $\mathbb{E}[\mathcal{A}]$

General Robustness condition *Suppose a randomized function A satisfies (ϵ,δ)-feature differential privacy with respect to changes of size L in p-norm metric. Letting* $\mathbb{E}^{ub}(A_i(x)$ *and* $\mathbb{E}^{lb}(A_i(x)$ *be the* η *upper and lower bounds for a Mote-Carlo estimate, then for any input x, if for some* $k \in K$ [47]

$$\mathbb{E}^{lb}(A_k(x)) > e^{2\epsilon} max_{i:i\neq k}\mathbb{E}^{ub}(A_i(x)) + (1+e^{\epsilon})\delta \tag{2.10}$$

The proof for this equation can be found in Lecuyer et al. [47]. Then in order to determine the robustness, the following algorithm is used. Note that as previously stated the Laplace and Gaussian mechanisms have noise standard deviations σ that grow in $\frac{\Delta_{p,q}L}{z\epsilon}$. For a given σ used at prediction time, we then solve for the maximum $L_{x_{max}}$ for which the robustness condition holds. This is such that, according the general robustness condition [47]:

$$L_{x_{max}} = max_{L\in\mathbb{R}}\, L\ s.t.$$

$$\mathrm{E}^{lb}(A_k(x)) > e^{2\epsilon} max_{i:i\neq k}\mathbb{E}^{ub}(A_i(x)) + (1+e^{\epsilon})\delta\, AND\, either$$

- $\sigma = \Delta_{p,1}/\epsilon\, and\, \delta = 0\, (for\, Laplace)\, OR$
- $\sigma\sqrt{2ln(\frac{1.25}{\delta}}\Delta_{p,2}L/\epsilon\, and\, \epsilon \leq 1\, (for\, Gaussian)(2.11)$

Thus, if the robustness value is greater that the predefined value L, then we say that given datapoint example x is robust to adversarial examples of size $_{attack}$.

Robustness Certificate

In addition to being able to withstand attacks of up to a given size $_{attack}$ (for the predefined fixed $_{attack}$ for which the model is trained) the returned *robustness certificate* for each tested datapoint can actually has a more relaxed interpretation. The returned *robustness certificate* is actually the maximum attack size for each input, against which that particular datapoint is robust. The *robustness size certificate* incorporates the size of possible adversarial attack L_{attack} as well as the η-confidence upper and lower bounds to create a stability bound. This bounds the change an adversary could make on the average score of a given label with a p-norm attack. This η value is thus also used to construct a η-confidence upper bound and lower bound on the change an adversary can make with a p-norm input change of up to L_{attack} [47]. With this *robustness certificate*, if the lower bound of the label with the largest score is strictly greater than that of every other label, then the input x is *robust* to arbitrary p-norm attacks of size L_{attack}.

This then is basically the calculated L_{max} from the robustness calculation. This can be seen most clearly in Fig. 2.5. Thus, every correct prediction has a given robustness bound and each one can be different. For example, for a model trained to be robust to attacks of size $L = 0.1$, some of the tested datapoints will have robustness certificates of size 0.11 and others will have certificates of size 0.8. Furthermore, the failure probability of this *robustness certificate* can be as small as one wishes, by invoking more calls of the prediction algorithm. Despite the probabilistic nature of stability bounds, note that the differential privacy aspect of this procedure is not probabilistic. Rather, this algorithm

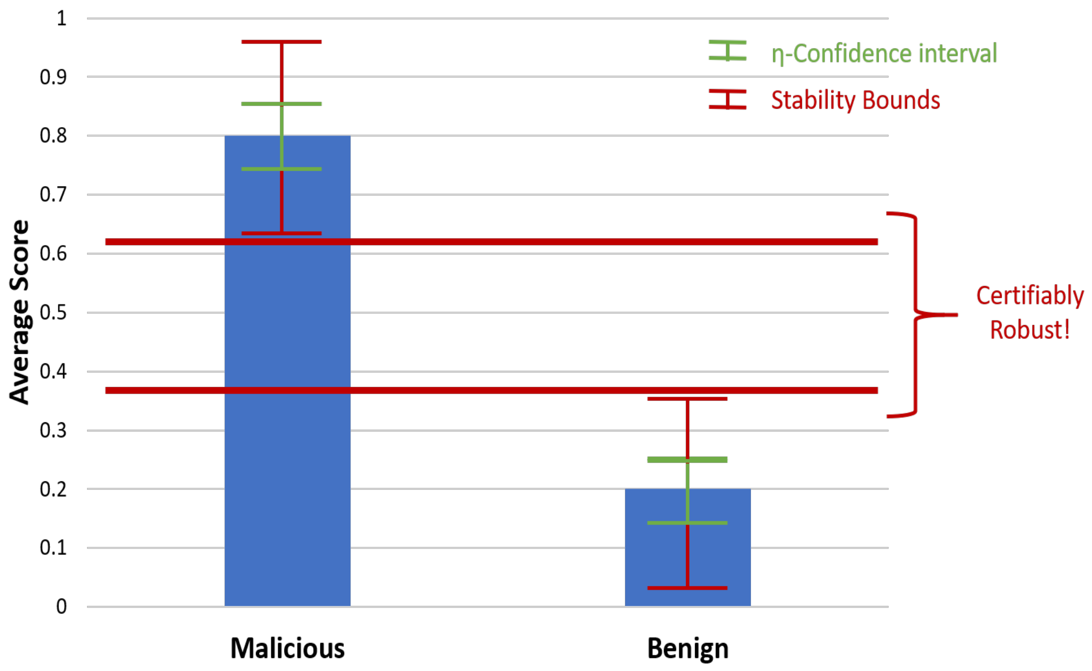


Figure 2.5: Example of how robustness is determined with a malicious flow example. Here the malicious flow is certifiably robust due to the gap that exists between possible probability labels for the benign label and the malicious label.

is probabilistic due to the Monte-Carlo simulation that must be conducted in order to estimate the $\mathbb{E}[\mathcal{A}(x)]$.

2.5 Data Analysis Algorithms

Within this work, we used several data analysis algorithms in order to examine our datasets. We give brief overviews of these algorithms here.

2.5.1 K-means clustering

K-means clustering is a partition-based clustering algorithm that assigns points x to cluster in a D-dimensional Euclidean space.

Within the k-means formulation, k 'means', the algorithm assigns points to $\boldsymbol{\mu_1}, ...\boldsymbol{\mu_k}$ where each μ_i represents a given cluster. In the k-means, the goal is to minimize the following objective over all possible partitions $C_1, .., C_k$ [11].

$$W(C_1, ...C_k; \boldsymbol{\mu_1}, ...\boldsymbol{\mu_k}) = \Sigma_{j=1}^{k} \Sigma_{i \in C_j} ||\mathbf{x_i} - \boldsymbol{\mu_j}||_2^2$$

This problem is NP-Hard, but a greedy algorithm to optimize this objective was developed.

Algorithm 2 Lloyd's K-means clustering

```
1: Initialize $\boldsymbol{\mu_1}, ..., \boldsymbol{\mu_k}$ (centres of clusters)
2: **repeat**
3:    **for** j=1,...,k **do**
4:       $C_j \leftarrow$ { i| j = arg  $\min_{j'} ||\mathbf{x_i} - \boldsymbol{\mu_j}'||_\mathbf{2}^\mathbf{2}$}
5:    **end for**
6:    **for** j=1,...,k **do**
7:       $\boldsymbol{\mu_j} \leftarrow 1 \frac{}{C_j \Sigma_{i \in C_j} \mathbf{x_i}}$
8:    **end for**
9: **until** convergence
```

2.5.2 Principal Component Analysis

Principal component analysis (PCA) is a dimensionality reduction technique that aims to give succinct representation of data whilst minimizing information loss. PCA allows the uncovering patterns in data and leads to informative representations of data.

PCA works by first translating data vectors to be mean cantered. PCA then finds an orthonormal family of k vectors that 'explain most of the variation of the data'. Specifically, for each data point, PCA approximates each data point x_i by a linear expression with each orthonormal vectors. The optimal orthonormal vectors are the k principal component. The first vector u_1 is the direction of the greatest variance, u_2 is then the direction of greatest variance that is orthogonal to u_1, etc. More specifically, in order to find the principal component, singular value decomposition is applied to the dataset. For more details, on how PCA in calculated see [44]

2.5.3 t-Distribution Stochastic Neighbour Embedding

t-Distribution Stochastic Neighbour Embedding is another dimensionality reduction technique that is well suited for high-dimensional datasets. t-SNE models each high-dimensional data point by either two- or three-dimensional point. The object of the t-SNE algorithm is to model similar datapoints as nearby points and dissimilar datapoints as distant with high probability [50].

The t-SNE algorithm has two main phases. t-SNE first calculates a probability distribution over pairs of datapoints such that similar datapoints have a high probability of being picked from the distribution while the dissimilar points have a low probability of being picked from the distribution.

$$p_{j|i} = \frac{exp(-||\mathbf{x_i} - \mathbf{x_j}||^2/(2\sigma_i^2))}{\Sigma_{k\neq i} exp(-||\mathbf{x_i} - \mathbf{x_k}||^2/(2\sigma_i^2))} \tag{2.12}$$

The similarity of datapoint $\mathbf{x_j}$ to datapoint $\mathbf{x_i}$ is the conditional probability $p_{j|i}$, that $\mathbf{x_i}$ would pick $\mathbf{x_j}$ as its neighbour, if neighbours were picked in proportion to their probability density under a Gaussian cantered at $\mathbf{x_i}$ [50].

$$p_{ij} = \tfrac{p_{j|i} + p_{i|j}}{2N}$$

The bandwidth σ_i is chosen in such a way that the perplexity of these conditional distributions equals a predefined chosen perplexity. As a result, the bandwidth has the property the smaller values are used in denser parts of the data space.

After this stage, t-SNE then defines a similar probability distribution over the same points in a lower dimensional map. It then attempts to the minimize the Kullback-Leibler divergence between the distributions [50]. We will not go into these details here. For more information about t-SNE see [50].

Chapter 3

Attack Scenario, Adversary Model, and Proposed Defence

In this chapter, we give an overview of the adversary model that we considered within this work. Specifically, we look at the goals of a potentially adversary and their motivations. We also look at the goals of the attacker's victim or the *defender* and how they might go about preventing attacks.

3.1 Scenario

We first outline the specific scenario that we try to protect against. In this scenario, an institution/system has a neural network to detect malicious flows that have entered its internal network. After designing and implementing this neural network by training it on a series of example malicious and benign examples, the network is able to efficiently and effectively differentiate malicious from benign flows. However, because of their use of a neural network, this detection system is vulnerable to adversarial examples. These adversarial examples are slight perturbations in training images/testing flows that cause them to be misclassified. Within this setting slight perturbations are defined as changes that do not require a large change within the malware such that it becomes non-functional. A potential adversary thus attempts to design a new malware that the neural network is supposed to detect, but changes it in key ways, in order to allow this new malware to evade detection.

3.2 Adversary Capabilities

Before giving details on how we designed our experiments and what parameters that we used, we first give a formal definition of the adversarial model that was used within our work. In order to perform this attack, the adversary could have several different types of capabilities. In the first set of capabilities, the adversary could be in a **black-box** setting. Here the adversary would only be able to check whether his designed malware was actually detected by the institution's malware detector. Besides this, the adversary would not know any additional information.

In addition to the black-box setting, the adversary could gain access to the malware detector's weights and architecture. After discovering the weights, the adversary could a design a particular malware that would be able to evade detection based on his knowledge of the detector. This type of scenario is known as a **white-box** setting.

We assume in both of the above settings that an adversary has access to a reasonable amount of computing power (i.e. of an individual or of a small corporation) and has a reasonable amount of time to formulate her attack.

As shown in Section 2.2.5, adversarial examples are always keen to exist within neural networks. We will show in the rest of this work that in both the white-box and the black-box settings an adversary can easily design and implement malware by taking advantage of the adversarial example weakness of neural networks. We take this step further by also showing that targeted attacks on given features can also create adversarial examples against neural networks (i.e. by only raising the number of packets forwards, one can fool a network as well). These are also considered **white-box** setting attacks.

3.2.1 Adversary Goals

An adversary within both of these setting can have two different goals. The first goal of an adversary is the most obvious: the adversary could take a specific malware and then change it slightly in order to evade detection by the malware detector. The second goal of the adversary is less obvious. For this goal, the adversary takes many benign flows and changes them slightly in order to make them appear malicious. The adversary could then do this on a large amount of flows. In this way the adversary could potentially overwhelm the triage system of the network and make it nearly impossible for the institution using the network to identify truly malicious flows from within the malware labelled flows. Either of these types of attacks could be potentially disastrous.

3.3 Defence

Now that we have outlined the adversary's capabilities, we now outline the **purpose of this work** and the defence an institution could potentially implement do to prevent these types of attacks. This is namely by adding differential privacy to the training and prediction process, the institution can create stability bounds for a given prediction of whether a flow is benign or malicious. By creating stability bounds about which a flow is certified to be either benign or malicious, the attacked institution we imagine could do the following. (1) When a flow is certified as benign with a high probability for a given attack range, the institution can discard the flow information and move onto other flows. (2) When a flow is certified as malicious with high probability for a given attack range, the institution can vigorously investigate the flow to determine what malware has infected the system. (3) When a flow is labelled benign or malicious but only is certifiably robust to only a small perturbation based on the certificate returned, this can be used to prioritize the flow for additional screening. In this way the certificate returned acts as a means of triage for investigating different types of flows. Note that this does not have to do with the probabilities (i.e. 99% malicious 1% benign) returned for flow as in other confidence system. Rather this has to do with the robustness of the flow to small changes that could potentially change the classification. (4) When a flow is returned with a certificate, have knowledge of exactly what bounds about which the flow would have to be changed in order to switch the classification. In this way, a network operator would know for example, that in order to make this particular malicious flow benign (i.e. the attacker would have had to at least raise the number of bytes per second by 10,000).

Chapter 4

Methodology

In this chapter, we first give overview of several tools used in this project. Second, we discuss the datasets used within our work.

This project was developed mainly in Python 3.6 with the TensorFlow library [23]. All models were constructed without Keras. In addition, to these tools, four different tools were used heavily within this project: Google Colab [9], Cisco Joy [2], CleverHans [55], and SMOTE [33]. Google Colab is on online environment in which much of the code for this project was written. For the other three tools, we will give an overview of their use in the upcoming sections.

The datasets used for this work were UNSW-NB15 [53], USTC-TFC016 [75], and CSE-CIC-IDS2018 [4]. We will give a brief overview of the features within these datasets that we used as well as some data analysis that we performed on each of them.

4.1 Cisco Joy

Cisco joy is a software package developed by Cisco that extracts data features from live network traffic as well as pcap files and stores them in JSON format. The data that it is able to extract is similar to that of Neflow [2]. However, in addition to being able to process live network flows and process packets, Joy allows the exploration of data at scale. The information that can be gleaned from joy includes HTTP, DNS, TLS, SNI data as well as the following:

1. the sequence lengths and arrival times of IP packets
2. the probability distribution of bytes within the data portion of a flow and their entropy
3. the sequence of lengths and arrival times of TLS records
4. non-encrypted TLS data, such as the list of offered cipher-suites
5. DNS names, addresses, and TLLs
6. HTTP header elements
7. the name of processes associated with the flow
8. bidirectional flow information including packet timings, packet sizes, and the number of bytes in and out

Within our work, Cisco Joy was used in order to get bidirectional flow information. We specifically used the following command to process USTC dataset flows.

```
bin/joy bidir=1 ppi=1 http=1 tls=1 dns=1 output=cridex.json cridex.pcap
```

This command allows joy to process flows, group them as bidirectional, extract packet level information and to extract http, tls, and dns information. After joy processing, a subset of the statistics (a subset of the statistics extracted in Malalert [59] on a set number of features were calculated. These statistics included the maximum, the sum, the minimum, the average, the standard deviation, and the variance.

4.2 CleverHans

For the creation of adversarial examples, the open source library CleverHans was used. CleverHans was developed and is maintained by Ian Goodfellow and Nicolas Papernot [55]. CleverHans, more precisely, is a TensorFlow library that implements several different popular adversarial attacks. Taking in either a Keras model or a tensor that returns a probability function or the logits, CleverHans is able to construct and perform several different types of adversarial attacks including all of those described in the Background chapter. We used CleverHans in order to create adversarial examples using the sparse l_1 descent attack, fast-gradient sign method attack, and the HopSkipJump attack specifically.

Note that we used the version of CleverHans released as of June 2019. We noticed near the end of this project in August of 2019 that some of the functions within CleverHans were changed slightly. To replicate our results, please use the most updated version of CleverHans as of June 2019.

4.3 SMOTE: Synthetic Minority Over-sampling Technique

Within our work, both the USTC and UNSW datasets were heavily imbalanced. This imbalance mirrors the imbalance between benign and malicious traffic in real world settings. Most traffic that is sent is benign; it is only rare malicious traffic that needs to be detected and stopped. However, misclassifying these malicious flows comes at a high cost.

In training machine learning models, often the larger majority class can overwhelm the minority class, causing the model to only return one type of label. Basic under sampling of the benign flows and oversampling of the malicious flows is thus often used when training models. This of course can lead to somewhat skewed results.

Instead of simply oversampling the minority class (malicious flows in our work), we employed SMOTE. SMOTE create synthetic minority class examples in order to better train the model. SMOTE works by taking each minority class sample and introducing synthetic examples along the 'line segments joining any/all of the k minority class nearest neighbours' [33]. These k nearest neighbours are randomly chosen. More specifically, synthetic examples are constructed by:

Algorithm 3 SMOTE

```
1: Take the difference between the feature vector and its nearest neighbour
2: Multiply the difference by a random number between 0 and 1 and add it to the
   feature vector
3: Add the new feature vector to the list of features X
4: Repeat until the number of minority examples equals the number of majority ex-
   amples
```

4.4 UNSW-NB15

The UNSW-NB15 dataset was developed by the Cyber Range Lab of the Australian Centre for Cybersecurity (ACCS) in order to update the contemporaneous set of datasets with one that included real normal malicious activities and synthetic attack behaviours [53]. UNSW-NB15 specifically sought to counteract the unavailability of standard network benchmark data set challenges. Older datasets like KDD98, KDD-CUP99, and KSLKDD, generated over a decade ago, at the time did not reflect modern network traffic and had relatively low bars for detecting malicious traffic. UNSW-NB15 updates these datasets and incorporates both real and synthesized modern network attack traffic. In addition to providing the raw traffic, UNSW-NB15 provides a list of generated features from the gleaned bidirectional flows.

Table 4.1: UNSW Dataset Flow Distribution

Type	**Number of Records**	**UNSW Description**
Benign	2,218,761	Normal transaction data
Fuzzers	24,346	Attempts to cause program failure by feeding random data
Analysis	2,677	Port scans, spam, and html file penetrations
Backdoors	2,329	A system security mechanism is bypassed to access computer data
DoS	16,253	Attempt to make a server unavailable to users, by temporarily interrupting or suspending the services of a host
Exploits	44,525	Attacker knows a security issue in an OS or piece of software and leverages vulnerability
Generic	215,481	Generic attack that works against block-ciphers of a given block and key size, without consideration of the structure of the block-cipher
Reconnaissance	215,481	Generic attack that works against block-ciphers of a given block and key size, without consideration of the structure of the block-cipher
Shellcode	1,511	Code used as a payload in exploitation of a vulnerability
Worm	174	Code replicates itself to spread.

The UNSW-NB15 data set was created using the IXIA Perfect Storm tool in a simulated environment over the course of 31 hours. The IXIA tools downloads information about new attacks and is continually updated from a CVE site (https://cve.mitre.org/). This particular site contains information about all publicly known security vulnerabilities and attacks. In order to capture the network traffic, the tcpdump tool was used. The dataset itself contains nine different families of attack in addition to benign traffic. Specifically, it contains: Backdoors, Fuzzers, Analysis attacks, DoS attacks, Exploits, Generic Block-cipher attacks, Shellcode and Worms. See Table 4.1 for details about the types of attacks.

From this dataset, 47 different flow statistics were gathered about the flow. We a subset (due to space constraints) of these flows and their indices in Table 4.2. These features were extracted using the Argus and Bro-IDs toolsets [53].

Table 4.2: UNSW Data Set Feature List 2

Number	Name	Description
		Content Features
19	swin	Source TCP window advertisement
20	dwin	Destination TCP window advertisement
21	stcpb	Source sequence number
22	dtcpb	Destination sequence number
23	smeansz	Mean of the flow packet size transmitted by the src
24	dmeansz	Mean of the flow packet size transmitted by the dst
25	trans-depth	the depth into the connection of the http request/response
26	res-bdy-len	The content size of the data transferred from the server's http service
		Time Features
27	sjit	Source jitter (mSec)
28	djit	Destination jitter (mSec)
29	stime	Flow Start time
30	ltime	Flow End time
31	sinpkt	Source inter-packet arrival time (mSec)
32	dinpkt	Destination inter-packet arrival time (mSec)
33	tcprtt	The sum of 'synack' and 'ackdat' of the TCP
34	synack	The time between the SYN and the SYN-ACK packets of the TCP
35	ackdat	The time between the SYN-ACK and the ACK packets of the TCP

4.4.1 UNSW Data Analysis

For the purposes of this work, we only considered HTTP service flows for consistency. Namely, in this work, we desired for the flows considered to be of the same type within the network so that the differentiation between DNS and HTTP behaviour patterns would not complicate later parts of this work. In any case, we now present the details of subset that we used within this work. As seen in this work, there were no shellcode

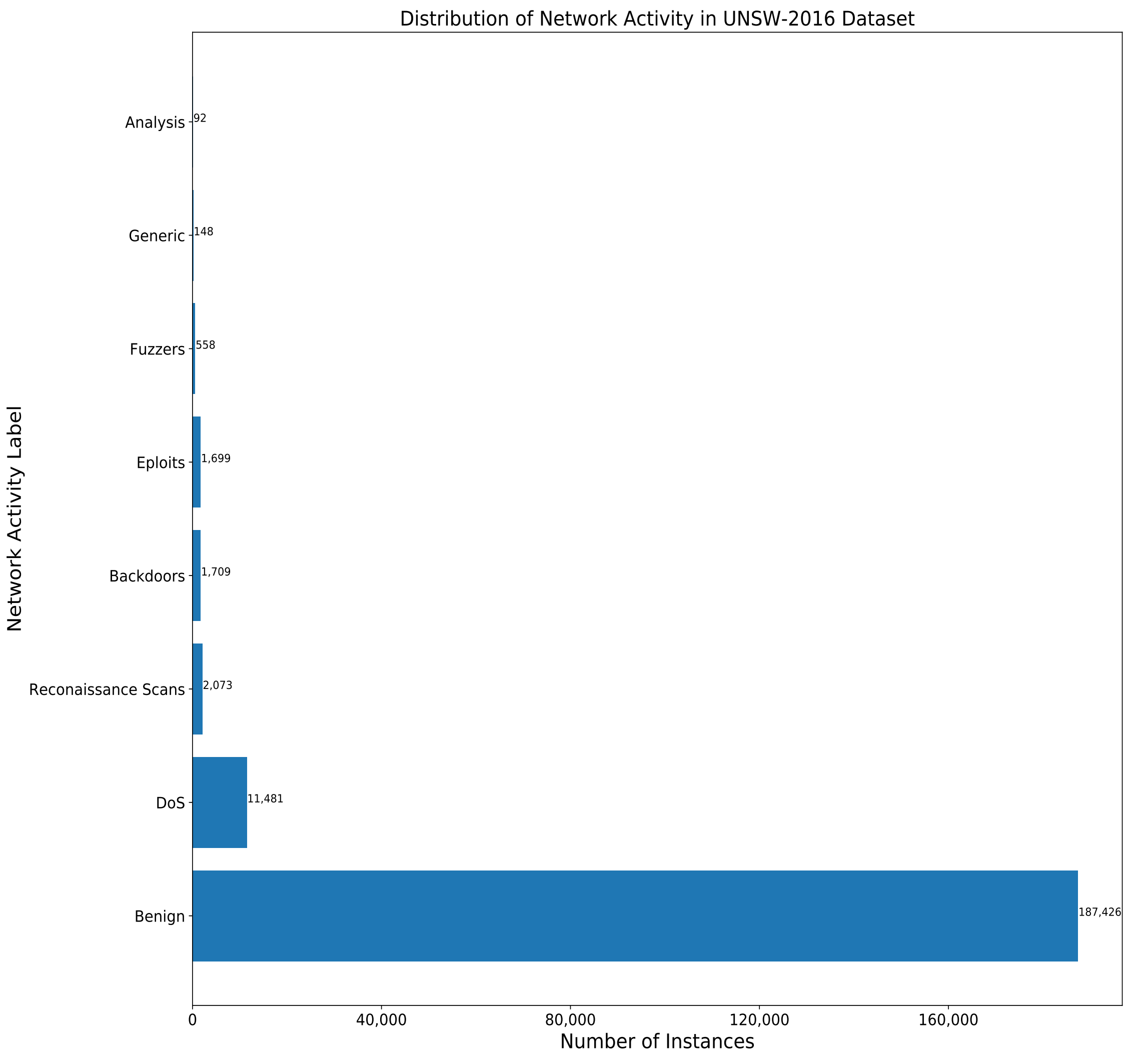


Figure 4.1: Data distribution of UNSW-NB15 HTTP flows considered in this work

attack flows under the http service (this is because it uses the ssh service). As a result, we do consider this type of attack within the rest of this work.

As seen in Fig.4.1, there is a significant difference in the amount of data between the malicious and benign flows. Furthermore, even amongst the malicious flows, there is much variance in the number of flows. As a result, in order to combat the imbalance within the dataset, we performed up sampling via SMOTE on clusters of each type of flow.

In order to get the clusters for up sampling each type of malicious flow, we first performed K-means clustering with the numerical features within UNSW dataset to identify the number of natural clusters within each type of malicious flow.

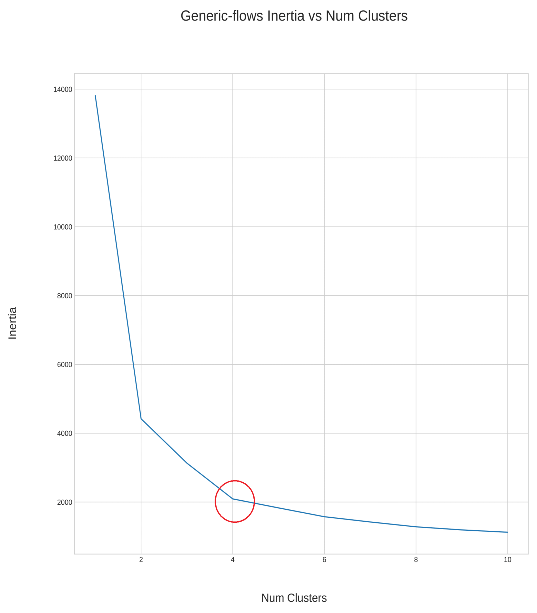


Figure 4.2: Clustering inertia vs. the number of clusters for Generic malicious flows.

Using the k-means algorithms we found the elbow of where the decrease in the inertia/mean squared error of clustering began to level off. Here we placed the number of clusters. This can be seen most clearly in Fig. 4.2. For example, for the Generic malicious flows, the number of clusters was 4. After performing this clustering, we performed resampling for each cluster to get at most 5,000 flows for each cluster. In this way we managed to achieve a balanced set.

After performing the clustering, we also wanted to ensure that we were obtaining useful information from each numerical feature in the dataset. For this reason, we conducted feature correlation in order to ascertain how much each feature could be predicted by the others. In this way we could determine, whether a feature was actually useful for machine learning.

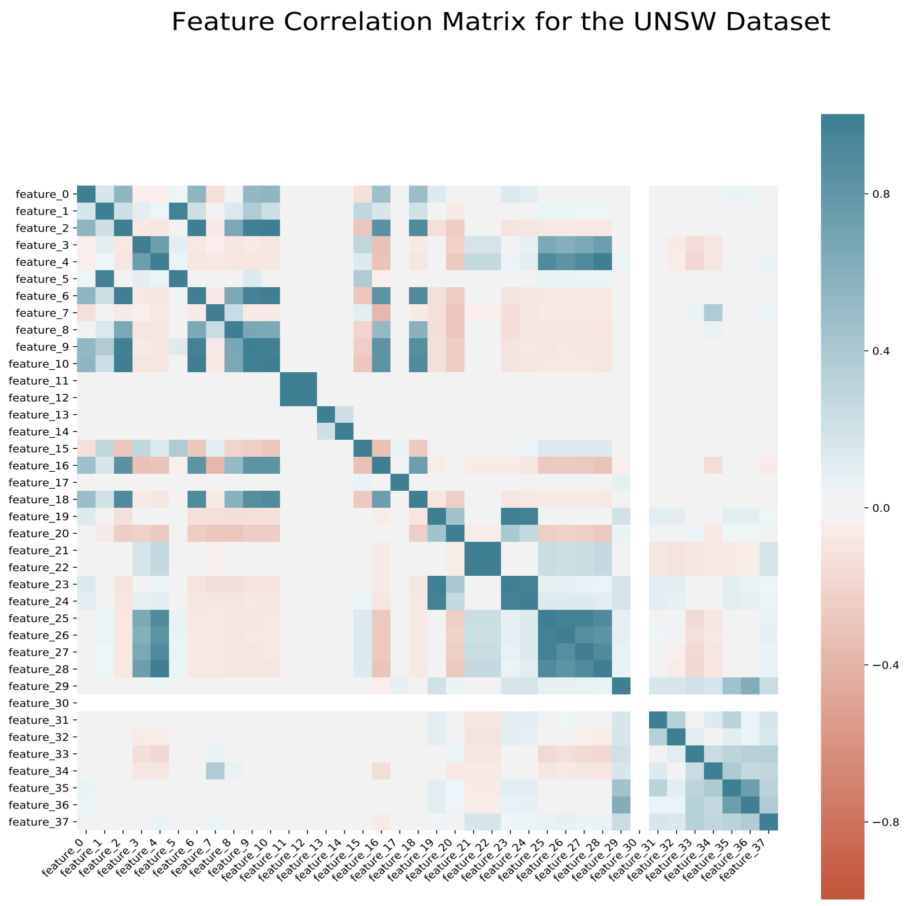


Figure 4.3: Numerical Feature Correlation Matrix for the UNSW dataset.

As seen in Fig. 4.3, a lot of the features are generally well-correlated with each other. However, despite this in most cases this is a fairly weak correlation. There is a very strong correlation amongst the feature 25-28. These are the features that contain the timing data ’tcprtt’, ’synack’, ’ackdat’ and the binary flag ’is-sm-ips-ports’. This correlation makes sense because ’tcprtt’ is the sum of ’synack’ and ’ackdat’. Furthermore, it makes sense that this timing data is correlated to ’is-sm-ips-ports’ because this timing data would be short or long if the client equalled the server to which it was trying to connect. For this paper, we decided to keep these respective features in order to better conduct finer-grained analysis of different types of malware. However, for future work (see Future Work) it would be interesting to observe how different feature subsets would affect our experiment.

In addition to performing the clustering and feature analysis, we also performed PCA and TSNE analysis on this dataset. We performed this initial analysis of the datasets, in order to (on the first order) gain an understanding of the separation between malicious and benign flows. Furthermore, we further performed the TSNE analysis to see potential differences between the different types of malicious flows.

PCA analysis

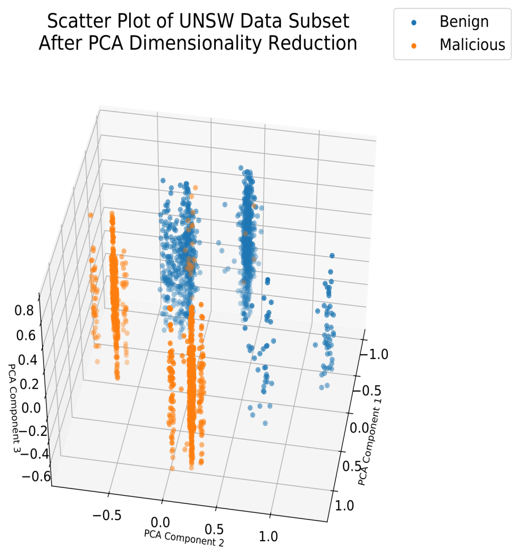


Figure 4.4: Principal component analysis of the UNSW dataset: Benign vs. Malicious.

We plotted the 3D representation of UNSW dataset as seen in Fig. 4.4. As seen in principal component analysis of the UNSW dataset, there is a clear division between the malicious and benign flows. While there are some areas where they overlap, this initial analysis, indicated that that there was a clear means of separating out most of the malicious activity within the dataset from the benign.

t-SNE Analysis

We now present 2-D and 3-D t-SNE representations of the UNSW dataset.

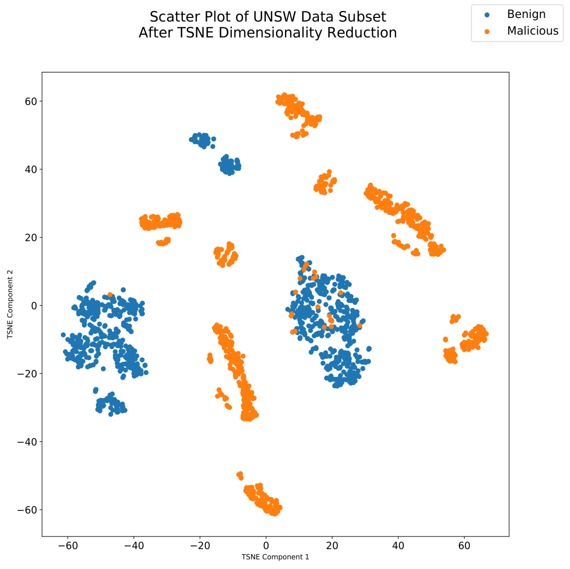


Figure 4.5: 2D t-SNE of the UNSW dataset: Benign vs. Malicious.

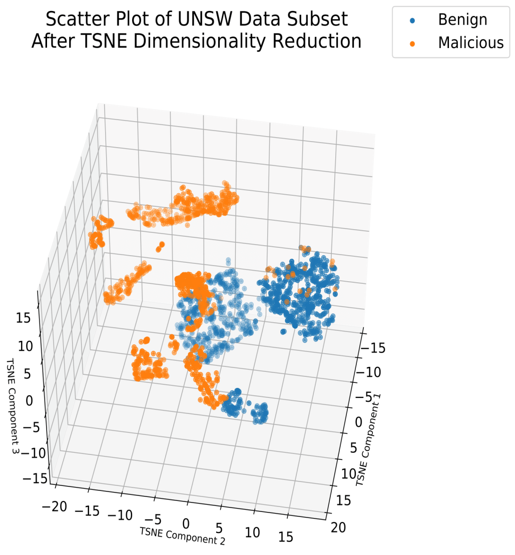


Figure 4.6: 3D t-SNE of the UNSW dataset: Benign vs. Malicious.

As can be seen in Figs. 4.5 and 4.6, there is an even more clear separation of the benign and malicious data. Although this is only a subset of the dataset (for visualization purposes), this further reinforces that this dataset is separable into benign and malicious flows.

Finally, we present a t-SNE plot for only the malicious flows. Here we look to see if there were clear defining differences between the different types of malicious flows. Here we see fewer dividing lines between the types of flows. Although there are certain

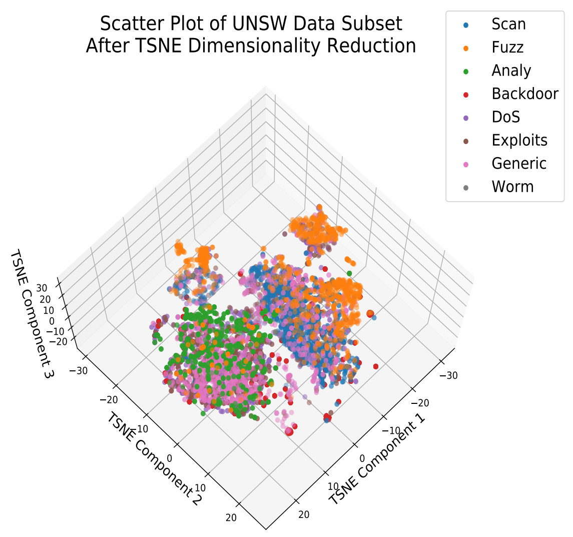


Figure 4.7: 3D t-SNE of the UNSW dataset: Different types of Malicious Flows.

lines that are clear, i.e. between Scan and Analysis flows, overall, there are not simple lines that can be draw that would separate the different types of flows. Despite this in Fig. 4.7, there is 'pancaking' of some of the types of flows along the z-axis suggesting another means of differentiating the types of flows.

This concludes our discussion of the UNSW-NB15 dataset.

4.5 USTC-TFC2016

Here we will discuss the details of the USTC-TF2016 dataset. The USTC-TF2016 dataset is separated into two different types of traffic. In addition to having ten different types of malware traffic from public websites that were collected by the CTU University of Prague Stratosphere Lab from 2011-2015 (see Table 4.3, the dataset includes normal traffic collected using the IXIA tool [75]. The normal traffic specifically came from eight different types of sources: BitTorrent, Facetime, FTP, Gmail, MySQL, Outlook Email, Skype, SMB, Weibo, and World of Warcraft.

Table 4.3: USTC Malware Traffic

Name	**CTU-Name**	**Binary MD5**
Cridex	108-1	25b8631afeea279ac00b2da70fffe18a
Geodo	119-2	306573e52008779a0801a25fafb18101
HtBot	110-1	e515267ba19417974a63b51e4f7dd9e9
Miuref	127-1	a41d395286deb113e17bd3f4b69ec182
Neris	423-3	bf08e6b02e00d2bc6dd493e93e69872f
Nsis-ay	53	eaf85db9898d3c9101fd5fcfa4ac80e4
Shifu	142-1	b9bc3f1b2aace824482c10ffa422f78b
Tinba	150-1	e9718e38e35ca31c6bc0281cb4ecfae8
Virut	54	85f9a5247afbe51e64794193f1dd72eb
Zeus	116-2	8df6603d7cbc2fd5862b14377582d46

4.5.1 USTC Malware Types

Here we give a brief overview of the types of malware that were within the USTC dataset.

Cridex

Cridex is a malware that incorporates the infected computer into a botnet. It also injects itself into a victim's web browser to steal information, specifically banking credentials. Cridex is further able to log keystrokes and capture screenshots.

Cridex is normally spread through emails with malicious attachments, but Cridex can also self-replicate through USB devices. Specifically, it executes through Microsoft macros when the attachments are opened by the user. In order to further execute, Cridex opens up a backdoor on the infected client and then downloads additional files before joining the botnet.

Cridex had a very wide distribution within the Western world, with most victims being in the United States, Japan, Germany, and the UK [3].

Geodo

Geodo/Emotet is a banking Trojan. Throughout 2018, Geodo malware was used to conduct financial theft throughout the world. Geodo is closely related to Cridex and Cridex's later iteration Dridex. Like Cridex, Geodo is normally spread through emails. However, unlike Cridex, Geodo sends malicious URL links within emails.

Once clicked on, geodo sends a HTTP Post request containing encrypted data to a list of command central IP addresses. Additional files are then sent back to the infected machine. Geodo is modular Trojan so most of its functionality is actually abstract away from its main code and is instead in downloaded files. After infecting a machine, Geodo attempts to gain additional credentials and (most important for its purpose) banking information.

Geodo continued to heavily infect many banking institutions within the UK in 2018 [7].

HtBot

HtBot is a malware that allows a remote attacker to gain access and send commands to a victim machine operated by the Windows OS. As the name implies, HtBot causes the infected computer to become a bot within a given network. HtBot was prevalent in 2013 [10].

Miuref

Miuref is a Trojan horse. Once a client is infected Miuref, Miuref downloads several malicious files. Miuref infected clients are often participants in click fraud. Miuref is known to spread via email attachments.

Specifically, Miuref infected computers usually attempt to connect to 176.9.245.16 address and download files such as BluetoothUtilperf.1, and BluetoothUtilperf.dll. and then engage in various forms of click fraud [13].

Neris

Neris is also a type of botnet sent via email. In particular Neris uses an HTTP-based communication channel in order to spam and DDoS victim websites [14].

Nsis-Downloader-Ay

Nsis (Nullsoft Scriptable Install System) is a Trojan botnet that specifically targets Windows platforms. After being infected, Nsis is often used to download and install additional malware. Like other types of malware, Nsis is spread via email [15].

Shifu

Shifu is a banking Trojan discovered in 2015 that targets Windows computers. Borrowing some of its central configurations from other successful banking Trojans like Zeus and Cridex, this Trojan was devastating for those that it infected. Specifically, Shifu targeted hosts by exploiting the Window vulnerability CVE-2015-0003 in order to gain system privileges.

Shifu was mostly active in banks in Japan and the UK [18].

Tiny Banker Trojan/Tinba

Tinba is a Trojan meant to target banking websites. Tinba works by establishing man-in-browser attacks. Specifically, Tinba uses packet sniffing in order to determine if an infected client is on a banking website. After confirming this, Tinba will form-grab the baking webpage before it is encrypted by HTTPS and then send user keystrokes to a command center.

Tinba was a major threat in 2012-2013 and infected many major institutions including Bank of America, Wells Fargo, and TD Bank [20].

Virut

Virut was a major botnet that operated from 2007-2012. Virut infected executable files as well as ASP, HTML, and PHP files. Virut has worm-like patterns (i.e. it spreads by copying itself to USB and network drives).

Furthermore, Virut is an entry-point obscuring (EPO) polymorphic file infecting virus. It is able to infect executable files by hooking system APIs. After infecting a given client, Virut creates a back door that allows the attacker to send commands to the compromised computer via the Internet Relay Chat.

Virut was prevalent in the United States, China, the UK, India, and Canada [21].

Zeus/Zbot

Zeus is a Trojan horse that steals information from the infected client. Zeus targets user credentials, and in particular system information and banking details. Zeus was built from the online Trojan-building toolkit, making its proliferation even more noteworthy. Zeus is primarily spread through email and drive-by downloads.

Zeus automatically gather any passwords that were stored on the client's browser while also monitoring user behaviour to glean more user credentials [22].

4.5.2 USTC Data Analysis

Unlike the UNSW dataset ready-made features were not available for USTC dataset. As a result, Joy was used in order to process these flows. After processing, statistics including the sum, maximum, minimum, mean, standard deviation, and variance were extracted for each feature from this data and for each bidirectional flow. This feature set is subset of feature set created in Malalert [59]. Note again for this work flows are considered all bidirectional packets that have the same client and server IPs and the same client and server ports.

1. The number of packets sent from the client to server
2. The number of packets sent from the server to the client
3. The number of total packets sent within the flow
4. The number of bytes sent from the client to the server
5. The number of bytes sent from the server to the client
6. The total number of bytes sent in the flow
7. The time intervals between packets sent from the client to the server

8. The time intervals sent from the server to the client
9. All time intervals in the flow
10. The flag counts for syn, ack, psh, and fin flags within packets from the client to server.
11. The flag counts for syn, ack, psh, and fin flags within packets from the server to the client.
12. The duration between bidirectional flows

After processing there were 158 different statistics collected. In addition to these flow statistics, we also collected the sequences of inter-arrival packet timings, packet sizes, and packet directions for each flow. This additional information was used in a different flow model. See the Results chapter for more details.

After processing, the following distribution of flows was observed.

Again, as seen in Fig.4.8, there is a significant difference in the amount of data between the malicious and benign flows. Again, as in the UNSW dataset evens amongst the malicious flows, there is much variance in the number of flows. Again, to combat the imbalance in the set, we performed up sampling on the malware data sets using SMOTE. Like in case of UNSW, we first performed k-means clustering before up sampling with SMOTE so that we could have equal amounts (5000) of data within each cluster.

As with the UNSW dataset, after performing the clustering, we also wanted to ensure that we were obtaining useful information from each numerical feature in the dataset. For this reason, we conducted feature correlation in order to ascertain how much each feature could be predicted by the others. This was especially important for our analysis given that the features that we designed were manually chosen by and were not a given feature of the dataset.

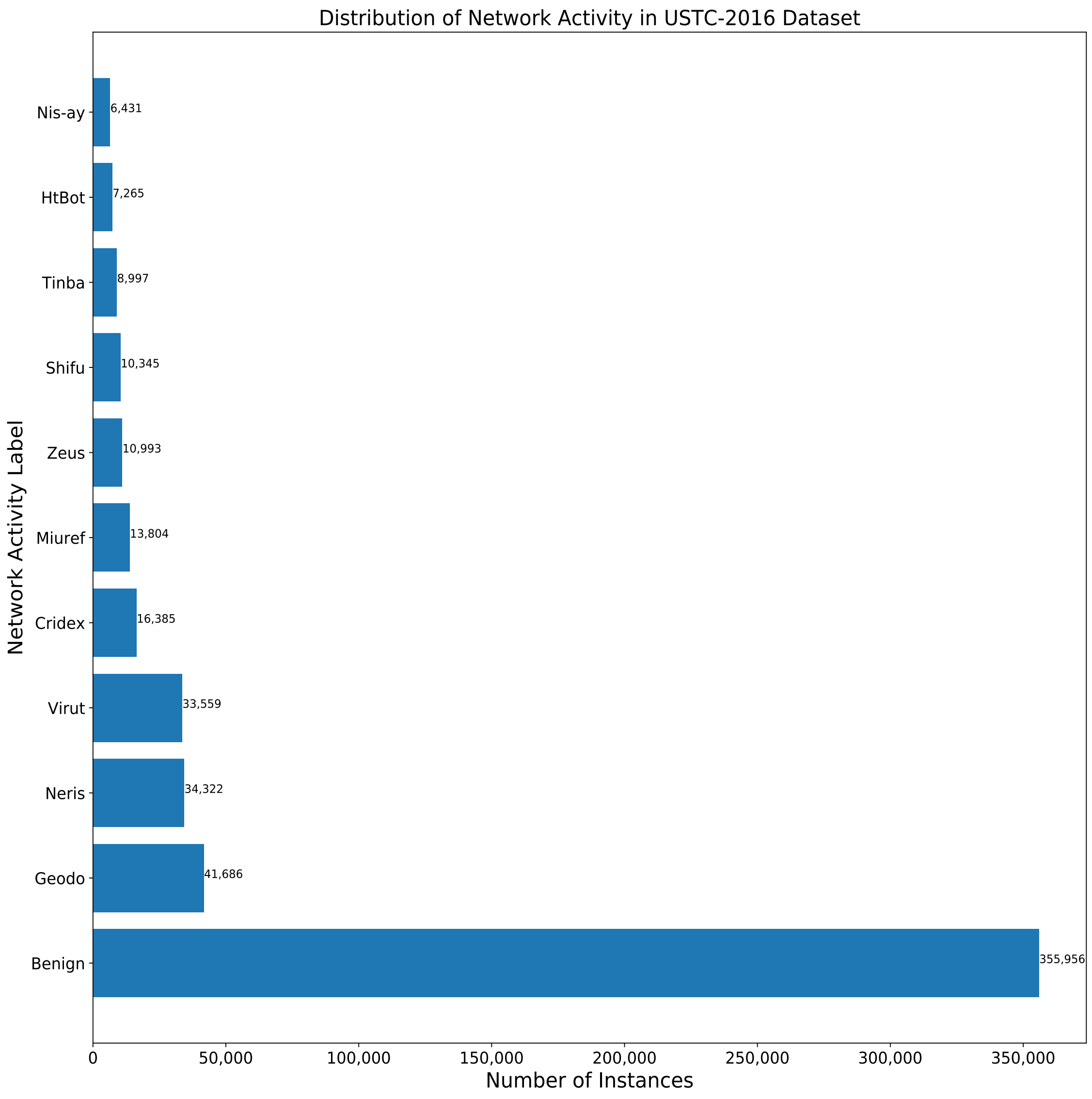


Figure 4.8: Distribution of USTC flows after Joy processing

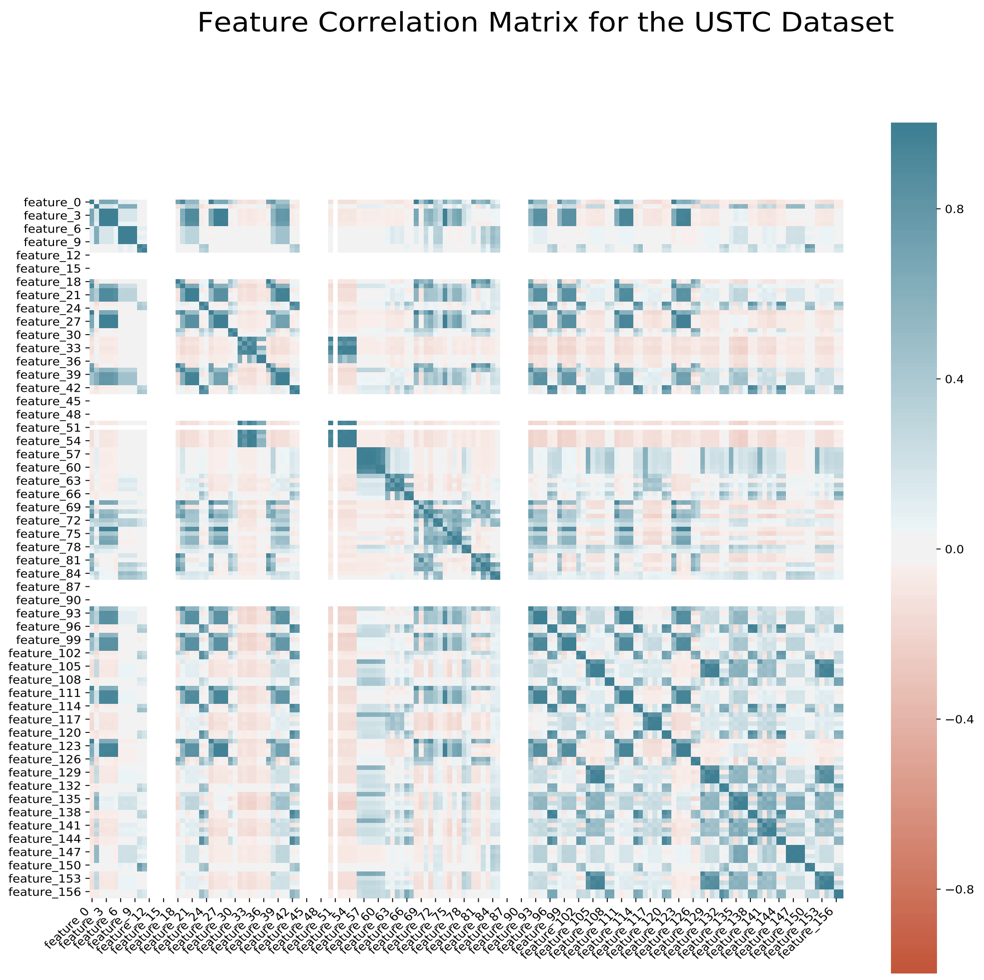


Figure 4.9: Feature Correlation for the USTC dataset

As expected, because we took five statistics for several features (i.e. number of bytes), we see groups high correlation in our feature set. Despite this, in several places we see areas of very low correlation or neutral correlation. Thus, as with UNSW dataset, we keep the features that we designed and leave feature selection and its effects on our experiments for future work (see Future Work chapter).

PCA analysis

Again, we do PCA analysis for the USTC dataset to ascertain the separability of the benign and the malicious flows using only the feature set.

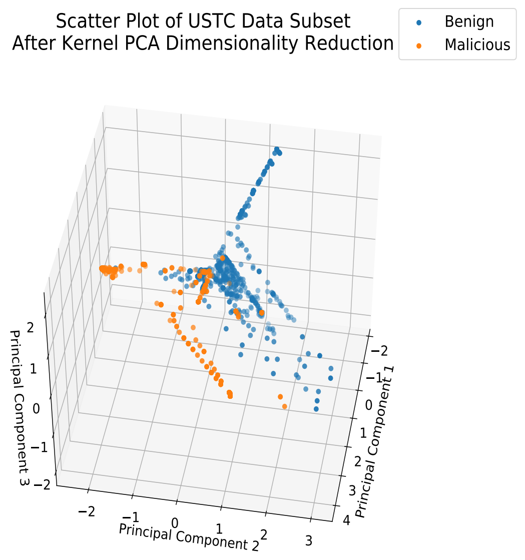


Figure 4.10: PCA Analysis of the USTC dataset: Benign vs Malicious

Again in Fig.4.10 we see a clear separation between the subsampled malicious and benign flows. This confirms that a neural network will be able to separate these two types of flows.

t-SNE analysis

As with the UNSW dataset we also performed t-SNE analysis for malicious vs benign flows in the USTC dataset. Figs.4.11 and 4.12 further confirms that there is a clear difference between the malicious and benign flows.

We now consider the separability of the USTC/CTU malicious flows into their different types using t-SNE analysis. As seen Fig. 4.13, there is more distinct separations between the different types of malware traffic using the features gleaned from Joy. There are clearer divisions between the different botnet/Trojans than that shown in the UNSW dataset. This completes the analysis of the USTC dataset.

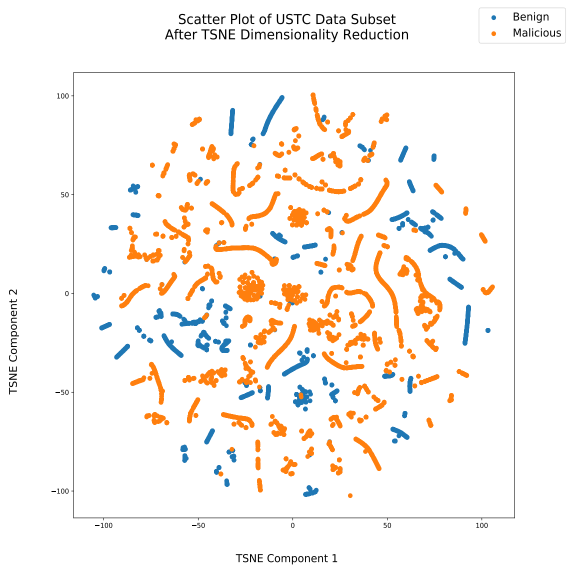


Figure 4.11: 2D t-SNE Analysis of the USTC dataset: Benign vs Malicious

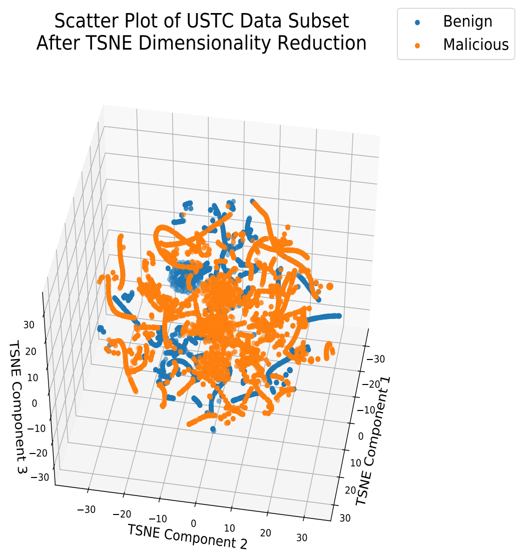


Figure 4.12: 3D t-SNE Analysis of the USTC dataset: Benign vs Malicious

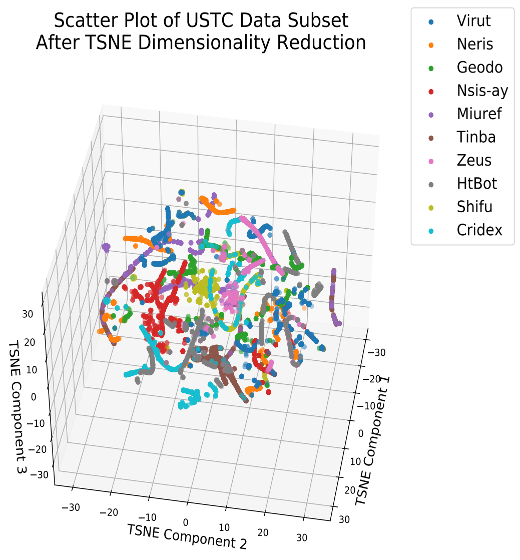


Figure 4.13: 3D t-SNE Analysis of the USTC dataset: Benign vs Malicious

4.6 CSE-CIC-IDS2018

Developed by the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC), the CSE-CIC-IDS2018 dataset is one the most up-to-date, comprehensive, and extensive network security datasets publicly available. This dataset generates a diverse set of user profiles that represent a variety of abstract network behaviours. This dataset further is updated almost annually with new attacks. The 2018 dataset includes seven different attack scenarios: Brute force attacks, the Heartbleed attack, Botnets, DoS attacks, DDoS attacks, Web-based attacks, and infiltration attacks from within the network. The adversarial architecture to perform these attacks includes over 50 machines. The defending organization itself is also massive; it includes 5 departments that contain 420 machines and 30 servers. The dataset includes network traffic and system logs of each machine as well as 76 statistical features extracted on bidirectional using CICFlowMeter-V3 [4], [6]. Due to the size of this dataset, for the purposes of this work, we consider a subset of this dataset, namely the botnet attacks from March 2, 2018. The Botnets used within the CSE-CIC-IDS2018 were both the Zeus and Ares botnet. For more details about the Zeus Trojan software see Section 4.5. The Ares botnet is an open source botnet with the ability for remote control, screenshotting, and key-logging [4]. These botnet software were used to take screenshots on victim machines every 400 second.

From this dataset, 76 different flow statistics were gathered about each flow. We give a subset(due to space constraint) of these features here in Table 4.4.

Table 4.4: CSE-CIC-IDS2018 Dataset Feature List 2

Number	Name	Description
		Flow Features
41	pkt-len-avg	Mean length of a flow
42	pkt-len-std	Standard deviation length of a flow
43	pkt-len-va	Minimum inter-arrival time of packet
44	fin-cnt	Number of packets with FIN
45	syn-cnt	Number of packets with SYN
46	rst-cnt	Number of packets with RST
47	pst-cnt	Number of packets with PUSH
48	ack-cnt	Number of packets with ACK
49	urg-cnt	Number of packets with URG
50	cwe-cnt	Number of packets with CWE
51	ece-cnt	Number of packets with ECE
52	down-up-ratio	Download and upload ratio
53	pkt-size-avg	Average size of packet
54	fw-seg-avg	Average size observed in the forward direction
55	bw-seg-avg	Average size observed in the backward direction
56	fw-byt-blk-avg	Average number of bytes bulk rate in the forward direction
57	fw-pkt-blk-avg	Average number of packets bulk rate in the forward direction
58	fw-blk-rate-avg	Average number of bulk rate in the forward direction
59	bw-byt-blk-avg	Average number of packets bulk rate in the backward direction
60	bw-blk-rate-avg	Average number of bulk rate in the backward direction

4.6.1 CSE-CIC Data Analysis

After processing and removing flows that were incomplete or contained nonsensical inputs (i.e. packets/s = inf), we got the distribution in Fig. 4.14.

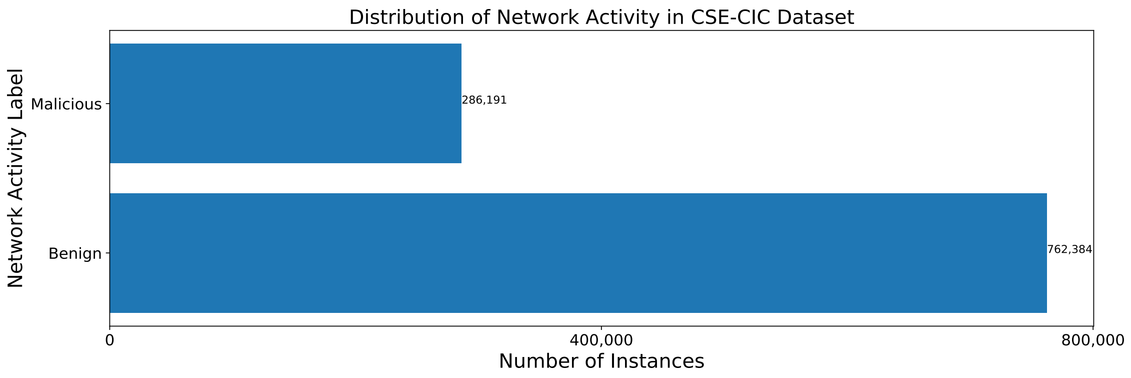


Figure 4.14: Distribution of CSE-CIC-IDS2018 flows

Unlike in the UNSW-NB15 and the USTC-TFC2016 datasets, we did not see a major disparity in number between the malicious and benign flows. As a result, for this dataset, we did not perform resampling.

As before we also wanted to ensure that we were obtaining useful information from each statistical feature in the dataset. For this reason, we conducted feature correlation in order to ascertain the usefulness of each feature.

Again, we see a great deal of heavy feature correlation between neighbouring features. This makes sense given that for several features the CSE-CIC-IDS2018 dataset takes multiple statistics. We do not seek to prune the number of features here but leave this to Future Work.

PCA analysis

Again, we do PCA analysis for the CSE-CIC-IDS2018 dataset to ascertain the separability of the benign and the malicious flows using only the feature set.

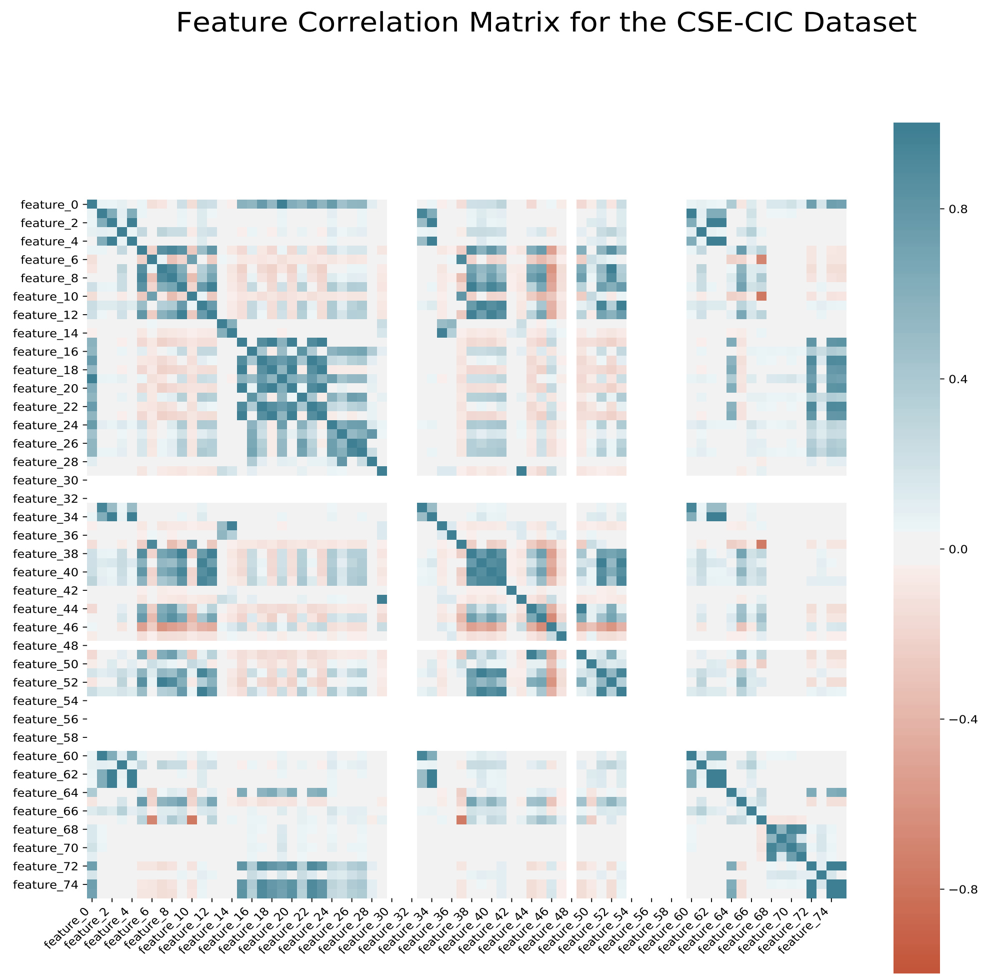


Figure 4.15: Feature Correlation for the CSE-CIC-IDS2018 dataset.

Again in Fig.4.16 we see a clear separation between the subsampled malicious and benign flows. In fact, it appears the malicious flows are very well clustered in one specific area. This shows the similarity amongst the botnet traffic when compared to benign. Because of this separation, neural network will surely be able to separate these two types of flows.

t-SNE analysis

As with the UNSW dataset and USTC dataset we also performed t-SNE analysis for malicious vs benign flows in the CSE-CIC-IDS2018 dataset.

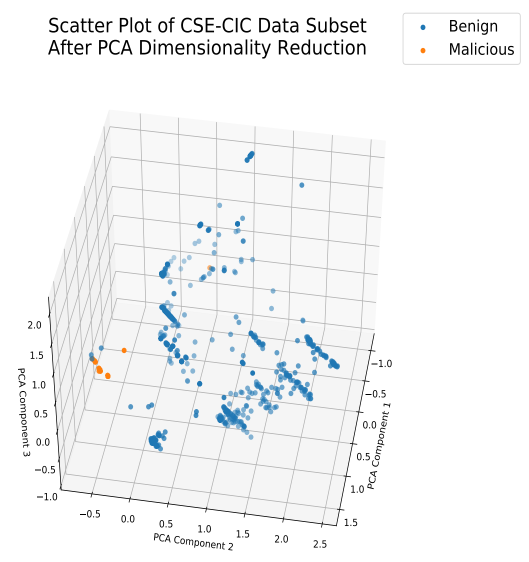


Figure 4.16: PCA Analysis of the CSE-CIC-IDS2018 dataset: Benign vs Malicious.

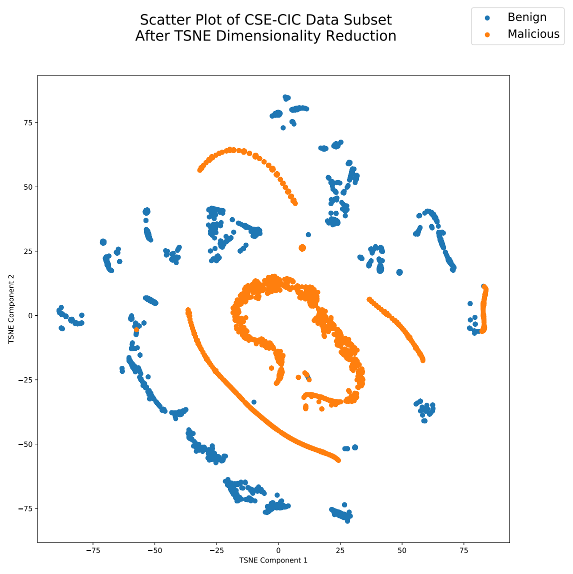


Figure 4.17: 2D t-SNE Analysis of the CSE-CIC-IDS2018 dataset: Benign vs Malicious.

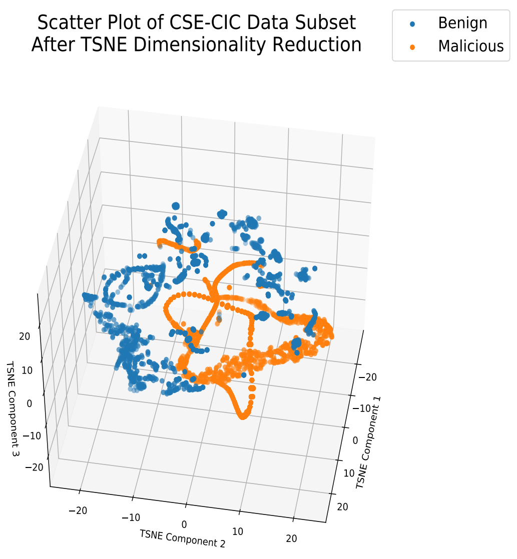


Figure 4.18: 3D t-SNE Analysis of the CSE-CIC-IDS2018 dataset: Benign vs Malicious.

Figs. 4.17 and 4.18 again confirm that there is a clear difference between the malicious/botnet and benign flows. This completes the analysis of the CSE-CIC-IDS2018 dataset.

Chapter 5

Set-up and Implementation

In this chapter, we consider the neural network implementation used in the experiments, how the models were trained, and give an overview of the metrics used in our experiments.

5.1 Neural Networks

In this section we outline, the neural networks that were used within this work and the training procedures that we used with them. The neural networks used included a 5-layer convolutional network, an 18-layer ResNet that made use of an ensemble method, and two different tree shaped deep neural networks that make use of the previous two neural networks. (For completeness we mention here that a LSTM [49] based approach was also attempted but returned dismal results, so we do not include it within the rest of our work or discussion).

Table 5.1: Considered Convolutional Neural Networks

Model	**Architecture Details**
CNN-0	Dense(784)-Conv2D(5,5,1,32)-MaxPool(2,2)-Conv2D(5,5,1,64)-MaxPool(2,2)-Dropout(0.2)-Dense(1024)-Dropout(0.4)-Dense(2)-Softmax() [61], [76]
CNN-1	Dense(784)-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,64)-BatchNormalization-Dense(1024)-Dense(2)-Softmax() [49], [36]
CNN-2	Dense(784)-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,64)-BatchNormalization-Dropout(0.2)-Dense(1024)-Dropout(0.4)-Dense(2)-Softmax() [49]
CNN-3	Dense(784)-Conv2D(5,5,1,16)-BatchNormalization-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,64)-Conv2D(5,5,1,128)-Dropout(0.3)-Dense(1024)-Dropout(0.5)-Dense(2)-Softmax() [49]
CNN-4	Dense(784)-Conv2D(5,5,1,16)-BatchNormalization-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,64)-Conv2D(5,5,1,128)-Dropout(0.4)-Dense(1024)-Dropout(0.5)-Dense(2)-Softmax() [49]
CNN-4	Dense(784)-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,32)-MaxPool(2,2)-BatchNormalization-Conv2D(5,5,1,64)-BatchNormalization-Conv2D(5,5,1,64)-MaxPool(2,2)-BatchNormalization-Conv2D(5,5,1,128)-Conv2D(5,5,1,64)-Dropout(0.3)-Dense(1024)-Dropout(0.5)-Dense(2)-Softmax() [49]
CNN-5	Dense(784)-Conv2D(5,5,1,32)-BatchNormalization-Conv2D(5,5,1,32)-MaxPool(2,2)-BatchNormalization-Conv2D(5,5,1,64)-BatchNormalization-Conv2D(5,5,1,64)-MaxPool(2,2)-BatchNormalization-Conv2D(5,5,1,128)-Conv2D(5,5,1,64)-Dropout(0.3)-Dense(1024)-Dropout(0.5)-Dense(2)-Softmax() [49]

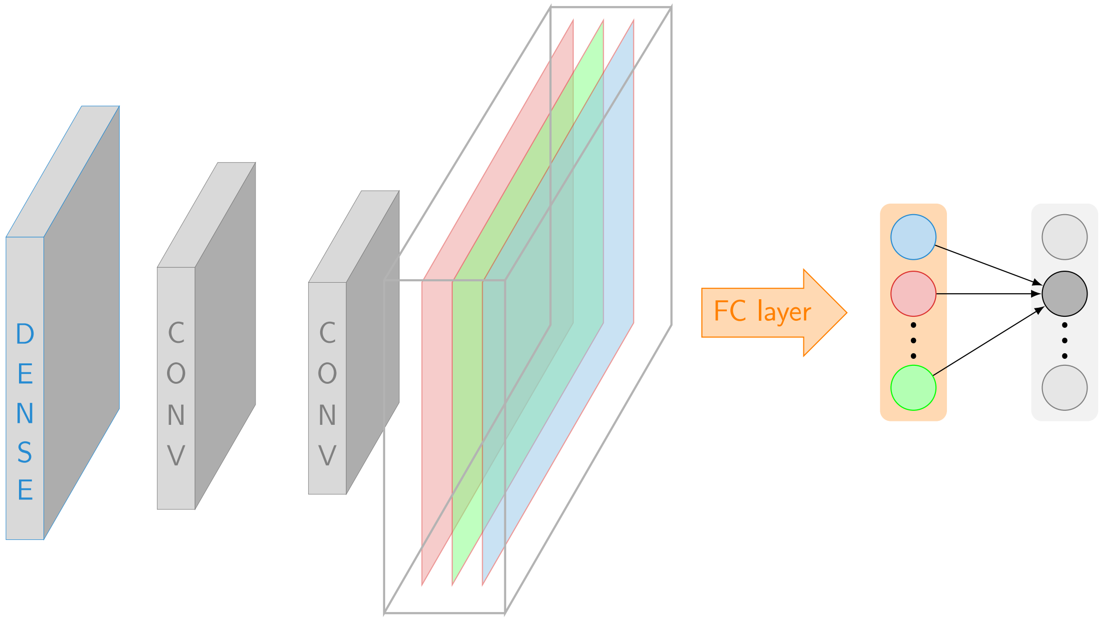


Figure 5.1: Diagram of the CNN-0 neural network. The CNN-0 network has a dense layer followed by two convolutional layers before going through a fully connected layer and then a SoftMax layer for prediction.

5.1.1 Convolutional Neural Network Implementation

Within Table5.1, Dense(n) describes a fully connected layer with n output nodes. Conv2D(n,o,p,q) describes a convolutional layer with kernel size (n,o), stride p, and q filters. Finally, Dropout (f) describes a dropout layer with rate f. These models were gleaned from [49], [76], [36], [61]. After training the above six models on the USTC dataset, we found that CNN-0 gave the best results in differentiating different malicious and benign flows. We thus chose to use this neural network for the basis of our implementation throughout the rest of this work. (Given that the crux of this work, is in creating a means of making these models robust to adversarial examples and identifying how adversaries can take advantage of these examples and not in creating the best means of differentiating flows in every situation, we leave finding the truly best model in all situation for this task to Future Work).

While training, we normalized all features to be within the range [0,1]. We further used an 80/10/10 split for the training, validation, and testing (this was both for the UNSW, USTC, and CSE-CIC datasets). In order to train this model, we used an Adam optimizer with a learning rate of $1x10^{-4}$. During training, we made use of early stopping based on the validation set. If we saw that the validation decreased twice in a row, we stopped training.

5.1.2 RseNet Implementation

In addition to using a convolutional network (CNN-0) in order to differentiate flows, we also used an 18-layer ResNet ensemble for this work [42]. Bhat et. al [30] used this ensemble ResNet in order to perform website fingerprinting based on sequence of packets and statistical flow data. In this work, we take their methodology and apply it to uncovering and recognizing deeper information in malicious traffic. For this architecture we only consider the USTC dataset as the UNSW and CSE-CIC datasets sizes were prohibitive.

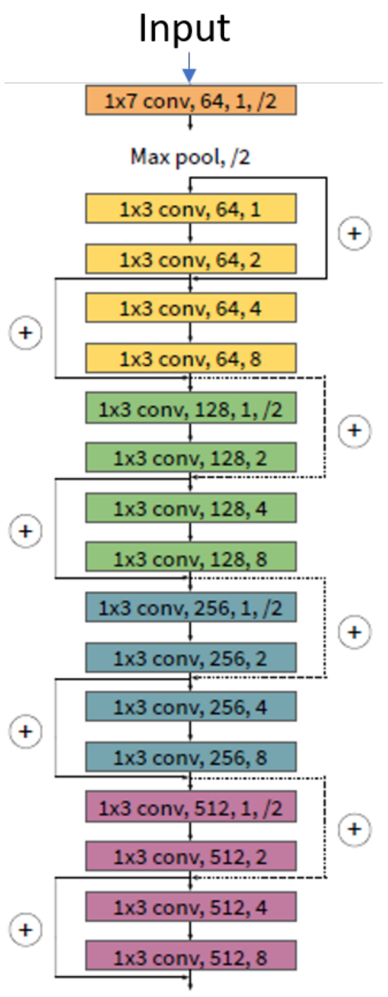


Figure 5.2: ResNet-architecture: This ResNet takes in sequential data [30].

In our work, for our sequential data, we consider not only the sequence of packet directions and packet timing information but also the sequence of packet sizes. For each of these three sequences we put them through an individual 18-layer ResNet that makes use causal padding and dilated convolutions. As discussed in the Background chapter, these aspects when including in convolutional layers allow for the convolutions to understanding longer term sequential trends in data. We further included the metadata that we gleaned from Joy after processing the timing data within each differently trained ResNet. We do this by concatenating the output timing embedding of the ResNet with the metadata before putting the combined embedding through a fully connected layer with Dropout.

This can be seen most clearly in Fig.5.4. This gives us three different get three different 1024-dimensional embeddings of the data. After putting this new embedding through a another fully connected layer with dropout, this final embedding is then sent through a final SoftMax layer.

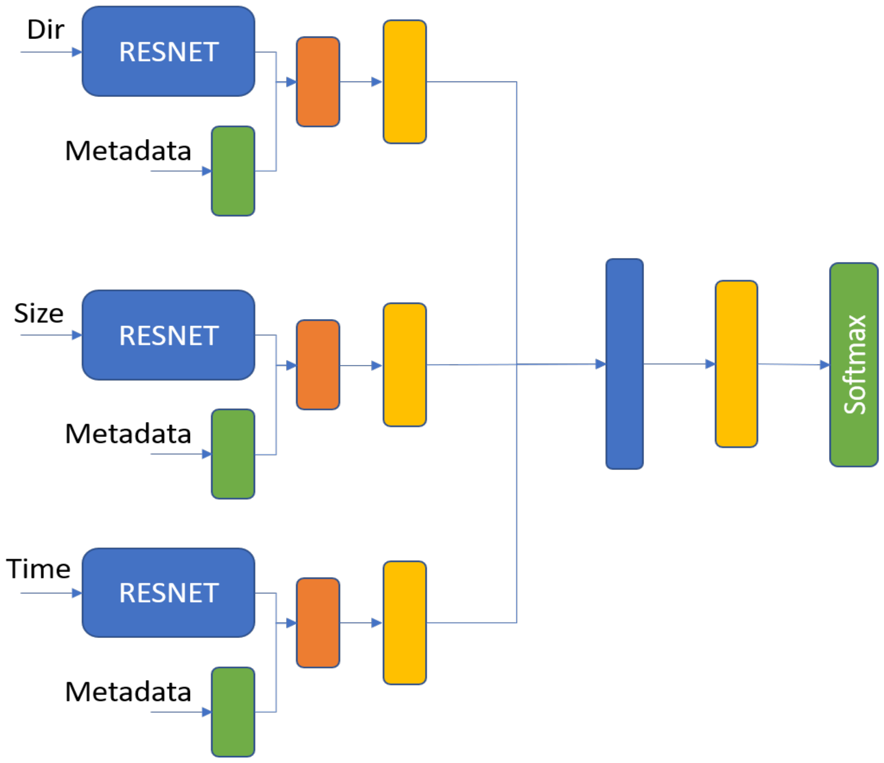


Figure 5.3: RESNET-implementation: This RESNET takes in timing, packet sizes, direction data as well flow metadata.

We used 128 as maximum sequence length for a single flow's timing information (i.e. we only consider the first 128 packets for the ResNet). We did this because at the 95th percentile, the length of the flows was only 36 packets. We thus padded all flows with 0s if they had less than 128 packets for the ResNet. Again, the dimensionality of the metadata in the USTC dataset was 158.

During training, we used an 80/10/10 split for the training, validation, and test set. In order to train this model, we used an Adam optimizer with an initial learning rate of 1.0×10^{-3}. After decreasing in accuracy twice in a row on the validation set, we decreased the learning rate by multiplying it by $\sqrt{0.1}$ until it reached a learning rate of 1×10^{-5}. Once it reached this learning rate, we stopped training after it decreased in accuracy on the validation set again.

5.1.3 Tree-Shaped Deep Neural Networks

In addition to the 5-layer convolutional network and the ensemble encoder, we made use of two different tree-shaped deep neural network. The idea behind the tree-shaped neural network is that after the first layer of tree separated out whether a flow was malicious or benign, the second layer of the tree would perform more fine-grained analysis and determine the type of malware/attack. The methodology of the two types of models used was inspired by Chen et al. [35]. As in Chen et al. [35], after determining whether a flow is malicious, an embedding of the flow from the second to last layer is taken from model and then concatenated with original input before a finer-grained classification is performed. For the first tree shaped model, the USTC data is placed through a pretrained ResNet that differentiates benign from malicious flows. After this, an embedding of size 1024 is retrieved from the second to last layer of the full ResNet model and concatenated with metadata. This new embedding is then fed to a new CNN-0 model for fine-grained analysis. During training of this tree-shaped model, we

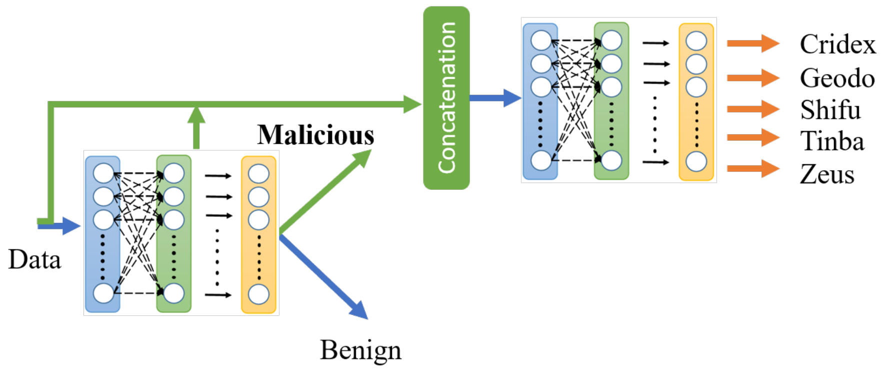


Figure 5.4: This model performs an 11-class classification in two stages for the USTC dataset. First the model determines whether a flow is malicious or benign. Second, the model performs a multi-class classification on the different types of botnets in the dataset. For the fine-grained multi-class classification, it makes use of an embedding found within the network that performed the classification between malicious and benign flows.

used 80/10/10 split for the training, validation, and test set. Further, in order to train this second branch of the tree model, we used an Adam optimizer with a learning rate of 1.0^{-4}. During training, we made use of early stopping based on the validation set. If we saw that the validation decreased twice in a row, we stopped training.

For the second tree-shaped model that we ran for both the USTC and the UNSW datasets (the subset of CSE-CIC that we used only contained botnets), the metadata for both model was placed through a pretrained CNN-0 model that differentiated benign from malicious flows. After this, an embedding of size 1024 (for both datasets) is retrieved from the second to last layer of the CNN-0 model and then concatenated with the original metadata. This new embedding is then fed to a new CNN-0 model for fine-grained analysis.

During training of this tree-shaped model, we used an 80/10/10 split for the training, validation, and test set. As before in order to train this second branch of the tree model, we used an Adam optimizer with a learning rate of 1.0^{-4}. During training, we again made use of early stopping based on the validation set.

5.2 Differentially Private Training

In this section, we give an overview differentially private noise was incorporated into the training methodology that we used when training our neural network architectures.

Training an (ϵ, δ)-differentially private robust model is similar to training ordinary models. For instance, the same loss and optimizer is used. For our purposes, in each model an Adam optimizer with categorical cross entropy was used during training. The main difference is the constraint on the sensitivity in the pre-noise layer. See the Background chapter on differential privacy robustness design for more details on this aspect.

After transforming the pre-noise layer to have sensitivity $\Delta <= 1$, we added either Gaussian or Laplace noise scaled by the Δ(1 in all cases within this work) and the maximum attack vector size L_{attack}. Because of post-processing property of differential privacy, this approach ensures that the output of the network is also differentially private and thus robust to changes in the input of size L_{attack}. How this noise is added of course also depends on which metric is being used: l_1-norm or l_2-norm (namely this changes how we normalize for sensitivity. For this work, we consider both in different experiments. Within this work, we considered L_{attack}-values varying from 0.1 to 1.0.

Autoencoder

In this work, instead of adding noise directly on the input or after the first layer of the network, we instead trained a stacked-autoencoder that we placed before the neural network that differentiated the flow type of the input. The autoencoder that we used had an encoder that first reduced the number of features to half of the number of input features before reducing them by half again. The decoder then increased the number of features by doubling the dimensional space before then returning the features to the original size. In many works, a convolutional auto-encoder is often used, however, in this work because we are not using images but rather a series of somewhat unrelated features, we simply used a series of fully connected layers for our autoencoder. In order to add noise to our autoencoder, we took three separate approaches. In the first approach, we added Laplacian noise after the first fully connected layer. Taking a sensitivity approach of $\Delta_{1,1}$, adding Laplacian noise here allows the model placed after the autoencoder to be robust to l_1-norm type attacks. Note that adding Laplacian noise allows for $(\epsilon, 0)$-differential privacy. See the Background chapter on differential privacy for more details. We secondly added Gaussian noise after the first layer. Taking a sensitivity approach of $\Delta_{1,2}$, this approach, allows the model placed after the autoencoder to be robust to l_1-norm types of attacks. Note that adding Gaussian noise allows for (ϵ, δ)-differential privacy. Despite the δ, this does not affect the later robustness calculations. Lastly, we added Gaussian directly onto the input before putting the input into the autoencoder. Taking a sensitivity approach of $\Delta_{2,2}$, this approach allows the model placed after it to be robust to l_2-norm type of attacks.

Note that while training a single draw of noise is used. Only during testing are multiple independent draws of noise used. These three different types of encoders were trained for all datasets as well as for an attack range L_{attack} from 0.1 to 1.0.

Full Model

After completing the training of the autoencoder and separately training each model, the static output of the autoencoder was fed into each pretrained model. Due to complexity of each model, once the autoencoder input was immediately input into the pretrained model, there is a significant drop in accuracy (even for small amounts of noise). Because of the post-processing property of differential privacy, we can however train the attached differentiating model for a few epochs in order to make it adjust to the noisy input. As Lecuyer et al. [47] did, we trained our differentiating (CNN-

0/ResNet) model on the autoencoder outputs for 3 epochs to get it to adjust to the noisy inputs.

5.3 Evaluation Metrics

Before moving into the results section of this work, in this section, we give an overview of the metrics that we used in order to evaluate the models. Note for each model, when running adversarial attacks (unless explicitly stated otherwise), we focused on targeted attacks that changed malicious flows in appearing as benign.

After creating full models that take into account differential privacy, we evaluated these models using two main metrics. The first metric is ordinary *accuracy*. This denotes the fraction of the testing set on which the model was correct. Within this first metric we also evaluated (most importantly) false negatives. False negatives give the percentage of malware that was labelled as benign. Labelling malware as benign is potentially catastrophic given that these flows would not be investigated and thus they would continue to wreak havoc within a given network.

We also evaluated our networks based on *certified accuracy*. Certified accuracy denotes the fraction for which the testing set on which a model's prediction are both *correct* and *certified robust* for a given threshold [47] (see the Background chapter on differential privacy for details on how this is done).

Formally these metric are as follows:

1. *Conventional accuracy:* $\frac{\Sigma_{i=1}^{n} isCorrect(x_i)}{n}$ where n is the size of the set.
2. *Certified accuracy:* $\frac{\Sigma_{i=1}^{n} isCorrect(x_i) \& robustSize(scores,\epsilon,\delta,L) \geq T}{n}$ where n is the size of the set and L is size of the adversarial attack that the model is trained against and T is an artificial numerical threshold that to which examples should be robust.

In addition to looking at the above metrics, we lastly considered the distribution of size of the robustness size.

Chapter 6

Attacks: Experiments and Results

In this chapter we focus on the details of the experiments run to attack our given networks using adversarial attacks. In order to conduct these attacks, we used cleverHans, an open source software that implements several of the most up-to-date adversarial attacks on neural networks (see Chapter 4 for details on this software).The experiments run for our adversarial attacks were: (1) black-box attacks with data synthesis through Jacobian augmentation, (2) black-box HopSkipJump attacks on the networks, (3) white-box attacks using the Sparse l_1 Descent attack,(4) targeted attacks on particular features of the neural network. Although we present the results for fine-grained analysis in this section, this work focuses mainly on differentiating malware from benign flows. As such these networks appear here only to show the real possibility of being able to differentiate amongst different types of malware just from statistical data. In order to see our code or to rerun our experiments, please see our anonymous GitHub: https://github.com/DPSelectro/DPNetwork.

6.1 Model Initial Results

Before we go into the results of the adversarial attacks and the subsequent description of the defences, we first give baseline results for each of the models.

6.1.1 CNN-0 UNSW: Malicious vs Benign Classification

Here, we give the baseline result for the CNN-0 on the UNSW dataset. Again, note that for the purpose of this work only the HTTP flows from the UNSW dataset were used. For training, the default settings outlined in Chapter 5 were used. We received the following results on the dataset.

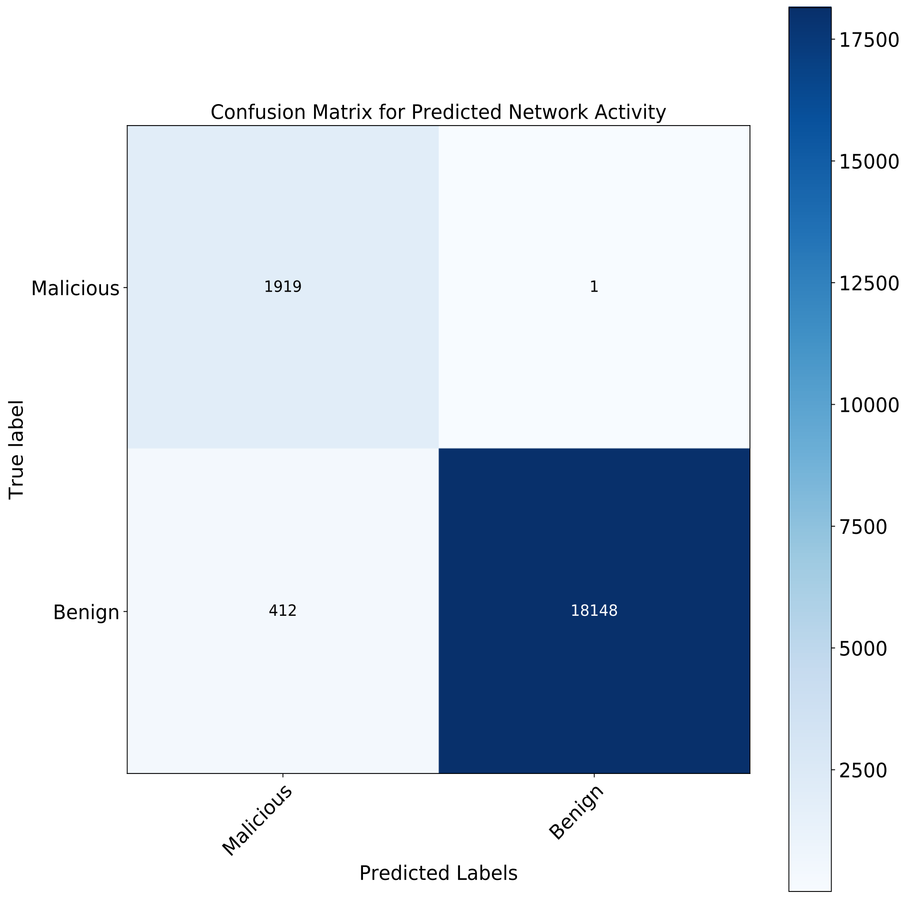


Figure 6.1: Baseline Confusion matrix for the UNSW dataset: Benign vs Malicious.

As seen in Fig. 6.1, the false negative rate was fairly low at 5×10^{-4}, which is basically 0. The recall likewise was nearly 1. The false positive rate was also fairly low at 2.2%. This would mean that there would be small about of extra work in investigating these positives. The overall accuracy of the network was 97.98%.

6.1.2 CNN-0 USTC: Malicious vs Benign Classification

We give here the baseline result for the CNN-0 on the USTC dataset. Again, for training , the default settings outlined in Chapter 5 were used. We received the following results on the dataset.

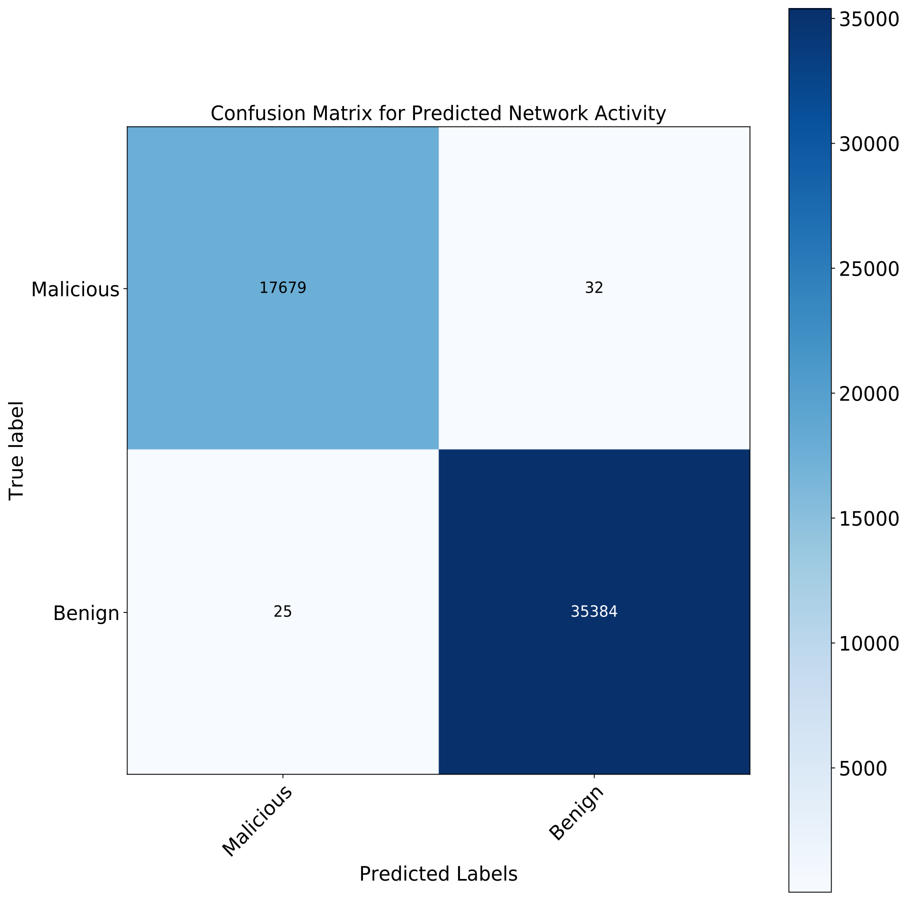


Figure 6.2: Baseline Confusion matrix for the USTC dataset: Benign vs Malicious.

As seen in Fig. 6.2, the false negative rate was fairly low at 1×10^{-3}, which is basically 0. The recall, likewise, was nearly 1. The false positive rate was also very low at 7×10^{-4}. The overall accuracy of the network was 99.89%.

6.1.3 CNN-0 CSE-CIC: Malicious vs Benign Classification

We give here the baseline result for the CNN-0 on the CSE-CIC dataset. The default settings outlined in Chapter 5 were used. We received the following results on the dataset.

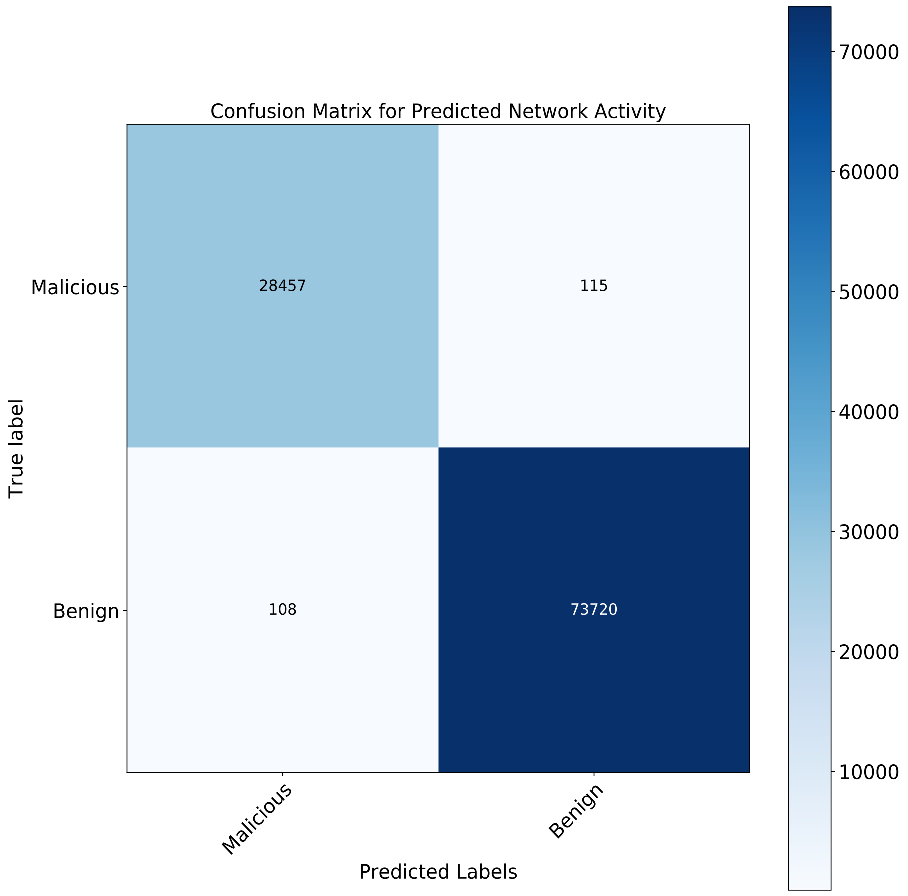


Figure 6.3: Baseline Confusion matrix for the CSE-CIC dataset: Benign vs Malicious.

As seen in Fig. 6.3, the false negative rate was fairly low at 4×10^{-3}, which is basically 0. The recall likewise was nearly 1. The false positive rate was very low at 1.4×10^{-3}. The overall accuracy of the network was 99.78%.

6.1.4 ResNet USTC: Malicious vs Benign Classification

We give here the baseline result for the ResNet on the USTC dataset. Note that this ResNet made use of directions, packet sizes, and packet inter-arrival sequences. For training, the default settings outlined in Chapter 5 were used. We received the following results on the test set.

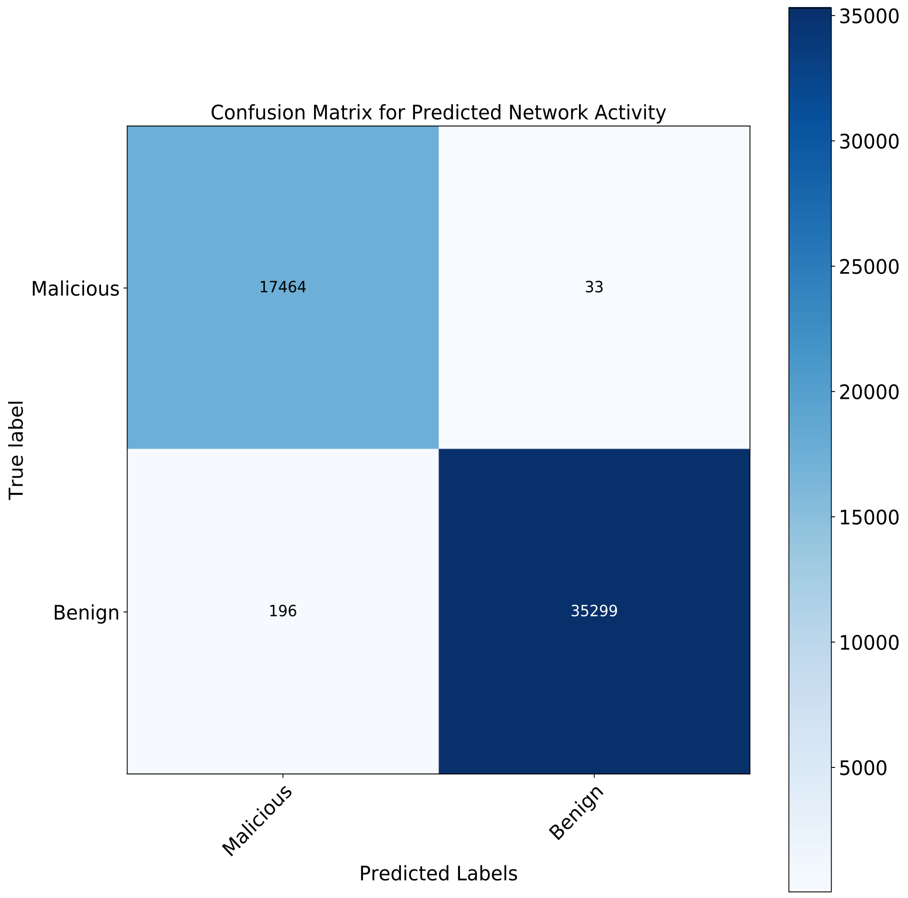


Figure 6.4: Baseline Confusion matrix for the USTC dataset: Benign vs Malicious.

As seen in Fig. 6.4, the false negative rate was fairly low at 1.9×10^{-3}, which is basically 0. The recall likewise was nearly 1. The false positive rate was very low at 5.5×10^{-3}. The overall accuracy of the network was 99.6%.

6.1.5 Simple Tree CNN-0(2) UNSW: Fine-Grained Analysis

Here we present the results for the fine-grained differentiation of the UNSW dataset's different types of network attacks. Again, the default settings were used for training. The CNN-0 model weights that were created while training CNN-0: UNSW Malicious vs Benign were used for the first branch of this tree. We received the following results after evaluating on the given test set.

The conventional accuracy of this model overall was 81.6%. For the interaction between how the model mislabelled different types of malware as other types see Fig. 6.5. This confirms that a tree-based approach can be used to differentiate among different types of network attacks using statistical features (at least for the UNSW-NB15 dataset).

6.1.6 Simple Tree CNN-0(2) USTC: Fine-Grained Analysis

Here we present the results for fine-grained analysis of the USTC dataset in differentiating different types of malware. The default settings were used for training. The CNN-0 model weights that were created while training CNN-0: USTC Malicious vs Benign were used for the first branch of this tree. We received the following results after evaluating on the given test set.

The conventional accuracy of this model overall was 88.1%. For the interaction between how the model mislabelled different types of malware as other types see Fig. 6.6.

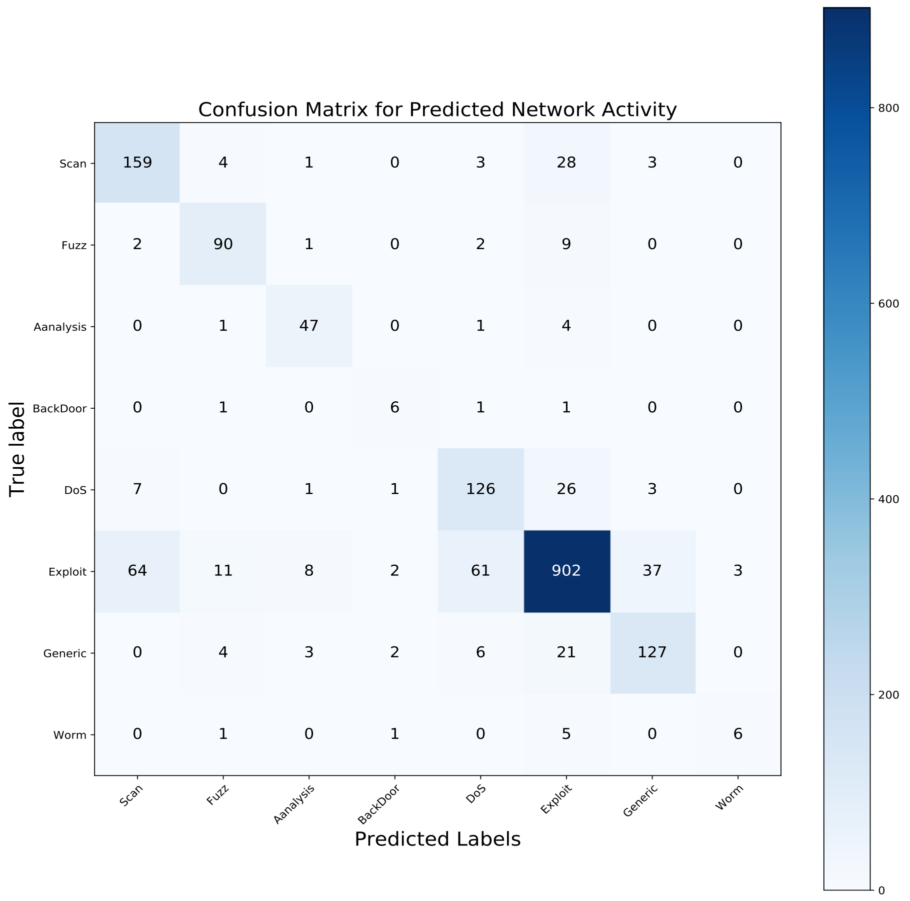
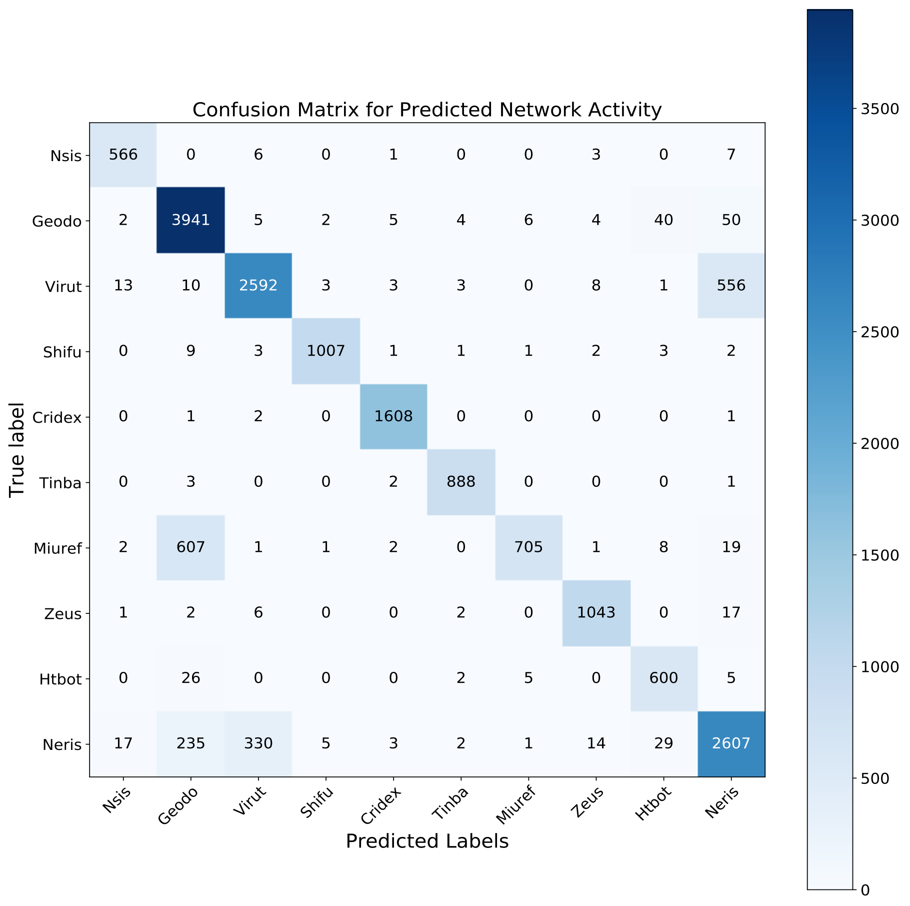


Figure 6.5: Confusion matrix for the UNSW dataset for differentiating different types of network attacks.

6.1.7 Complex ResNet +CNN-0 Tree USTC: Fine-Grained Analysis

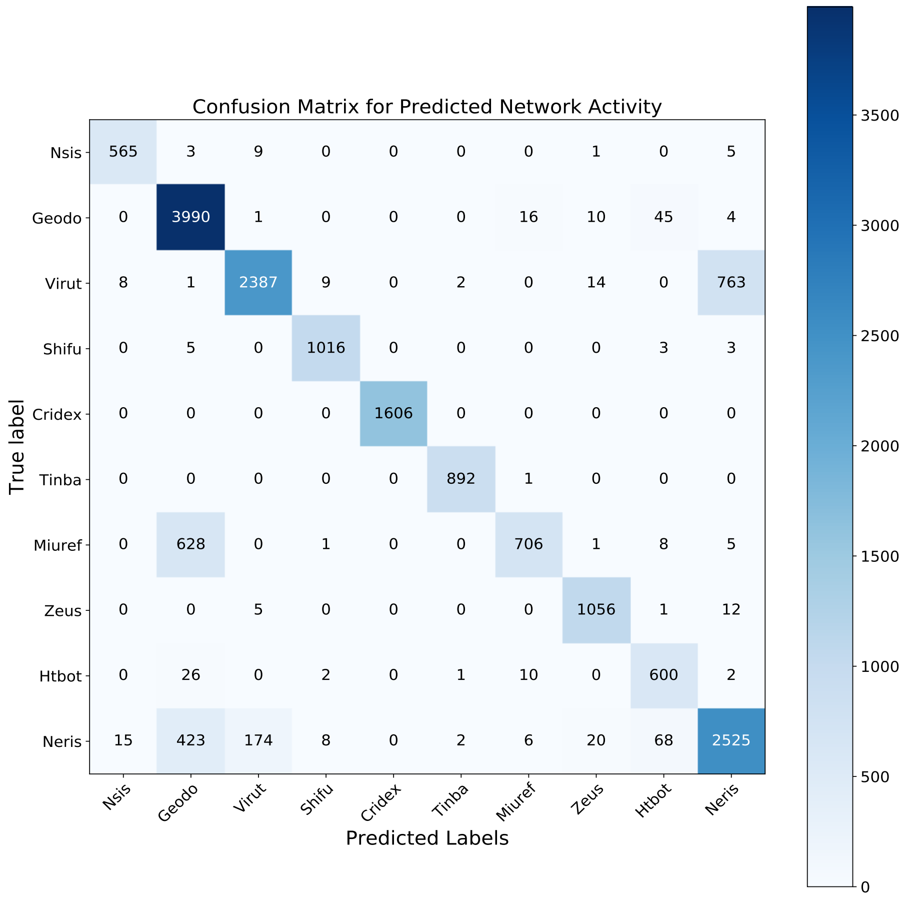
Here we present the results for fine-grained analysis of the USTC dataset in differentiating different types of malware when using ResNet for +CNN-0 combination. The default settings were used for training. The RESNET model weights that were created while training ResNet: USTC Malicious vs Benign were used for the first branch of this tree. We received the following results after evaluating on the given test set. The conventional accuracy of this model overall was 86.8%. For the interaction between how the model mislabelled different types of malware as other types see Fig. 6.7.



	Nsis	Geodo	Virut	Shifu	Cridex	Tinba	Miuref	Zeus	Htbot	Neris
Nsis	566	0	6	0	1	0	0	3	0	7
Geodo	2	3941	5	2	5	4	6	4	40	50
Virut	13	10	2592	3	3	3	0	8	1	556
Shifu	0	9	3	1007	1	1	1	2	3	2
Cridex	0	1	2	0	1608	0	0	0	0	1
Tinba	0	3	0	0	2	888	0	0	0	1
Miuref	2	607	1	1	2	0	705	1	8	19
Zeus	1	2	6	0	0	2	0	1043	0	17
Htbot	0	26	0	0	0	2	5	0	600	5
Neris	17	235	330	5	3	2	1	14	29	2607

Figure 6.6: Confusion matrix for the USTC dataset for Malware.

This completes our presentation of the baseline results of all the models that we considered within this work. For the rest of this work, we further limit our discussion to the CNN-0 neural network as that gave us the best results for the USTC dataset. For the CNN-0 neural network, we run these adversarial attacks for both the UNSW, USTC, and the CSE-CIC dataset. Despite our focus on this model, our approach is **agnostic** to the model that is used, and we only use this model to illustrate the possible attacks that can be levied against it and to showcase our defence. These types of attacks can be levied against any system that makes use of statistical features in order to label flows types (i.e. they can also use other features are well). We use neural networks here for their effectiveness and convenience in this setting. For our attacks we focus on



True label \ Predicted	Nsis	Geodo	Virut	Shifu	Cridex	Tinba	Miuref	Zeus	Htbot	Neris
Nsis	565	3	9	0	0	0	0	1	0	5
Geodo	0	3990	1	0	0	0	16	10	45	4
Virut	8	1	2387	9	0	2	0	14	0	763
Shifu	0	5	0	1016	0	0	0	0	3	3
Cridex	0	0	0	0	1606	0	0	0	0	0
Tinba	0	0	0	0	0	892	1	0	0	0
Miuref	0	628	0	1	0	0	706	1	8	5
Zeus	0	0	5	0	0	0	0	1056	1	12
Htbot	0	26	0	2	0	1	10	0	600	2
Neris	15	423	174	8	0	2	6	20	68	2525

Figure 6.7: Confusion matrix for the USTC dataset for differentiating different types of Malware,

networks that differentiated malicious from benign software as that is the main focus on this work.

6.2 Black-Box l_1-norm Adversarial Attacks using Jacobian-based Dataset Augmentation and l_2-norm Attacks using HopSkipJump

To begin, we ran black box attacks against our models that differentiated benign from malicious network traffic. This attack was performed on the CNN-0 models specifically. Note that in this scenario, an adversary only has access to the labels returned on specific flow instances. This is the equivalent of the adversary sending in different instances of flows into a network and then determining whether the flow was detected/investigated as malicious or not. In this section we give the results for two types of these attacks: Jacobian-based Dataset Augmentation for l_1-norm attacks and l_2-norm Attacks using the HopSkipJump attack.

We use the process for Jacobian-based Dataset Augmentation for l_1-norm black-box attacks as outlined by Papernot et al. [56]. As was shown in Fig. 2.4 in order to accomplish this, we create substitute models after performing data augmentation, training said model on the synthetic data. In this work, for the substitute model training, we create a 16384-size dataset. During the training of the substitute model, we use a generic 3-layer convolutional network. On these substitute model, in order to perform adversarial attacks, we use the Fast-Gradient Sign Method (FGSM). We specifically constrain this attack L_{attack} by a maximum on the l_1-norm for the respective test sets. After performing this attack on the substitute model, we then ran the adversarial examples we generated on original model from which the synthetic dataset was originally generated. We then ascertained the accuracy of the original model on the adversarial examples generated. Note that ordinarily the models had high accuracies (upwards of 97.9% in every case) for their respective validation set. For more details see our Background chapter on substitute model training.

We used the HopSkipJump attack [34] for the l_2-norm black black-box attack because it is optimized for that norm specifically. For details on the HopSkipJump attack see our Background chapter or [34].

To implement FGSM for the Jacobian-based approach and the HopSkipJump attack we used CleverHans.

In addition to running both l_1 and l_2-norm attacks, we also within those attack ran two different types those attacks:(1) We first ran attacks specifically targeting turning malicious examples into appearing benign and (2) we ran attacks that targeted turning benign examples into appearing malicious. These both accorded with the goals we outlined in Chapter 3. For each type of attack, we considered the range of L_{attack} from 0.1 to 1.0.

6.2.1 UNSW: CNN-0 model

We give the results for the UNSW black-box attack here. We begin with the attack that attempts to transform malicious flow traffic into appearing benign.

As seen in Fig. 6.8, the Jacobian-based black-box attack was not extremely effective. While it was able to generate adversarial examples, even at a size of 0.1, these examples did not make up a substantial portion of the dataset. Despite this, as shown in Fig 6.8, on this dataset, even without access to the model's weight and design, it was still possible to generate adversarial examples to trick the model .As seen in Fig. 6.9, the HopSkipJump attack also remains relatively ineffective until the attack size reaches 0.8 in terms of l_2 perturbation. After this, the attack is able to move the accuracy down to around 87% with an attack range of up to 1.4. This shows that if an adversary truly wished to attack this model using only a black-box attack, he would be successful

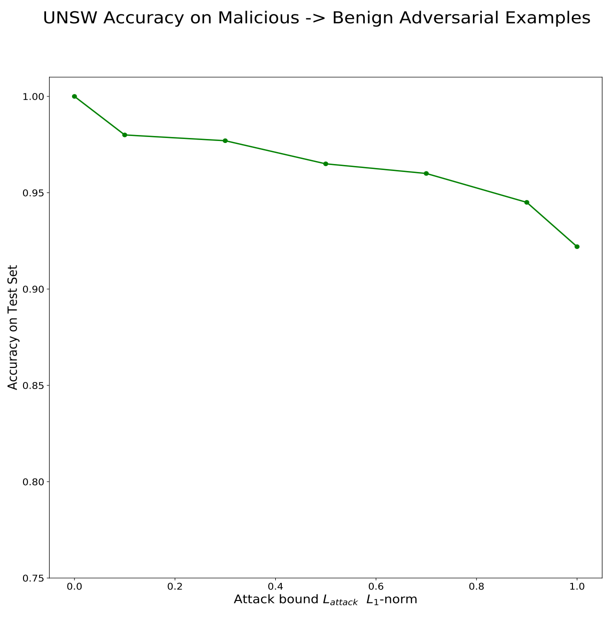


Figure 6.8: Accuracy of UNSW CNN-0 under Jacobian-based l_1-norm black box attacks of varying maximum-size L_{attack}.

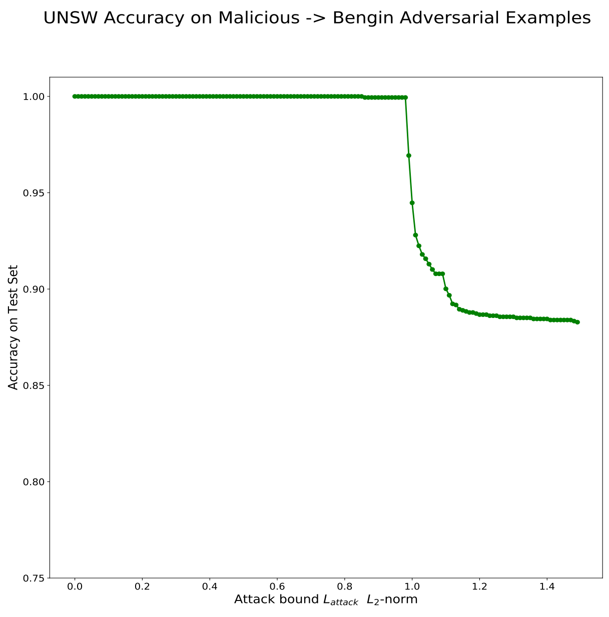


Figure 6.9: Accuracy of UNSW CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

in generating a *few* malicious flows that would go undetected. We now move on to the attack that attempts to generate adversarial examples to transform benign flow traffic into appearing malicious for the UNSW dataset. We see in Fig. 6.10, that the attack to transform benign flows into malicious flows was slightly more effective. Even at the relatively small attack l_1-norm size of 0.1, an attacker could still manage to transform around 12% of the testing set benign flows into malicious flows. Given the magnitude of the number of flows that could potentially be going through a network, being able to transform over 12% of these into being perceived as malicious evidences a clear ability to overwhelm a system that detects malicious flows. Furthermore, as seen in Fig 6.11, we are able to perform a relatively effective attack against the network using the HopSkipJump attack. Specifically, we see a large decrease in the accuracy of model when the size of the attack vector size $L_{attack} > 0.4$. After this value, accuracy

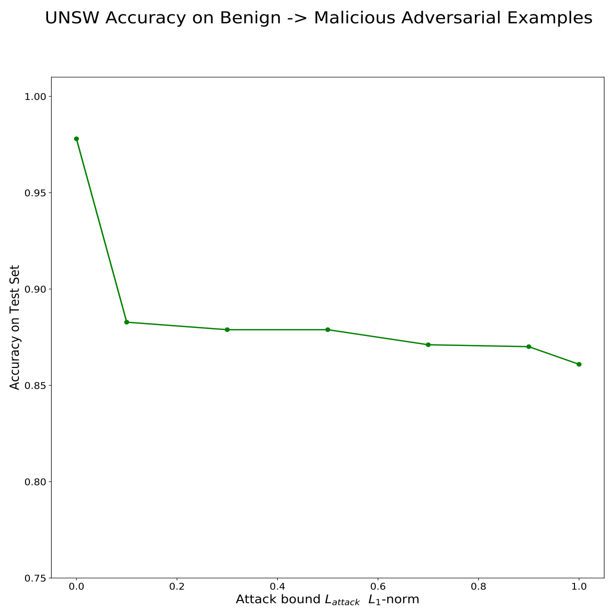


Figure 6.10: Accuracy of UNSW CNN-0 under Jacobian-Based Data Aug FGSM l_1-norm black box attacks of varying maximum-size L_{attack}.

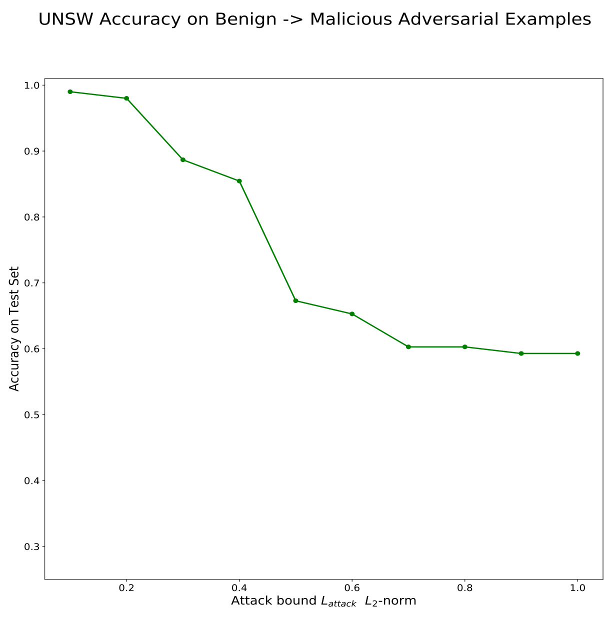


Figure 6.11: Accuracy of UNSW CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

of the model decreases to just over 60%. This shows that an adversary could manage to transform a significant amount of benign traffic into appearing malicious.

6.2.2 USTC: CNN-0 model

We now move on to exploring the efficacy of black-box attack on the USTC dataset using the CNN-0 model for differentiating malicious from benign flows. We begin with the attack that transforms malicious type flows into appearing benign. We see in Fig. 6.12 similar behaviour to that of FGSM attack on the UNSW CNN-0 model. At a maximum L_{attack} of 1.0, this type of attack can transform around 10% of the testing set into adversarial examples. Overall, while this attack again was able to generate adversarial examples even at a maximum perturbation size L_{attack} of 0.1, it was not overwhelmingly effective. As seen in Fig. 6.13, the HopSkipJump attacks is also able to steadily decrease the accuracy of the model as size of the attack increases. Despite

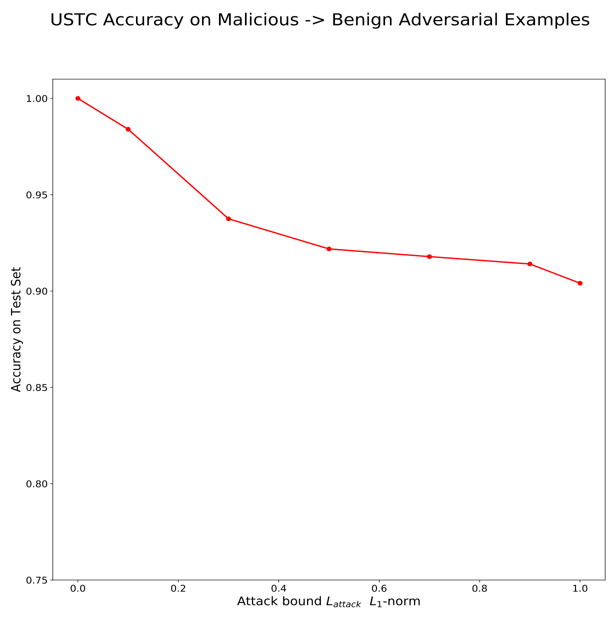


Figure 6.12: Accuracy of USTC CNN-0 under Jacobian-based l_1-norm black box attacks of varying maximum-size L_{attack}.

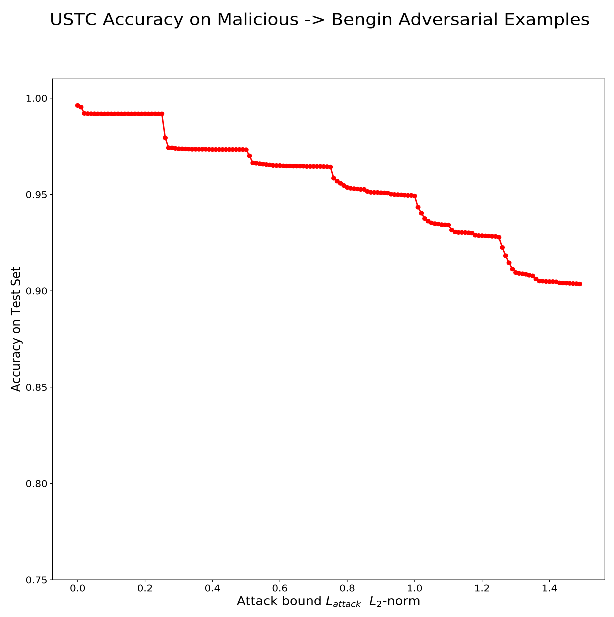


Figure 6.13: Accuracy of USTC CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

this, the HopSkipJump attack is still unable to create an incredibly significant decrease in accuracy on the model. With the maximum considered attack size $L_{attack} = 1.0$ the accuracy on test set decreases only to 91%. Despite this, these results collectively do exhibit the capability of an adversary to be able to generate malicious to benign adversarial examples, even if they are small in number. For these types of attack, being able to create even one targeted attack could have tremendous repercussions.

We now move on to the attack that transforms benign flows into appearing malicious for the USTC dataset.

Here in Fig. 6.14, we see significant results. An adversary in this scenario, would be able to transform over 30% of the incoming traffic into appearing malicious by only perturbing the features by 0.1 (l_1-norm). Given even more leeway, an adversary could end up transforming over 40% of observed benign traffic into appearing malicious. As seen in Fig 6.15, we again see the effectiveness of this type of attack against the CNN-0

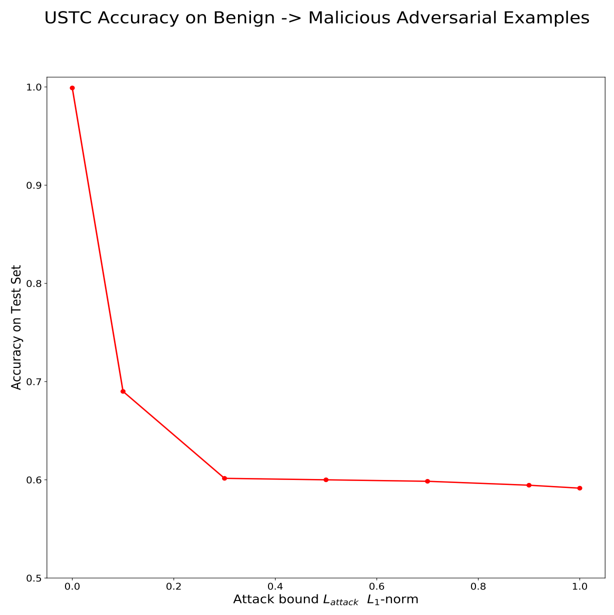


Figure 6.14: Accuracy of USTC CNN-0 under Jacobian-based l_1-norm black box attacks of varying maximum-size L_{attack}.

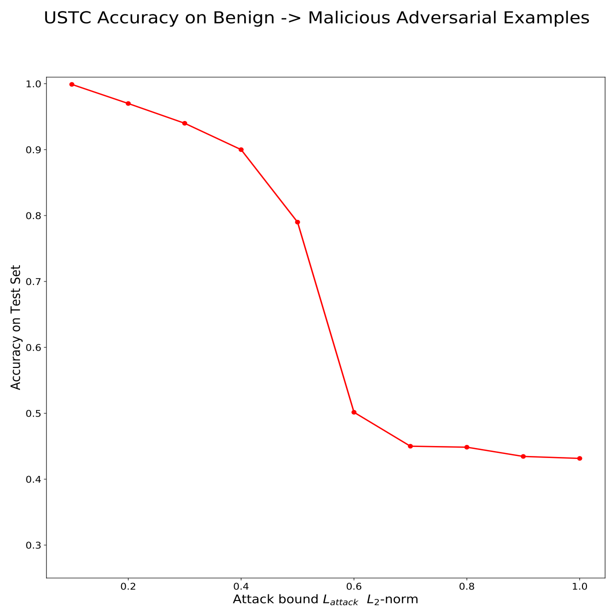


Figure 6.15: Accuracy of USTC CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

network. This attack allows the adversary to obtain an accuracy below 50% when the attack vector size $L_{attack} > 0.5$. This shows the efficacy of this type of the attack and the need to be able to defend against it. This importance is only reinforced by the fact that this is a black-box attack and thus can be run without any additional knowledge about the system. Given this capability, here, the adversary has a clear advantage. They could easily end up overwhelming an institution that makes use of a neural network to detect malicious traffic. By making a substantial portion of the traffic appear malicious, the adversary could ensure that the institution is unable to investigate it all.

6.2.3 CSE-CIC: CNN-0 model

We finally move on to exploring the results of black-box attack on the CSE-CIC dataset using the CNN-0 model for differentiating malicious from benign flows. We again begin with the attack that transforms malicious type flows into appearing benign.

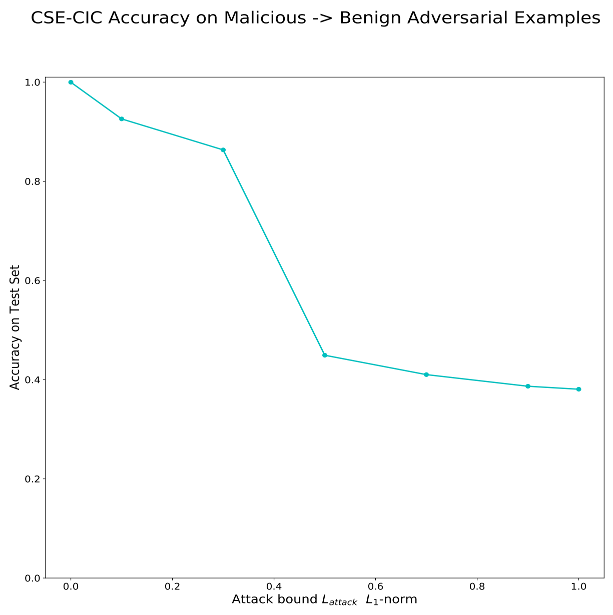


Figure 6.16: Accuracy of CSE-CIC CNN-0 under Jacobian-based l_1-norm black box attacks of varying maximum-size L_{attack}.

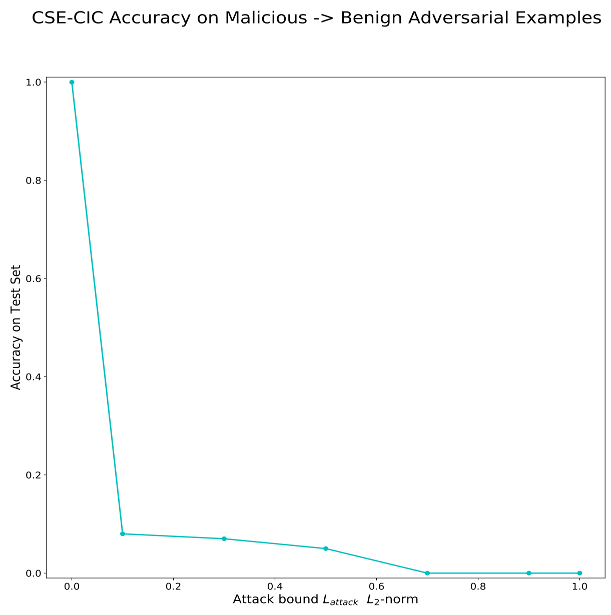


Figure 6.17: Accuracy of CSE-CIC CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

We see in Fig. 6.16, tremendously successful behaviour in terms of transforming malicious examples into benign one. With an attack vector size of $L = 0.3$, the adversary can transform upwards of 15% of observed malicious traffic into appearing benign. Similarly, with an attack vector $L \geq 0.5$, the adversary can transform over 60% of malicious traffic. Furthermore, as seen in Fig. 6.17, the HopSkipJump attacks is also able to majorly decrease the accuracy of the model as size of the attack increases. Even with an attack vector size $L = 0.1$, an attack is able to transform over 90% of traffic into appearing benign. These results for this dataset confirm an overwhelming ability of an adversary to transform malicious data into appearing benign and thus of infecting a system. We now move on to the attack that transforms benign flows into appearing malicious for the CSE-CIC dataset.

Here for transforming benign flows into appearing malicious, we see very different behaviour. Namely both our l_1-norm and l_2-norm attacks in Fig. 6.18 and Fig. 6.19

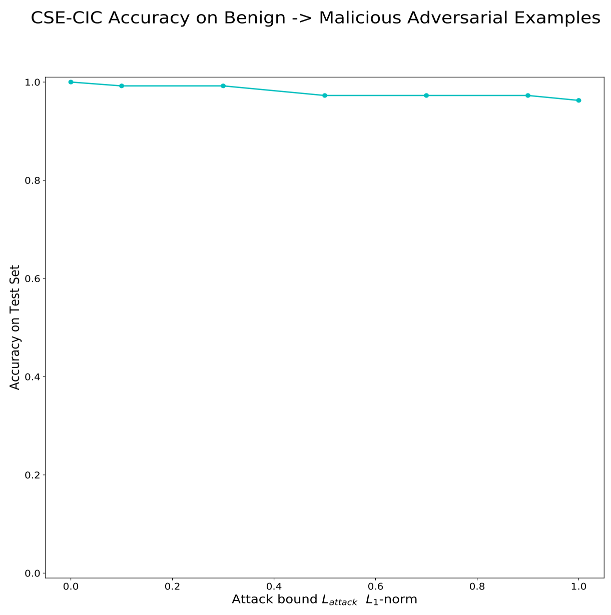


Figure 6.18: Accuracy of CSE-CIC CNN-0 under Jacobian-based l_1-norm black box attacks of varying maximum-size L_{attack}.

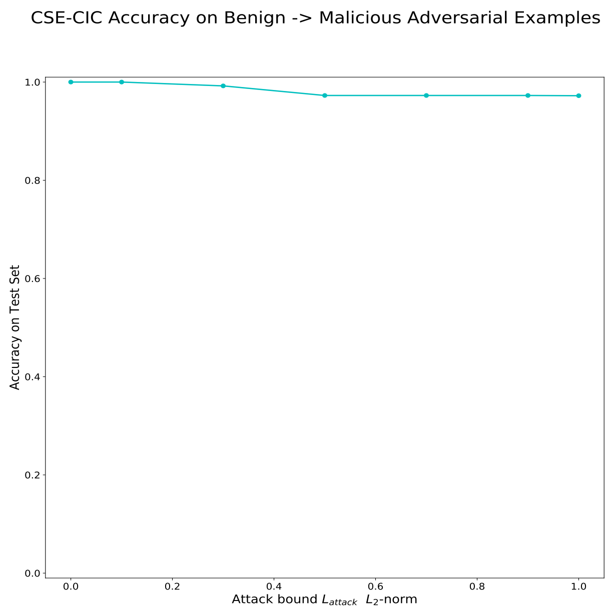


Figure 6.19: Accuracy of CSE-CIC CNN-0 under HopSkipJump l_2-norm white-box attacks of varying maximum-size L_{attack}.

respectively remained relatively ineffective. For both, they did not manage to go below 90%. This shows (for at least for this dataset), the difficulty of an adversary in using a black-box attack to transform benign flows into appearing malicious. The behaviour for this dataset was the reverse of what we observed for the UNSW and the USTC datasets. We believe that this may have to do with how we resampled our data. See our Discussion chapter for more details.

Now that we have given the results for all black-box attacks on our proposed networks, we will now move on to show that given additional information and data about a given model (white-box setting), even more devastating types of attacks can be conducted.

6.3 White-Box Attacks using Sparse l_1 Descent Attacks and l_2-norm Projected Gradient Descent

In this section, we present white-box attacks on our models using both Sparse l_1 Descent Attacks and Madry et al.'s [51] Projected Gradient Descent (PGD). Sparse l_1 Descent Attacks (2019) are known to outperform the FGSM approach and be comparable to Elastic-Net Method. In addition, they are incredible efficient (as the Elastic-Net Method can be prohibitive due to the power and time resources that it consumes). Similarly, Madry et al.'s [51] PGD, according to the authors is the most powerful 1st order adversarial attack for the l_2-norm and is not resource-intensive. In order to implement the Sparse l_1 Descent Attack and Madry et al [51]'s PGD we made use of CleverHans.

The specific scenario for this type of attack is that a person with access to a given model wants to sabotage it using his knowledge of a mode's weights and design.

6.3.1 UNSW: CNN-0 model

We give the results for the UNSW Sparse l_1 Descent Attack and the Madry et al. PGD white-box attacks here. We begin with the attack that attempts to transform malicious flow traffic into appearing benign.

As seen in Fig. 6.22, there is a sharp drop in accuracy after the maximum size of 0.3. When compared to Fig. 6.8, Fig. 6.22 shows the large advantage that this type of attack has over black-box attacks. The additional information allows the attack to transform over 30% of the malicious flows into appearing benign at We further see that l_2-norm attacks a fairly sizeable decrease in accuracy after the size of the attack L_{attack} is above 0.3.

We now present the results for an adversary attempting to transform benign flow into malicious flows.

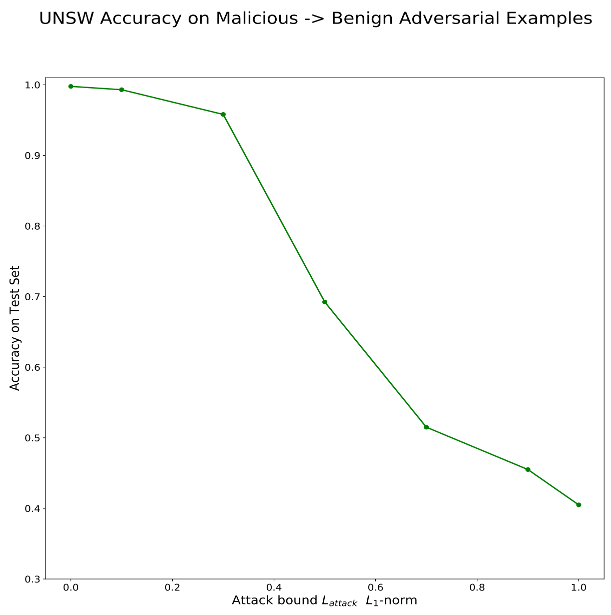


Figure 6.20: Accuracy of UNSW CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}.

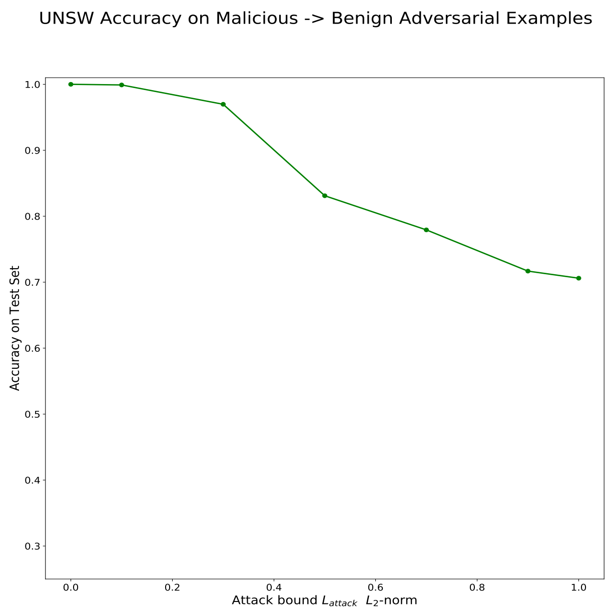


Figure 6.21: Accuracy of UNSW CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

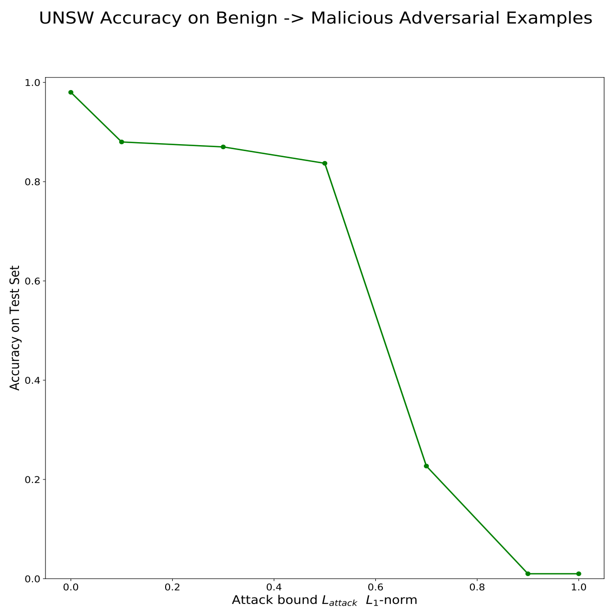


Figure 6.22: Accuracy of UNSW CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}.

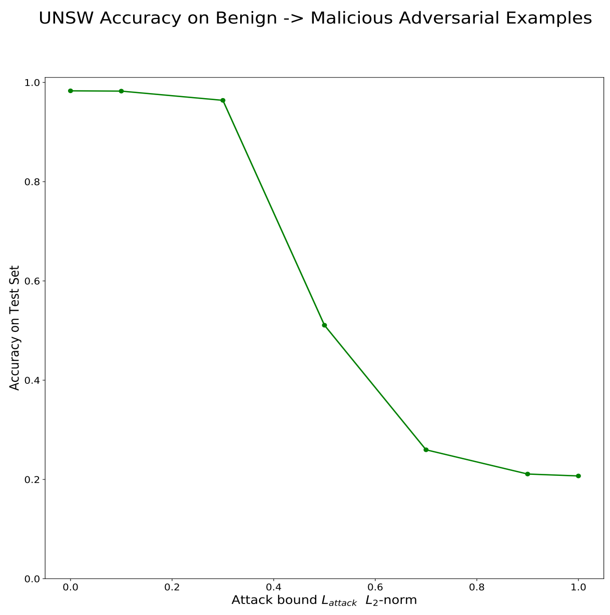


Figure 6.23: Accuracy of UNSW CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

From Fig. 6.22, it is clear that this l-norm attack was successful. One can easily observe how an adversary could potentially craft a heavy torrent of 'fake' malicious traffic in order to overwhelm a given system. This is especially true if the adversary can craft adversarial examples with l_1-norm perturbation attack vectors that are greater than 0.5. At this level, this sort of attack could be devastating. Similarly, the l_2-norm PGD attack was also largely successful. This attack further confirms the ability of an adversary to perform horribly deafening attacks against the network by producing adversarial examples.

In order to better illustrate what these adversarial attacks look like, we performed t-SNE analysis on a subset of benign and malicious flows and then placed the adversarial example within this 3-D environment.

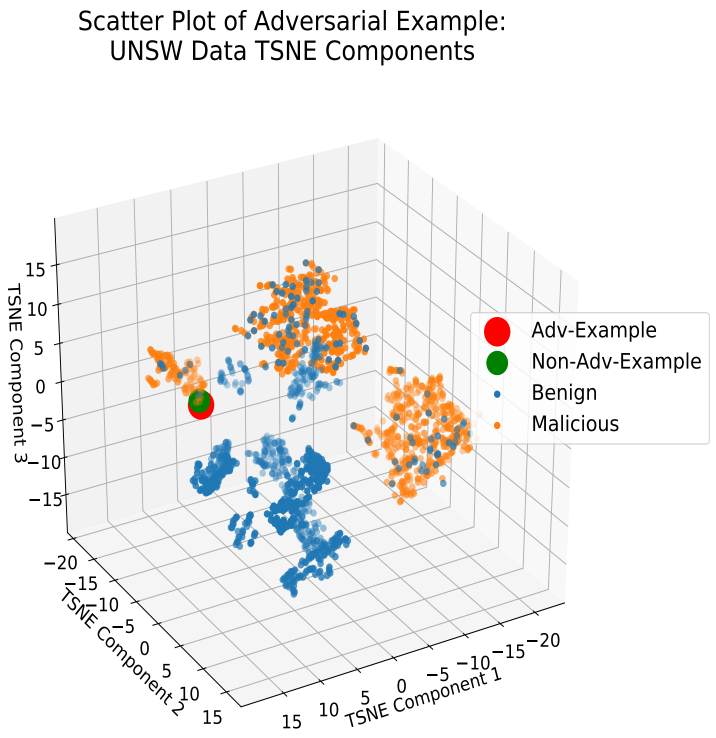


Figure 6.24: t-SNE of adversarial example with surrounding malicious and benign flows of the UNSW dataset. This shows at attack with maximum l_1-norm perturbation size of 0.5.

As seen in Fig .6.24 because of the closeness of the benign and malicious flows, small amounts of perturbation can cause a misclassification.

6.3.2 USTC: CNN-0 model

We give the results for the USTC Sparse l_1 Descent Attack and the Madry et al.'s PGD white-box attacks here. Again, we begin with the attack that attempts to transform malicious flow traffic into appearing benign.

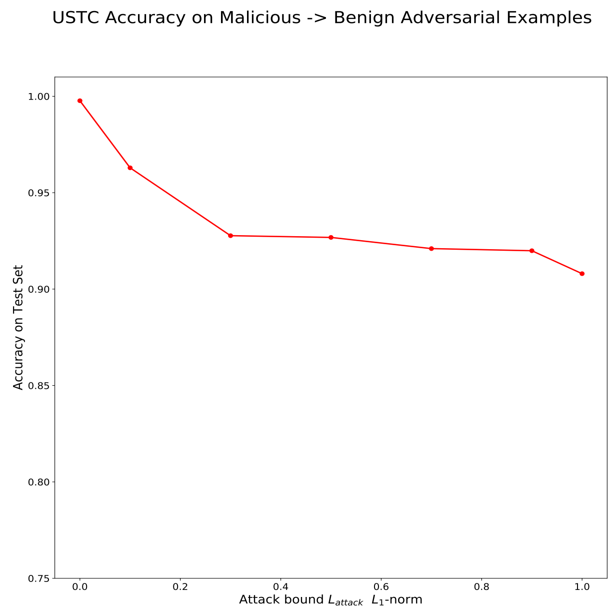


Figure 6.25: Accuracy of UNSW CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}.

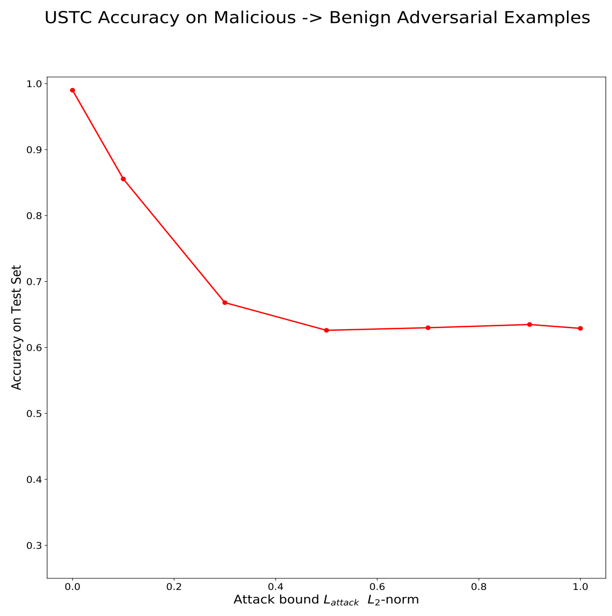


Figure 6.26: Accuracy of UNSW CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

As seen in Fig. 6.25, the additional information did not lead to a massive decrease in accuracy as expected for the l_1-norm attack. The decrease that we observed was comparable to that of the black-box attack. It is not initially clear why the decrease was so minimal. However, for the l_2-norm attack we do see the expected results. With the additional information of the model, this attack is able to transform upwards of 30% of the malicious flows into appearing benign.

In moving to attacks that transform benign flows into appearing malicious, we see more of the expected results.

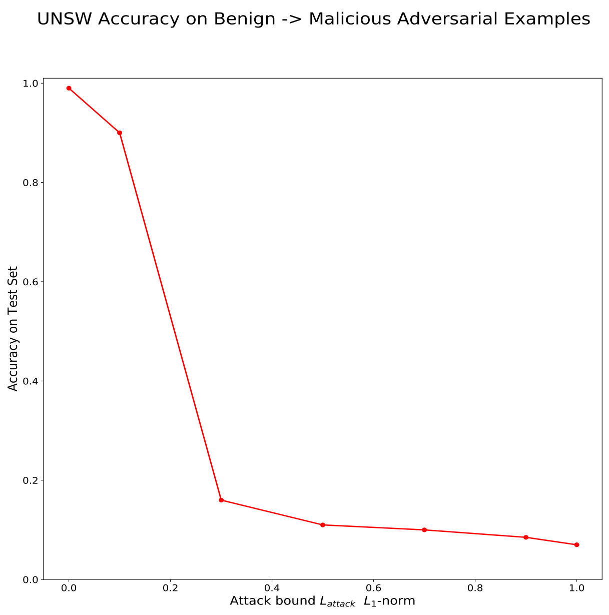


Figure 6.27: Accuracy of USTC CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}.

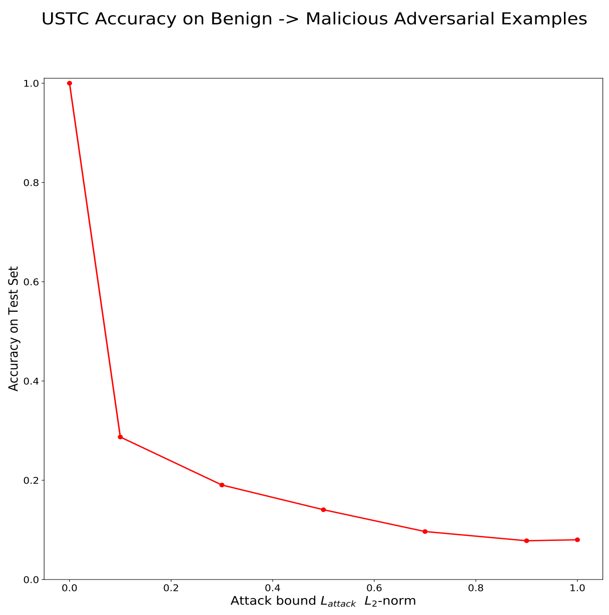


Figure 6.28: Accuracy of USTC CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

In Fig. 6.27, we see an almost immediate drop off in the accuracy of the accuracy. With a maximum attack vector size of 0.1 the accuracy decreases to 90% and at a maximum attack vector of 0.3, it then decreases again to being only 18%. This again shows the effectiveness of an attack meant to craft a heavy torrent of ’fake’ malicious traffic in order to overwhelm a given system. Looking at the l_2-norm attack in Fig. 6.28, we see a similar huge drop-off in the accuracy of the system. Even with an attack bound of just 0.1, the accuracy of model on the malicious test set drops to just under 30%.

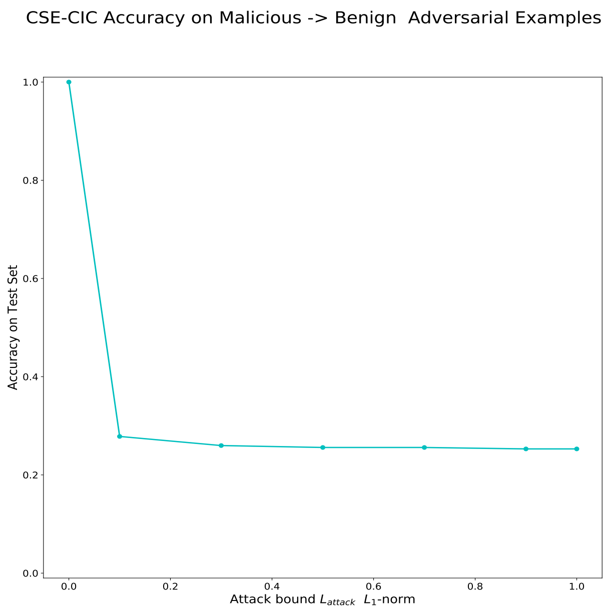


Figure 6.29: Accuracy of CSE-CIC CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}

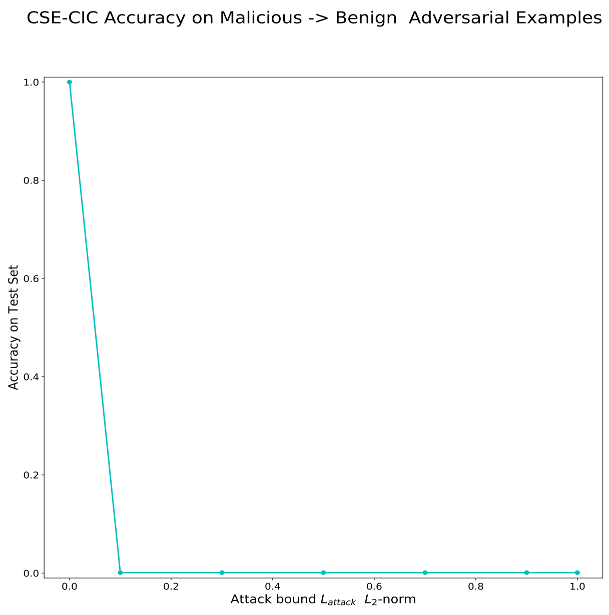


Figure 6.30: Accuracy of CSE-CIC CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

6.3.3 CSE-CIC: CNN-0 model

We give the results for the CSE-CIC Sparse l_1 Descent Attack and the Madry et al.'s PGD white-box attacks here. Again, we begin with the attack that attempts to transform malicious flow traffic into appearing benign.

As seen in Fig. 6.29, these types are enormously effective. The l_1-norm white-box attack managed to get the accuracy of the model to only 25% even with an attack vector $L_{attack} = 0.1$. This exhibits the large advantage that knowledge of a system can potentially has in attacking it. Similarly, for the l_2-attack in Fig. 6.30, we again see a major decrease in accuracy, nearing 0. This serves to show how an adversary with the appropriate knowledge can attack this system and can create adversarial examples with near perfect accuracy, thus defeating the system. We now move on to the reverse attack, namely transforming benign traffic into appearing malicious. In Figs. 6.31 and

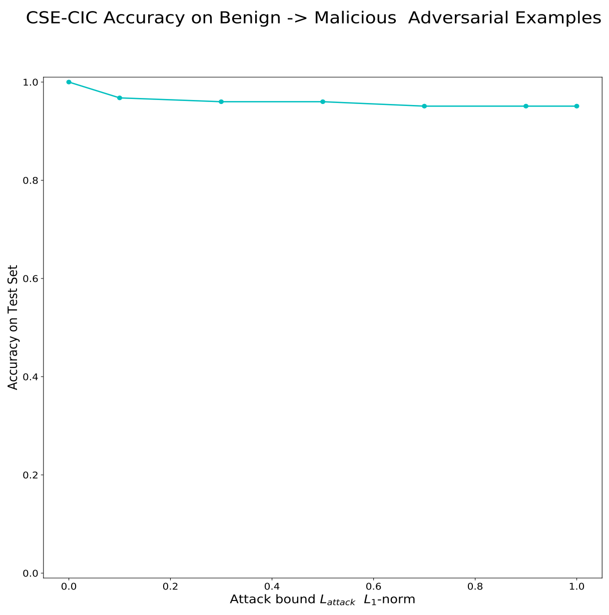


Figure 6.31: Accuracy of CSE-CIC CNN-0 under Sparse l_1-norm Descent white-box attacks of varying maximum-size L_{attack}.

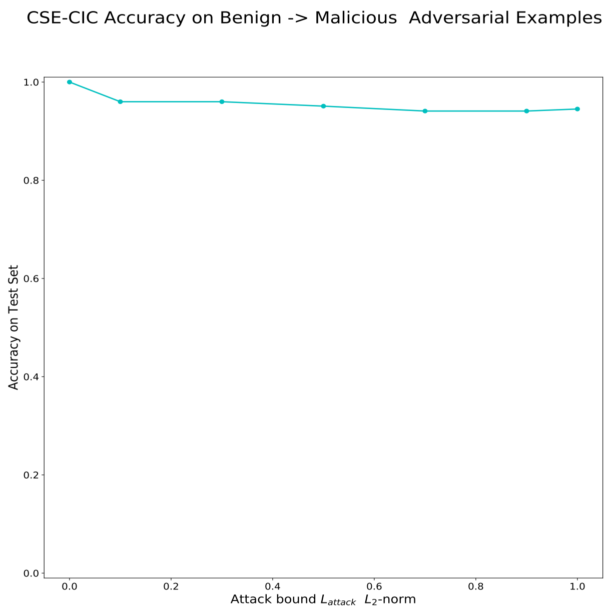


Figure 6.32: Accuracy of CSE-CIC CNN-0 under Madry et al. l_2-norm white-box attacks of varying maximum-size L_{attack}.

6.32, we do not see any significant improvement of the effectiveness of the white-box attack compared to the black-box attacks. The accuracy of the model to this type of attack remains near 97% for both types of attacks considered here. This shows that while white-box attack that transform benign flow to malicious flow are possible, this particular network is fairly robust to them at the levels of attack considered. Again, this is nearly the opposite of the type of behaviour that we saw for the other datasets. See the Discussion chapter for our thoughts on this discrepancy.

6.4 Targeted White-Box Attacks

After performing the baseline black-box and white-box attacks presented in Sections 6.2 and 6.3, we wished to understand what these adversarial attacks translated to in terms of the feature space that we used for classification. We further wished to ascertain whether we could perform targeted attacks where we changed only one or two of the features (i.e. number of packets or bytes/s). We thus performed targeted attacks on our networks where we used CleverHans to perform adversarial attacks whilst only changing one or two pre-selected features. We acknowledge that in order to perform these targeted attacks that we would need to be in a white-box scenario. Performing these attacks in a black-box scenario does not seem feasible.

In order to first understand what features that we should target in our adversarial attack, we first wanted to get an approximation of which features were the most important for classification. In order to do this, we first classified our datasets as either malicious or benign by using a random forest machine learning algorithm. For more details about random forests see our Background chapter. Once we had classified the datasets, we then extracted the indices of the most important features used in the classification.

The way that feature importance is calculated is based on the information gain (decrease in entropy). When using decision tree each tree node is treated as a condition of how to split a given training instance on a single feature. This split is done so that similar training instances are split in the same way after the node. For random trees, specifically used for classification, the split is determined based on entropy/the amount of information that is gained based solely on the feature. These features can then be ranked in terms of the amount of entropy decrease caused by splitting on that feature. Thus, this functionality can be used to gain a rough understanding of which features are the most important for classification purposes [17].

Once these features were determined for both the UNSW, USTC, and CSE-CIC datasets, we then ran targeted attacked against the CNN-0 dataset where we attacked these particular features. For all of the attacks described below we capped our max attack vector size $L_{attack} = 0.1$.

6.4.1 UNSW Targeted Attacks

For UNSW dataset some of the most important features used for classification were the (1) dttl (destination to time to live value), (2) the ct-state-ttl (ttl information), (3) the sttl (source to destination time to live), (4) the tcprtt (the TCP connection setup round trip time), (7) and dload (destination bits per second).

Once we determined that these were the most important features used for classification, we then ran a Sparse l_1 Descent white-box attack, changing only one or two features at a given time. We now list a subset of these attacks here:

1. We found that for a Generic malware that increasing the destination to source packet count by 76 caused it to be reclassified as benign.

2. We found that for an Exploit malware that decreasing the total destination to source packet count by 800 caused it to be reclassified as benign.

3. We found that for an Analysis malware that increasing the destination to source bits per second by 2.134Kbits caused it to be reclassified as benign.

As seen in this quick enumeration, somewhat counter-intuitive single changes can cause malware to be misclassified as benign. Given that the above changes would not be massive alterations to these malware and that they could be easily accomplished, this is enormously troubling. By knowing the inner-workings of the given CNN, it is thus possible to craft malware that is functionally equivalent to other malware variants but that is able to evade detection.

6.4.2 USTC Targeted Attacks

For the USTC dataset, some of the most important features used for classification were the (1) the total flow duration, (2) the minimum number of PSH-flagged packets in a given flow, (3) the average flow duration multiple flows using the same IPs and Ports, (6) the average number of packets with the ACK flag in a given flow, (8), the average number of packets sent by the server that have the ACK flag in a given flows.

For the USTC dataset, once we determined that these were the most important features used for classification, we again ran a Sparse l_1 Descent white-box attack, changing only one or two features at a given time. We list a subset of these attacks here:

1. We found that for a Cridex malware flow that decreasing the flow duration by 2 seconds and decreasing the average flow duration by 1.8 seconds caused it to be reclassified as benign.
2. We found for a group of Cridex flows (same IPs and ports) increasing the minimum flow duration by 2 seconds caused it to be reclassified as benign.
3. We found for a group of Zeus malware flows (same IPs and ports) that increasing the total flowset duration by .27 second and decreasing the average flow duration by 2.35 seconds caused it to be reclassified as benign.
4. We found for a group of Hbot malware flow that increasing the standard deviation of the packet lengths by 0.09 bytes caused the flow to be reclassified as benign.

The above examples further show the efficacy of performing these types of targeted attacks. Although timing information is more difficult to control, by sending multiple flows that try to change the flow parameters as above, an adversary can eventually become successful in evading detection.

6.4.3 CSE-CIC Targeted Attacks

For CSE-CIC dataset some of the most important features used for classification were the (1) tot-fw-pk (the total number of forward packets), (2) tot-bw-pk (the total number of backward packets), (3) bwd-iat-min (the minimum inter-arrival time for backward packets), (4) bwd-iat-mean (the average inter-arrival time for backward packets), (7) and fw-pkt-l-std (the standard deviation of the length of forward packets sent).

1. We found that for a Botnet malware flow that increasing the total number forward bytes by 11005 bytes causes the flow to be reclassified as benign.

2. We found that for a Botnet malware flow that increasing the total number forward bytes by 5503 bytes causes the flow to be reclassified as benign.

3. We found that for a Botnet malware flow that increasing average length of the forward flows by 65.7 bytes causes the flow to be reclassified as benign.

4. We found that for a Botnet malware flow that increasing average length of the forward flows by 44 bytes causes the flow to be reclassified as benign.

The above examples that we gleaned from targeted attacks on the CSE-CIC dataset exhibits the practicality of these sorts of attacks on this particular network. These types of attack would be fairly easy to carry out by padding packets sent or sending more packets within a given flow. This thus further confirms the vulnerability of these networks to adversarial attacks. As we now have shown, the networks that we have presented are fairly vulnerable to adversarial attacks, both in white-box and black-box settings. Given enough time and resources, an adversary can craft malware that could easily evade detection. Similarly, they could also cascade a network with benign traffic that appears to be malicious.

Chapter 7

Defense: Experiments and Results

Now that we have given an overview of the adversarial attacks that can be conducted against our networks, we now move to the set of our defences that can be used against these types of attacks. Note that in this chapter, we do not present all graphs that we received from our experiments due to space constraints. In order to view these figures please see our anonymous GitHub (https://github.com/DPSelectro/DPNetwork). For these defences we focus on networks that differentiated malicious from benign software as that is the main focus on this work. We again further limit our discussion to the CNN-0 neural network as that gave us the best results for the USTC dataset. For the CNN-0 neural network, we measure their effectiveness for the UNSW, USTC, and CSE-CIC dataset. The specific defense experiments that we ran were: (1) robustness measurements for l_1-attacks using Laplace noise after the 1st layer of network, (2) robustness measurements for l_1-attacks using Gaussian noise after the 1st layer of network, (3) robustness measurements for l_2-attacks using Gaussian noise directly on features. In addition, we also performed Poisson subsampling for the USTC dataset (with all three types of noise placements) in order to ascertain whether this was a possible step forward in improving the accuracy and defense trade-offs.

For each model we focused explicitly on attacks that transformed malicious flows into appearing benign as these present the most compelling use of our system. For these results, we used a $\eta = 0.95$ where η is the probability that the bounds that we found hold. Note that η can be arbitrarily small by performing more predictions. For our work, we performed 40 predictions for each label. Recall that unlike in training, for predictions, we draw fresh noise each time that make a prediction. Note that for a fixed η value, the thresholds to which examples are robust can also be improved by performing more predictions. See our Background chapter for more details on how this prediction and η determination work.

7.1 DP Defense: Laplace Noise After 1st Layer of Autoencoder

In this section, we focus on using Laplace noise after the first layer of an autoencoder in order to protect against l_1-norm attacks. Note that Laplace noise incorporate $(\epsilon, 0)$ -differential privacy into our autoencoder. For these experiments we chose $\epsilon = 1$. This replicates the value chosen by Lecuyer et al. [47].

For the Laplace noise trained models, we ascertained their robustness to adversarial l_1-norm attacks. For this work, again we considered attack vectors that ranged in size from 0.1 to 1.0.

7.1.1 UNSW CNN-0 Differential Privacy Laplace Layer-1 Protection

We now present the results for our differential privacy approach for the UNSW dataset CNN-0 model. We first showcase the accuracy and. robustness of the model for labelling malicious flows using multiple predictions.

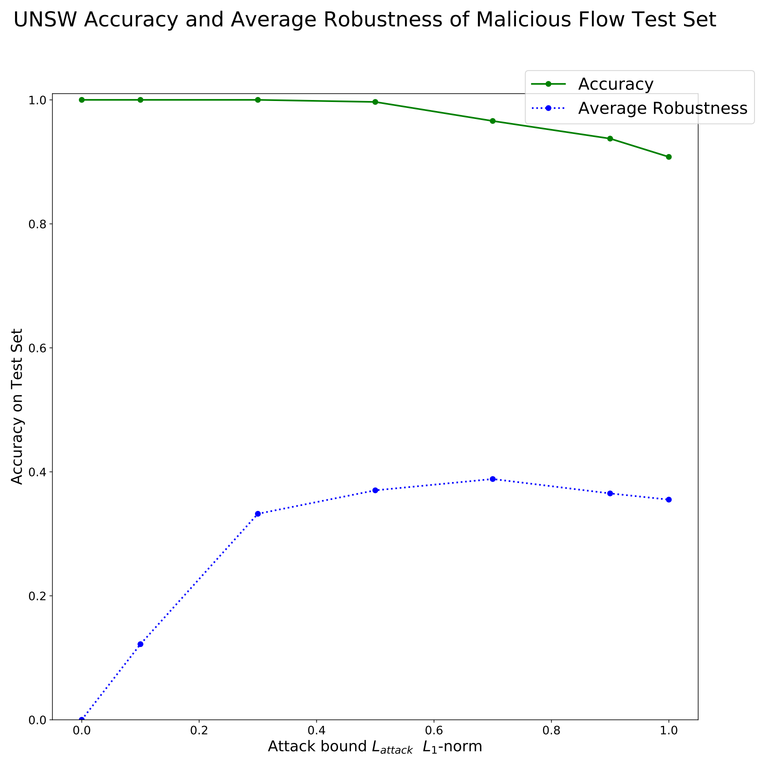


Figure 7.1: Laplace Layer1: Graph of UNSW Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

As seen in Fig. 7.1, the accuracy of the model remains relatively high due to the number of predictions that are made. Furthermore, the robustness of the model to adversarial examples generally follows an upward and linear relationship with the attack bound until starting to level off after 0.3. This initial results evidences a good trade-off for robustness to adversarial examples vs. accuracy. The model is able to maintain nearly perfect accuracy on the dataset until incorporating larger noise for an attack

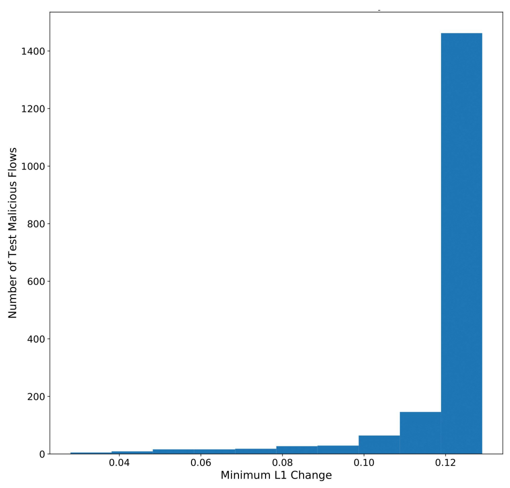


Figure 7.2: Laplace Layer1: Robustness Distribution for UNSW dataset after training with noise for an l_1-norm attack vector of size 0.1.

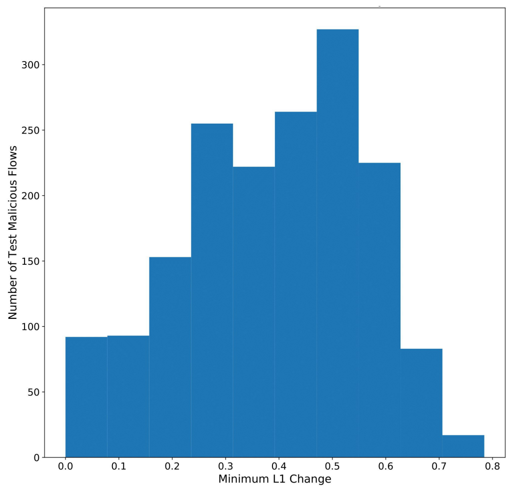


Figure 7.3: Laplace Layer1: Robustness Distribution for UNSW dataset after training with noise for an l_1-norm attack vector of size 0.7.

vector of size $L_{attack} = 0.5$. Furthermore, after attack vector size $L_{attack} = 0.7$, there is lesser returns in terms of robustness, so higher defense values could not be recommended anyway.

We finally evidence a subset of the graphs that showcase the distribution of robustness for varying attack/noise-training levels.

As seen in Figs 7.2 to 7.3 for smaller attack vector training sizes, the robustness of the model is very tail heavy. As the size of the model attack vector increases however, this begins to shift (becoming almost Gaussian).

This completes our presentation of results for the UNSW dataset defense for Laplace noise placed after the first layer of autoencoder under attack from $l - 1$-norm attacks.

7.1.2 USTC CNN-0 Differential Privacy Laplace Layer-1 Protection

We now present the results for our differential privacy approach for the USTC dataset CNN-0 model. We again first showcase the accuracy and. robustness of the model for malicious flows using multiple predictions.

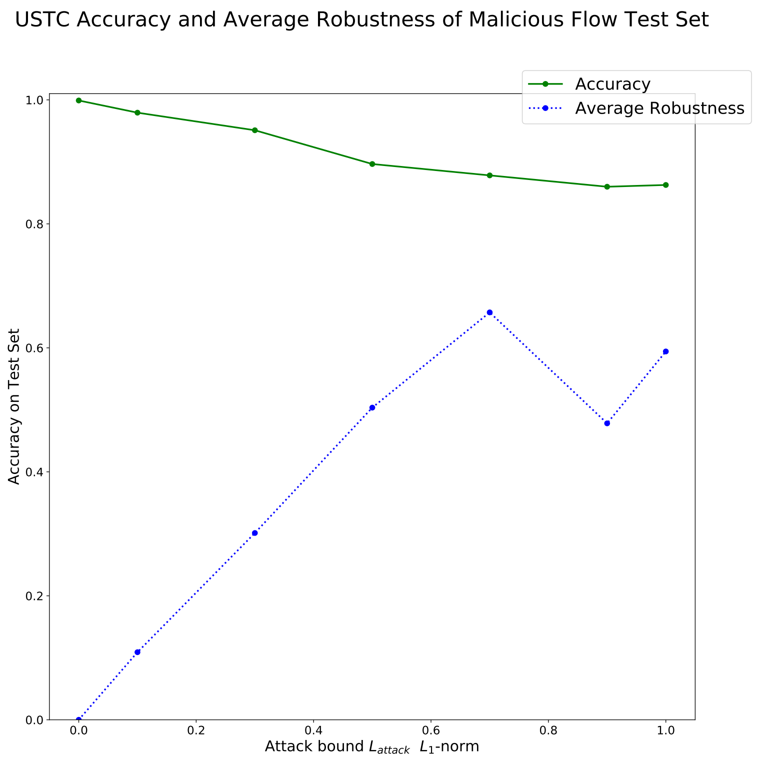


Figure 7.4: Laplace Layer1: Graph of USTC Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

As seen in Fig. 7.1, the accuracy of the model degrades slightly more than for the UNSW dataset. The accuracy almost immediately moves towards being around 90%. However, in a likewise fashion, the average robustness is able to reach higher values in this case as well. At an attack vector size $L_{attack} = 0.7$, the average robustness reaches a level of 0.657. In fact, the average robustness does not begin to degrade until the attack-vector/training size vector $L_{attack} > 0.7$. Like for the UNSW dataset, this initial results

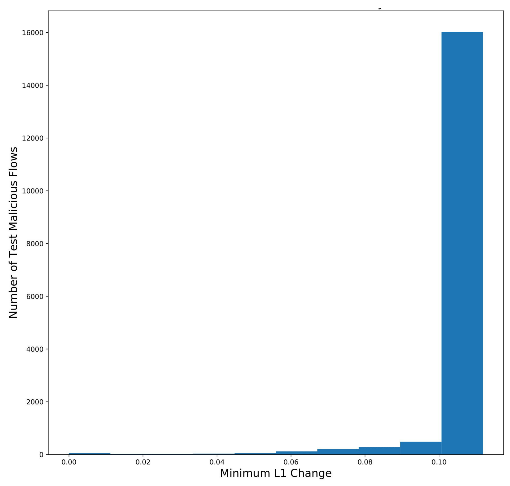


Figure 7.5: Laplace Layer1: Robustness Distribution for USTC dataset after training with noise for an l_1-norm attack vector of size 0.1.

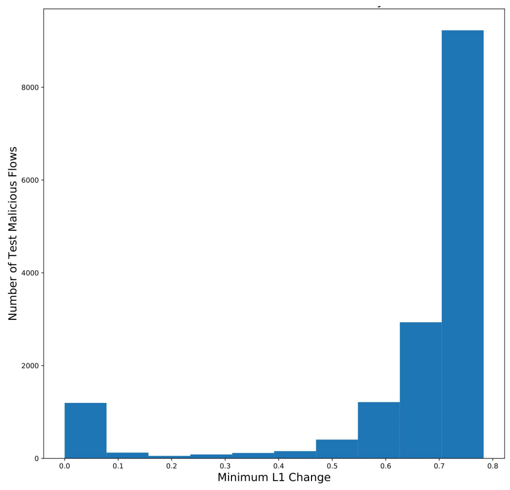


Figure 7.6: Laplace Layer1: Robustness Distribution for USTC dataset after training with noise for an l_1-norm attack vector of size. 0.7.

further evidences a good trade-off for robustness to adversarial examples vs. accuracy. For example, all of the targeted adversarial attacks that were presented in section 6.4 would all provably fail when using this defense with L=0.1 (where accuracy of the model on malicious samples is upwards of 98%). We finally evidence a subset of the graphs that showcase the distribution of robustness for varying attack/noise-training levels.

As seen in Figs 7.5-7.6, there remains a dichotomy in this particular calculation of robustness. Either the examples that are found are extremely robust to adversarial examples or they are very much vulnerable to attack. Only at an attack vector size $L_{attack} = 0.7$ do we see the robustness levels begin to spread out. This accords with the average values that we saw in Fig 7.4. This completes our presentation of results for the USTC dataset defense for Laplace noise placed after the first layer of autoencoder under attack from $l - 1$-norm attacks.

7.1.3 CSE-CIC CNN-0 Differential Privacy Laplace Layer-1 Protection

We now present the results for our differential privacy approach for the CSE-CIC dataset CNN-0 model. We first showcase the accuracy of the model on malicious flows using multiple predictions. As seen in Fig. 7.7, the accuracy of the model remains near

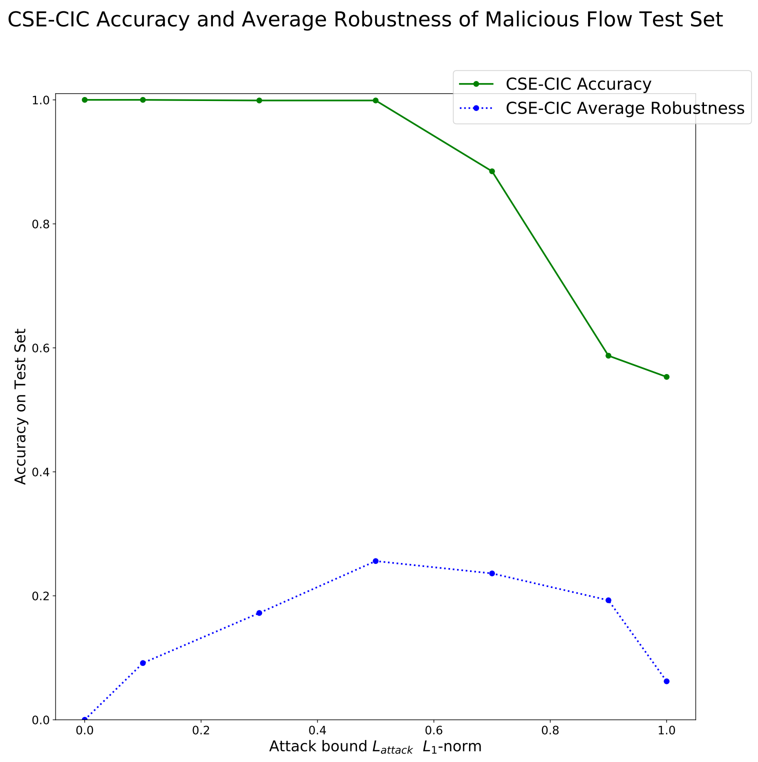


Figure 7.7: Laplace Layer1: Graph of CSE-CIC Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

perfect for small values of L_{attack}. Only when L_{attack} is above $L_{attack} \geq 0.7$ does the accuracy begin to degrade significantly. The robustness of the model also begins to degrade at this value as well (so there is perhaps no reason to increase the training vector beyond this value anyway). As with the two previous datasets, we see a very good trade-off for small values of L_{attack} illustrating the benefits of this approach. We

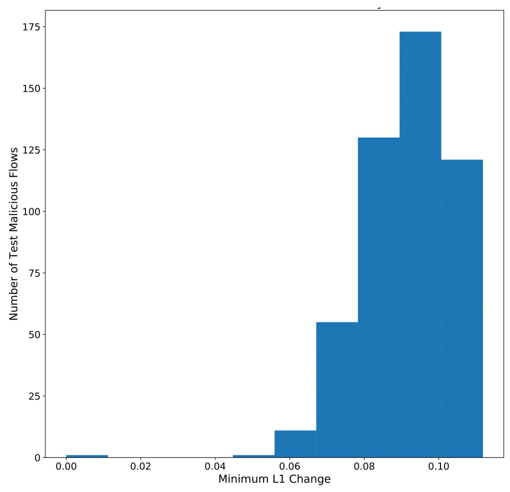


Figure 7.8: Laplace Layer1: Robustness Distribution for CSE-CIC dataset after training with noise for an l_1-norm attack vector of size 0.1.

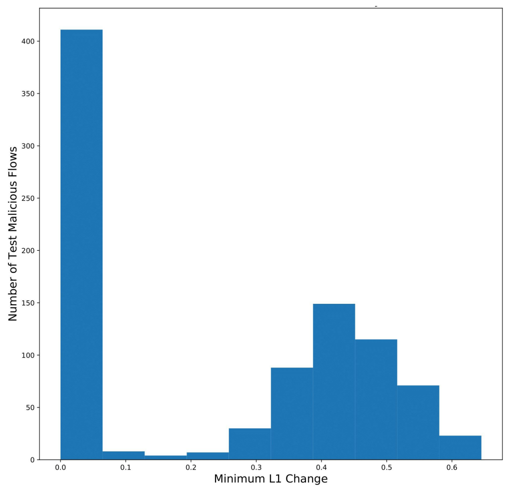


Figure 7.9: Laplace Layer1: Robustness Distribution for CSE-CIC dataset after training with noise for an l_1-norm attack vector of size 0.7.

finally present a subset of the graphs that showcase the distribution of robustness for varying attack/noise-training levels.

As seen in Figs 7.8-7.9, there remains a dichotomy in this particular calculation of robustness. However, in this dataset, there are much more flows that retain a fairly small robustness value.

This concludes our discussion of Laplace noise after the first layer of an autoencoder in order to protect against l_1-norm attacks.

7.2 DP Defense: Gaussian Noise After 1st Layer of Autoencoder

In this section, we focus on using Gaussian noise after the first layer of an autoencoder in order to protect against l_1-norm attacks. Note that Gaussian noise incorporation leads to (ϵ, δ) -differential privacy in our autoencoder. For these experiments we chose

$\epsilon = 1$ and $\delta = 0.05$. This replicates the values chosen by Lecuyer et al. [47]. However, by using Gaussian noise, we do not lose out on any of the differential privacy properties upon which we are dependent. See our Background chapter for more details

For the Gaussian noise trained models, we ascertained their robustness to adversarial l_1-norm attacks. For this work, again we considered attack vectors L_{attack} that ranged in size from 0.1 to 1.0.

7.2.1 UNSW CNN-0 Differential Privacy Gaussian Layer-1 Protection

We now present the results for our differential privacy approach for the UNSW dataset CNN-0 model. We first present results that give the trade-off between model robustness to adversarial examples and the accuracy on malicious flows.

As seen in Fig. 7.10, the results are marginally worse than for the Laplace case in Section 7.1. We see in Fig. 7.10 that the robustness falls significantly after the attack vector size $L_{attack} = 0.5$. In fact, the robustness level bottoms out when the attack vector size $L_{attack} = 0.9$. We thus see from this graph that in general, while we still get reasonable results for $L \leq 0.5$, that using Gaussian noise cannot be recommended over using Laplace noise for these types of attacks.

We finally provide a subset of the graphs that showcase the distribution of robustness for varying attack/noise-training levels.

As seen in Figs. 7.11-7.12, there is a shift towards low amounts of robustness at an attack vector size $L_{attack} = 0.7$. This accords with our average robustness size seem in Fig. 7.10. This further confirms that Gaussian noise for higher l_1-attack vectors sizes does not result in better results than using Laplace noise.

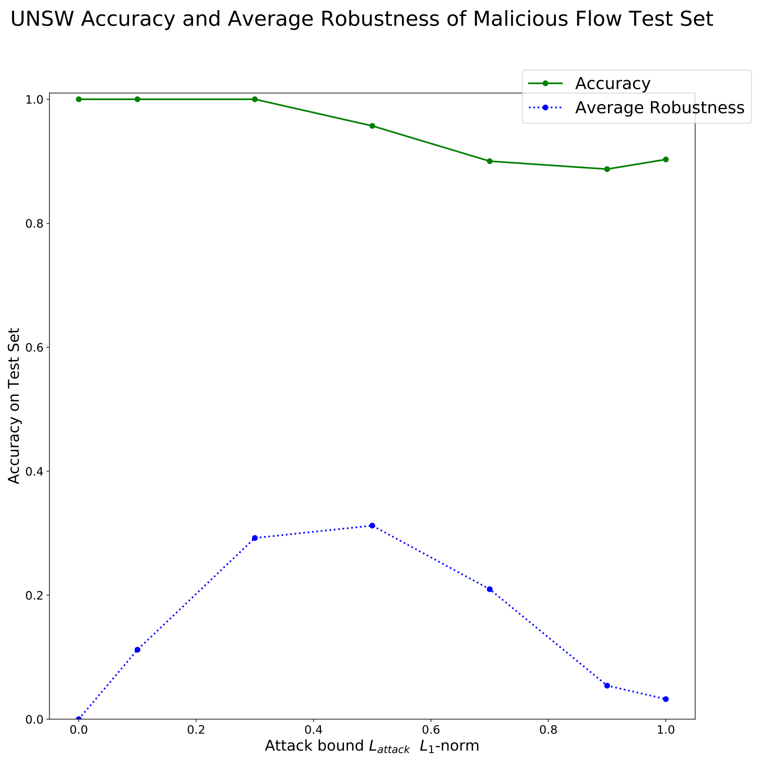


Figure 7.10: Gaussian Layer1: Graph of UNSW Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

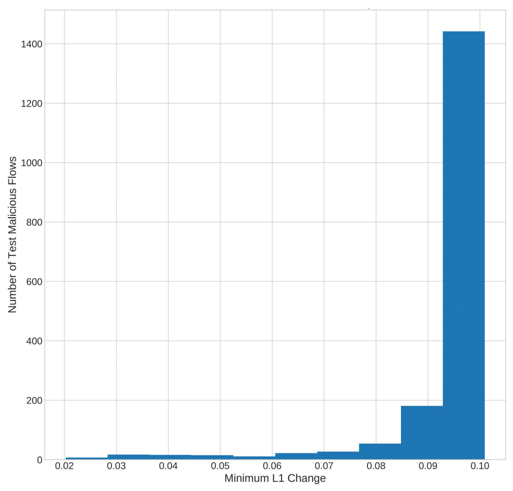


Figure 7.11: Gaussian Layer1: Robustness Distribution for the UNSW dataset after training with noise for an l_1-norm attack vector of size 0.1.

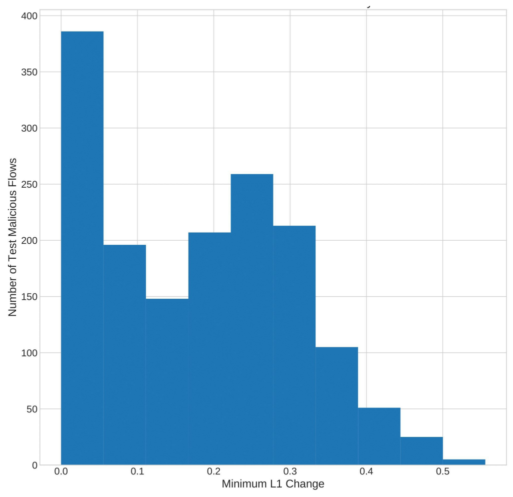


Figure 7.12: Gaussian Layer1: Robustness Distribution for the UNSW dataset after training with noise for an l_1-norm attack vector of size 0.7.

7.2.2 USTC CNN-0 Differential Privacy Gaussian Layer-1 Protection

We now present the results for our differential privacy approach for the USTC dataset CNN-0 model. We now again present the picture of the trade-off between robustness and accuracy.

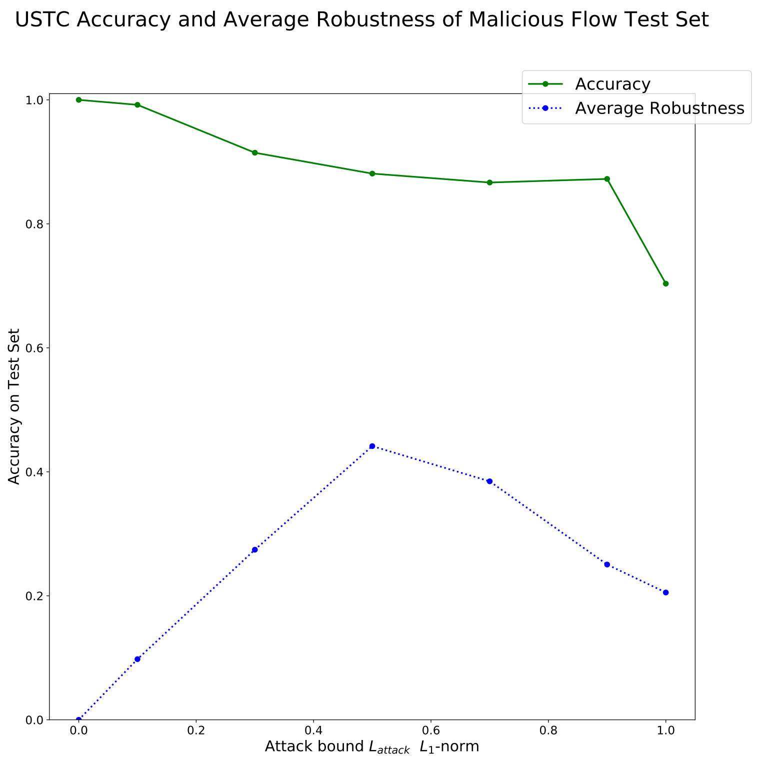


Figure 7.13: Gaussian Layer1: Graph of USTC Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

As seen in Fig. 7.13, we do not see the dramatic drop-off in robustness as we saw for the UNSW dataset. However, despite this we again see that for these l_1-norm attacks that the Gaussian noise does not perform as well. We see an average robustness that reaches a maximum value of 0.44 when the training/attack vector size $L_{attack} = 0.5$. This however is significantly less than the maximum of 0.657 average robustness that we

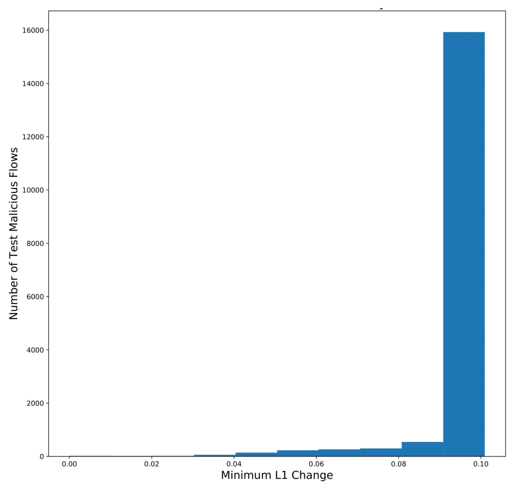


Figure 7.14: Gaussian Layer1: Robustness Distribution for the USTC dataset after training with noise for an l_1-norm attack vector of size 0.1.

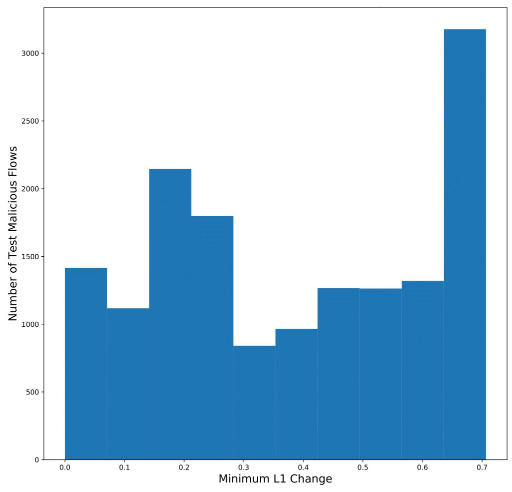


Figure 7.15: Gaussian Layer1: Robustness Distribution for the USTC dataset after training with noise for an l_1-norm attack vector of size 0.7.

achieved with the Laplace noise. Furthermore, at a training/attack size vector $L = 1.0$, we do see a dramatic drop-off in overall accuracy to 70.2%. As such, we confirm that the Laplace noise gave us better results for both the UNSW and the USTC dataset.

We finally give a subset of robustness distributions for some of the attack vector sizes.

As before in Figs. 7.14-7.15, we see the normal shift towards the center as the size of the attack vector increases.

7.2.3 CSE-CIC CNN-0 Differential Privacy Gaussian Layer-1 Protection

We now present the results for our differential privacy approach for the CSE-CIC dataset CNN-0 model. We first turn to see the trade-off between the accuracy and robustness of the model.

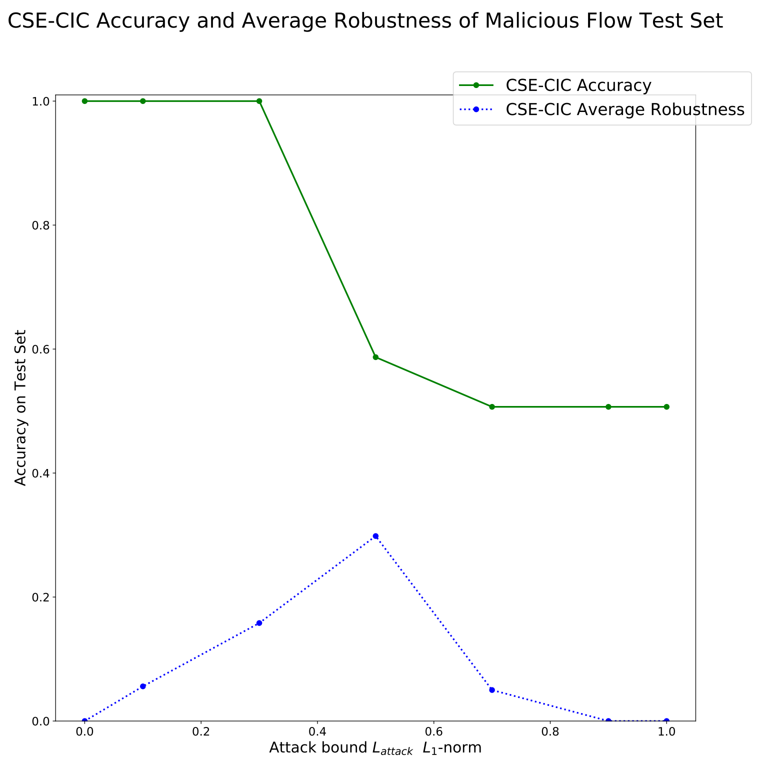


Figure 7.16: Gaussian Layer1: Graph of CSE-CIC Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples using

In Fig. 7.16 we see a much quicker drop-off for the accuracy than in the Laplace case (see Fig. 7.7. However, for $L_{attack} \leq 0.3$, we do see a good trade-off between the accuracy and the robustness of the model.

We finally showcase a subset of the graphs that showcase the distribution of robustness for varying attack/noise-training levels.

As seen in Figs 7.17-7.18, there remains a dichotomy in this particular calculation of robustness for $L_{attack} \leq 0.7$. However here we also see that for $L_{attack} = 0.7$ that the distribution actually spreads out completely becoming almost Gaussian. We are not entirely sure what caused this change in behaviour. We leave determining the reasons behind given robustness distributions to Future Work.

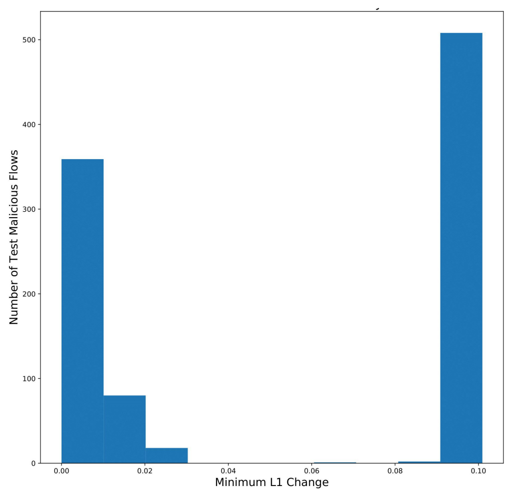


Figure 7.17: Gaussian Layer1: Robustness Distribution for CSE-CIC dataset after training with noise for an l_1-norm attack vector of size 0.1.

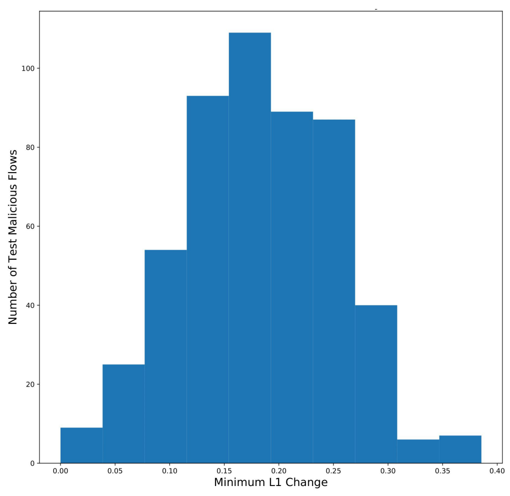


Figure 7.18: Gaussian Layer1: Robustness Distribution for CSE-CIC dataset after training with noise for an l_1-norm attack vector of size 0.7.

This completes our analysis of Gaussian noise after the first layer of an autoencoder in order to protect against l_1-norm attacks.

7.3 DP Defense: Gaussian Noise Directly on Features

In this section, we focus on using Gaussian noise directly on input to autoencoder in order to protect against l_2-norm attacks. Note that using our formulation Laplace noise cannot be used to protect our models from l_2-norm attacks (this has to do with how the sensitivity and norms are computed. For details see the Background section about sensitivity). As a result of this, to protect against l_2-norm attacks we focus only on the approach discussed here. For these experiments for our differential privacy parameters we again chose $\epsilon = 1$ and $\delta = 0.05$.

7.3.1 UNSW CNN-0 Differential Privacy Gaussian Noise Directly on Features Protection

We now present the results for our differential privacy approach for the UNSW dataset CNN-0 model. We first look at the relationship between the robustness and accuracy.

As seen in Fig 7.19, the robustness of the model to adversarial examples never goes above 0.23. Despite this, the accuracy of the model likewise does not decrease significantly until the size of the attacking vector $L_{attack} = 0.5$. Furthermore, for attack vector values $L_{attack} < 0.3$, there is a reasonable trade-off between the overall accuracy on malicious flows and the robustness of the model.

We again highlight a subset of the distributions of the robustness under varying attack vector sizes.

As seen in Figs. 7.20-7.21, the distribution of robustness switches orientation after an attack vector size $L_{attack} = 0.5$. Furthermore, we found that $L_{attack} > 0.5$, the results show that we are unable to get single values that are highly robust as well. As such, for training this particular model, training for an attack vector $L_{attack} > 0.5$ could not be

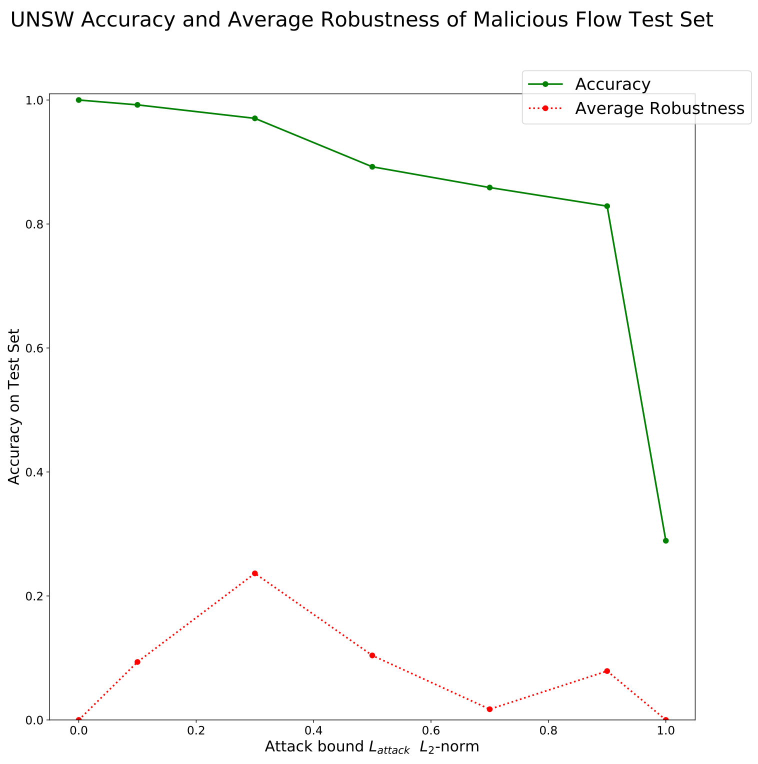


Figure 7.19: Gaussian Direct: Graph of UNSW accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples

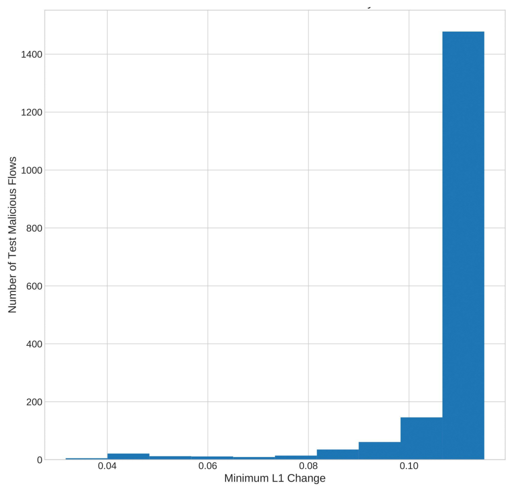


Figure 7.20: Gaussian Direct: Robustness Distribution for UNSW after training with noise for an l_1-norm attack vector of size 0.1.

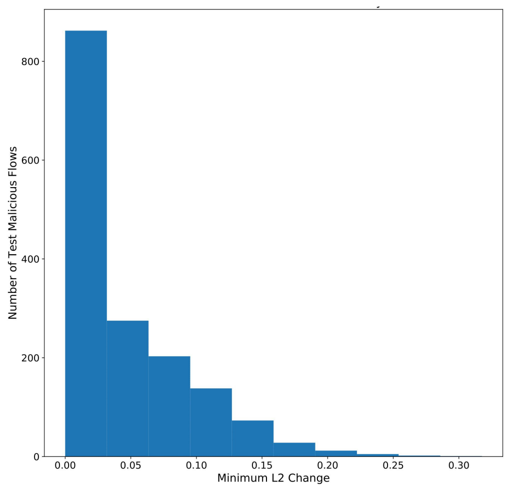


Figure 7.21: Gaussian Direct: Robustness Distribution for UNSW after training with noise for an l_1-norm attack vector of size 0.5.

recommended both due to decreases in overall robustness but also due to the decreases in accuracy.

7.3.2 USTC CNN-0 Differential Privacy Gaussian Noise Directly on Features Protection

We now present the results for our differential privacy approach for the USTC dataset CNN-0 model. We first showcase the trade-off between prediction accuracy (using multiple predictions) and robustness.

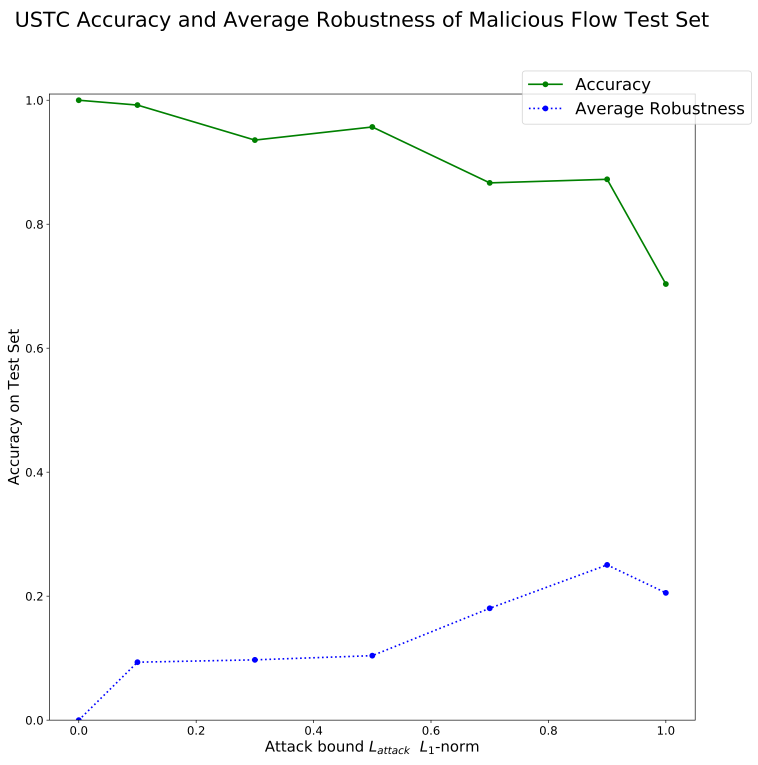


Figure 7.22: Gaussian Direct: Graph of USTC accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

Unlike for the UNSW dataset, the average robustness continues to increase as seen in Fig 7.22. However, for larger robustness values there is a somewhat significant decrease in accuracy on the malicious flows. For example, in order to achieve a robustness of

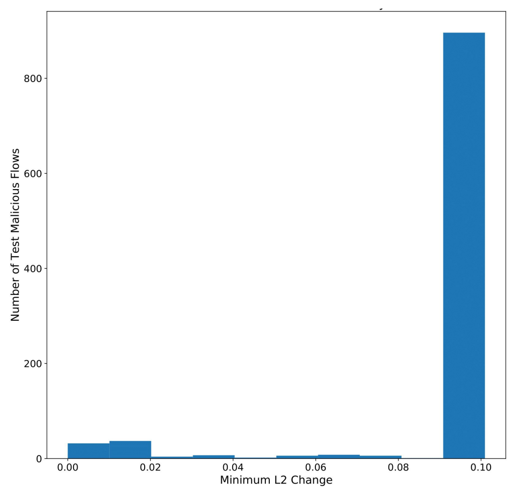


Figure 7.23: Gaussian Direct: Robustness Distribution for USTC after training with noise for an l_1-norm attack vector of size 0.1.

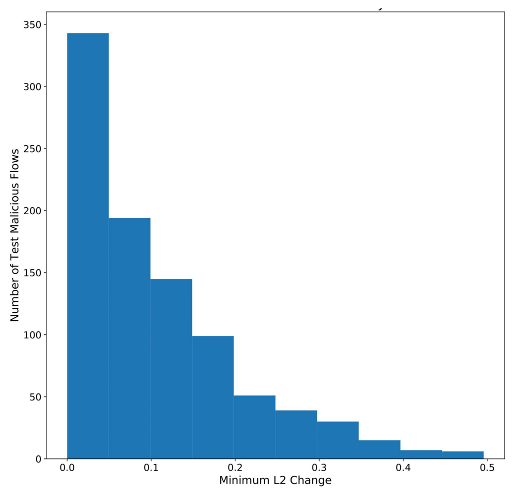


Figure 7.24: Gaussian Direct: Robustness Distribution for USTC after training with noise for an l_1-norm attack vector of size 0.7.

0.25, we only get an accuracy of 0.872. Whether this decrease in accuracy is worth the increase in average robustness is a question that individual particular systems must take into account.

We again highlight a subset of the distributions of the robustness under varying attack vector sizes.

7.3.3 CSE-CIC CNN-0 Differential Privacy Gaussian Noise Directly on Features Protection

We now present the results for our approach for the CSE-CIC dataset CNN-0 model. We now showcase the trade-off between prediction accuracy (using multiple predictions) and robustness. Here we see very lacklustre results for $L_{attack} \geq 0.5$. The robustness goes to 0 as the amount of noise increases. This illustrates that for very large values of L_{attack} that this defense is largely unable to cope. However again for small values of $L_{attack} \leq 0.3$, we again see a good trade-off in terms of robustness and accuracy.

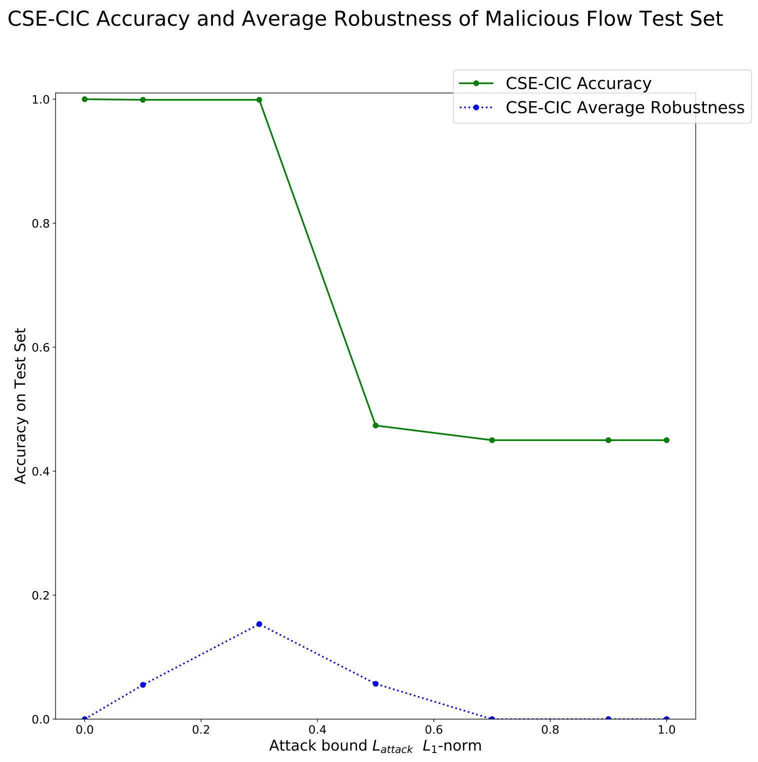


Figure 7.25: Gaussian Direct: Graph of CSE-CIC accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

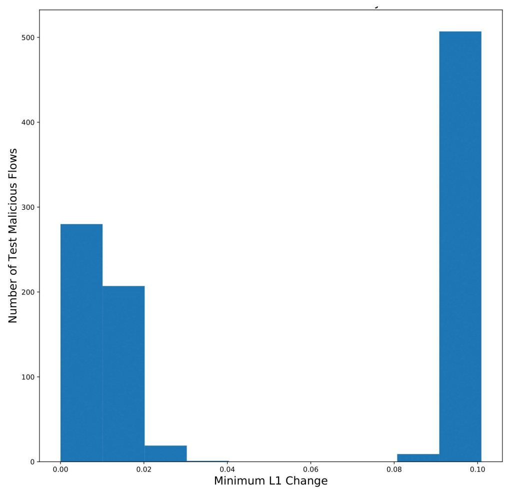


Figure 7.26: Gaussian Direct: Robustness Distribution on CSE-CIC after training with noise for an l_1-norm attack vector of size 0.1.

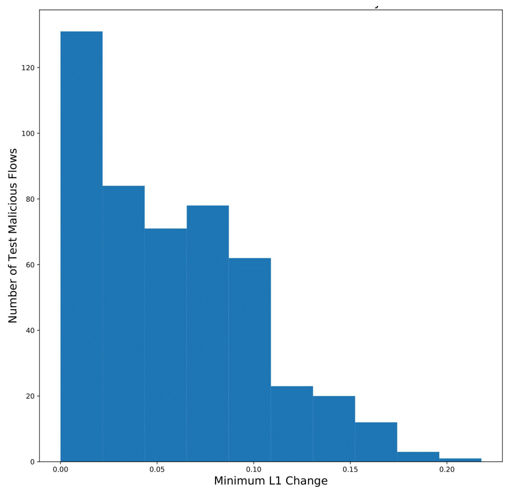


Figure 7.27: Gaussian Direct: Robustness Distribution on CSE-CIC after training with noise for an l_1-norm attack vector of size 0.5.

We again highlight a subset of the distributions of the robustness under varying attack vector sizes.

We again see a clear split again for smaller values L_{attack}, with some flows having very high robustness and other flows having very small amounts.

This concludes our discussion of adding Gaussian noise directly on the features in order to protect against l_2-norm attacks.

7.4 DP Defense: Poisson Subsampling for Privacy Amplification

After performing the defences described in Sections 7.1-7.3, we wanted to determine whether we could get better results using other properties of differential privacy. We found that subsampling in conjunction with noise is a means of incorporating differential privacy into a machine learning algorithm. Knowing this, we decided to subsample a dataset while training our autoencoders. In particular, for this work, we performed Poisson subsampling at a rate of 0.5. This in effect lowers the ϵ and δ values used in differential privacy by half. See our Background chapter for more details. As a result of this, while training our autoencoder, we could in effect lower the amount of noise that is added while training. Because the standard deviation of the noise that we used is linear in the size of ϵ, by subsampling by half, we could in effect halve the noise standard deviation used while maintaining a value of total $\epsilon = 1.0$.

For this part of the work, we performed Poisson subsampling only on the USTC dataset whilst considering all of the attack vectors sizes L_{attack} from 0.1 to 1.0 and noise placements (Laplace and Gaussian noise after the 1st layer of the autoencoder, and Gaussian noise placed directly on the features) that we previously described. We now succinctly describe the results of these experiments.

7.4.1 USTC CNN-0 Differential Private Noise for l_1-adversarial protection

We begin by illustrating some of the baseline results for this Poisson subsampled instance. We firstly show how this noise incorporation affected the trade-off between accuracy and robustness to l_1-norm attacks for both Laplace and Gaussian noise types.

[height=10cm,width=12cm]ch-results-defense/Figures/Poisson-USTC-Accurary-with-Robustness.pdf

Figure 7.28: Graph of USTC Accuracy for both Laplace and Gaussian noise on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples

As seen in Fig. 7.28, we achieve similar result in terms of accuracy for both the Gaussian and the Laplace noise. In general, we see a somewhat more graceful drop-off for the robustness. Accuracy, further, appears to about the same for both the Gaussian and Laplace noise. For the l_1-adversarial attacks using Poisson subsampling, Laplace noise appears to have the best trade-off.

We finally showcase a subset of the graphs of the distribution of robustness for varying attack/noise-training levels for Gaussian and Laplace noise. Overall for these initial results, we see similar results to when we did not use Poisson subsampling for privacy amplification.

We see in Figs. 7.29- 7.32 the distribution of for Laplace noise remains fairly bimodal while the distribution for the Gaussian noise begins to spread out much more quickly.

7.4.2 USTC CNN-0 Differential Private Noise for l_2-adversarial protection

We again start by illustrating some of the baseline results for this Poisson subsampled instance for the l_2 Gaussian noise protected network. We first present the graph of the trade-off between robustness and accuracy.

Here again we see a similar behaviour when compared to when we did not use Poisson subsampling. However, we also see lesser values of robustness for larger attack vector sizes $L_{attack} > 0.5$. This shows that that Poisson subsampling was relatively ineffective in improving the trade-off between robustness and accuracy. We conclude

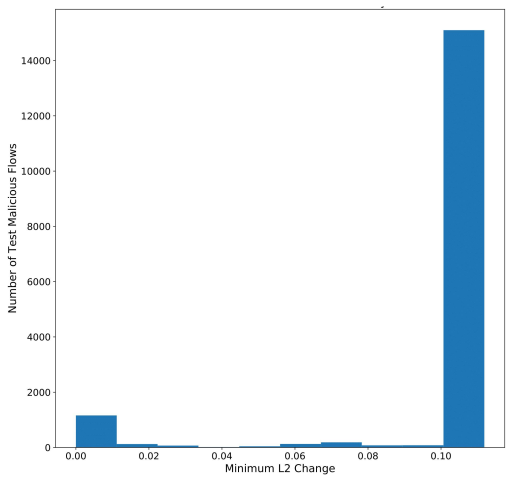


Figure 7.29: Robustness Distribution after training with Laplace noise for an l_1-norm attack vector of size 0.1.

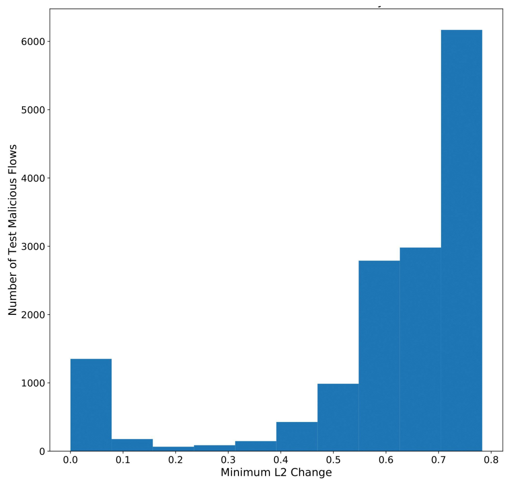


Figure 7.30: Robustness Distribution after training with Laplace noise for an l_1-norm attack vector of size 0.7.

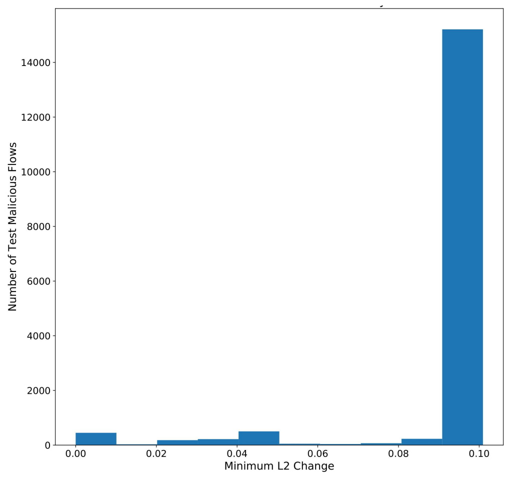


Figure 7.31: Robustness Distribution after training with Gaussian noise for an l_1-norm attack vector of size 0.1.

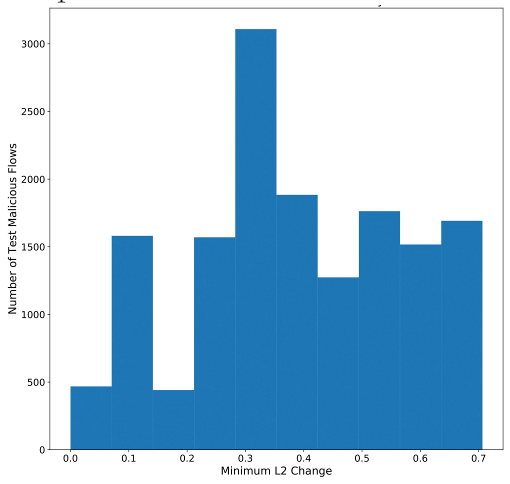


Figure 7.32: Robustness Distribution after training with Gaussian noise for an l_1-norm attack vector of size 0.7.

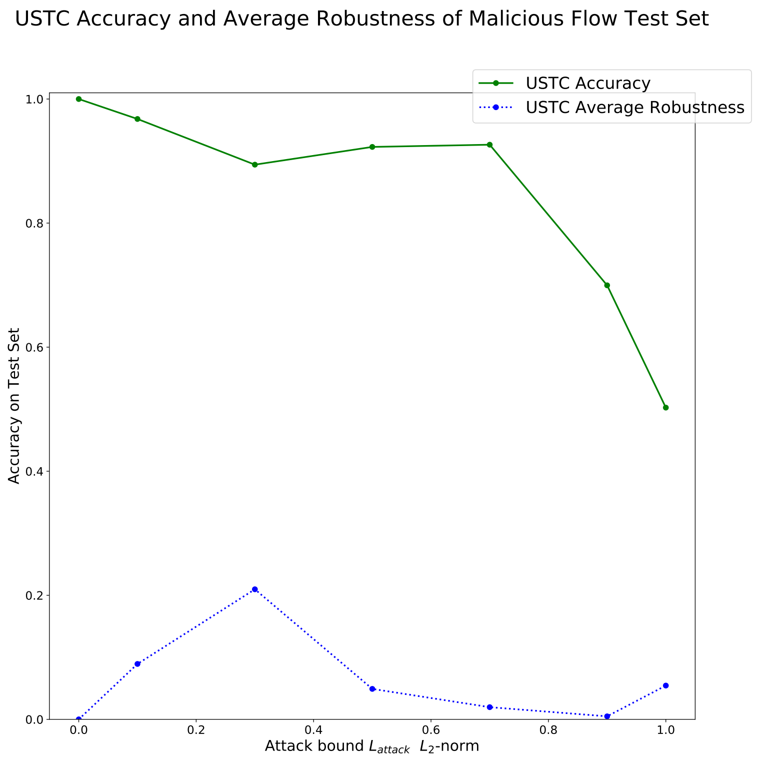


Figure 7.33: Graph of USTC Accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples.

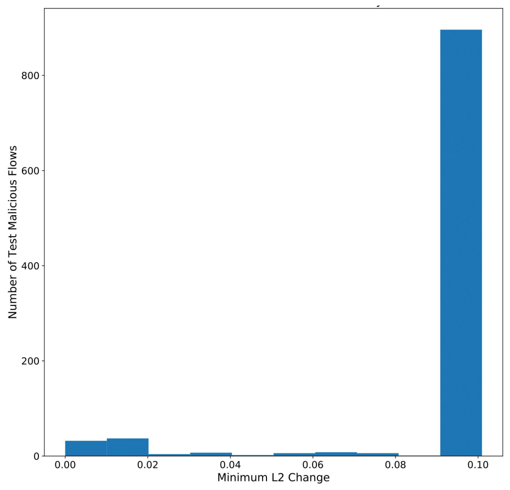


Figure 7.34: Robustness Distribution after training with Gaussian noise for an l_2-norm attack vector of size 0.1

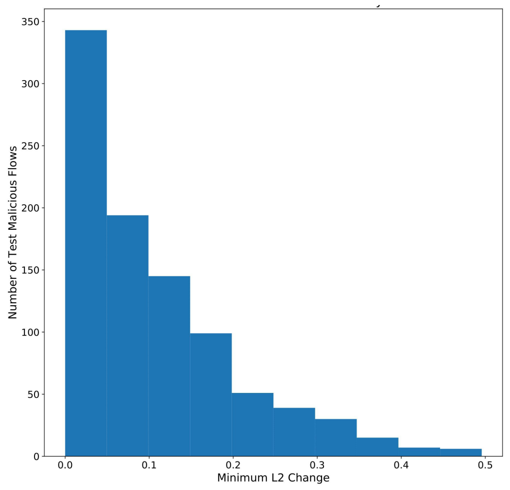


Figure 7.35: Robustness Distribution after training with Gaussian noise for an l_2-norm attack vector of size 0.7

here by presenting a subset of the Poisson subsampled distributions of robustness for varying attack/noise-training levels. Figs. 7.34-7.35 of the distributions of robustness further confirm that there is a shift earlier on towards lower levels of robustness for the Poisson subsampled autoencoder. As a result of this earlier shift, we generally see lower levels of average robustness for this network.

This concludes our presentation of results for using Poisson subsampling for privacy amplification within autoencoder while preparing our defense. We find in general that its use led to somewhat mixed and inconclusive results.

7.5 Defence: Summary

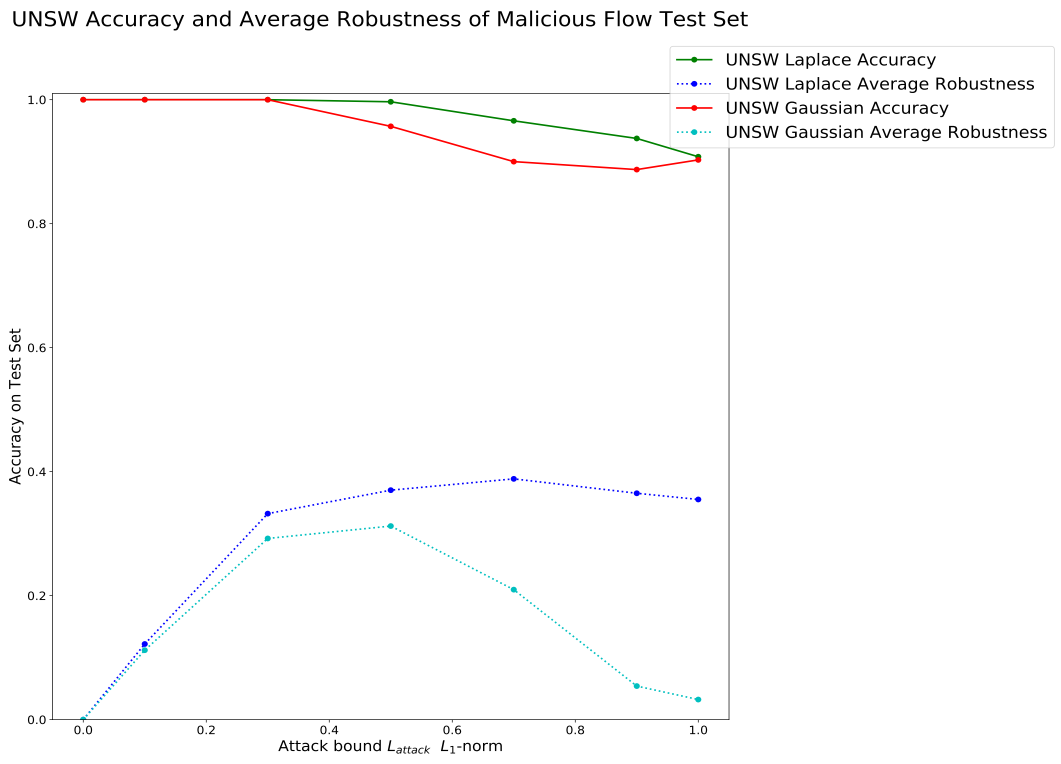


Figure 7.36: Graph of UNSW accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples on the l_1-norm.

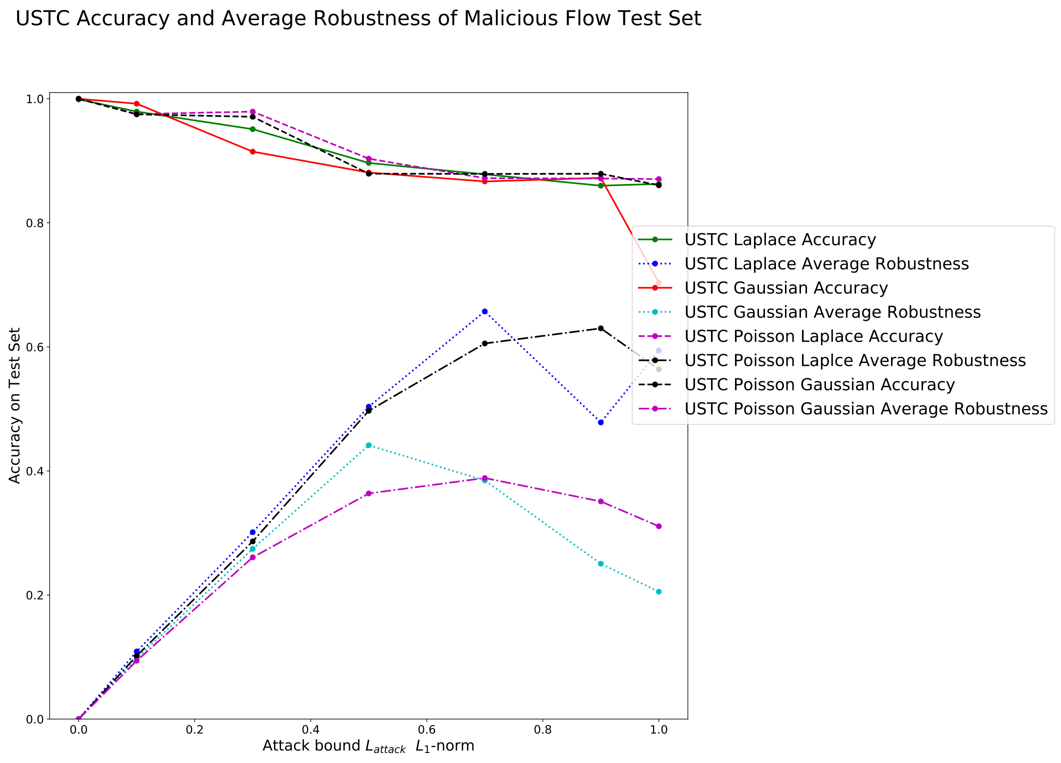


Figure 7.37: Graph of USTC accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples on the l_1-norm.

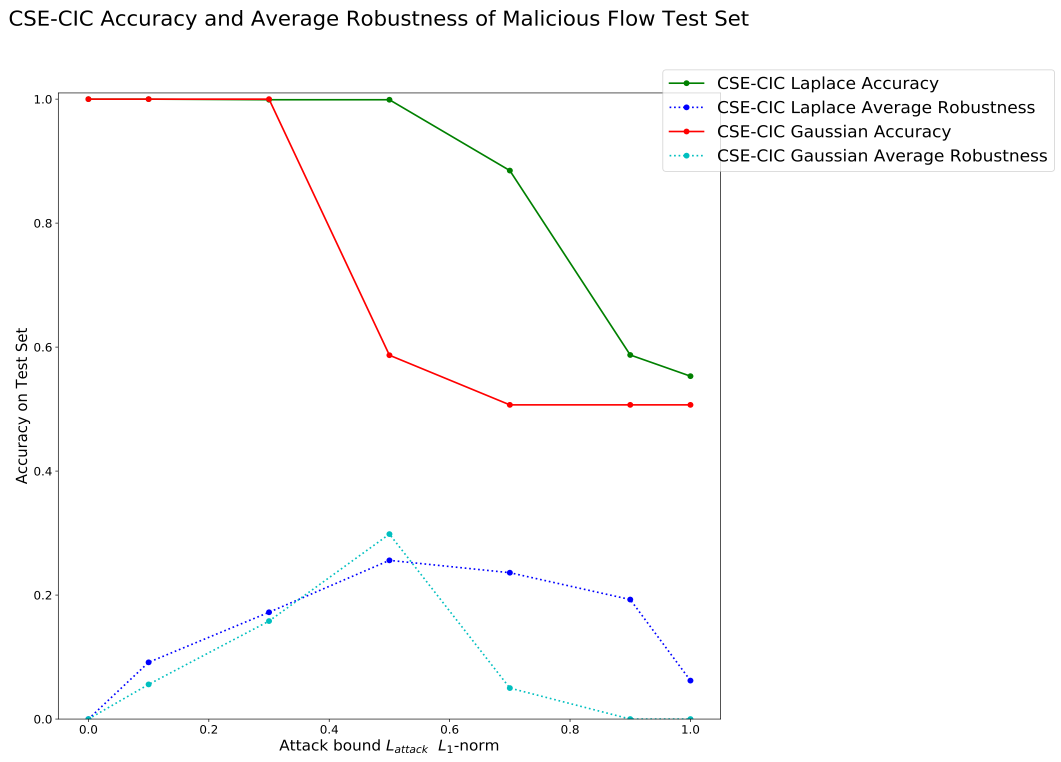


Figure 7.38: Graph of CSE-CIC accuracy on the classification of malicious flows as well as the average calculated robustness of these flows to adversarial examples on the l_1-norm.

Chapter 8

Discussion

We set aside this chapter to discuss the meaning and implication of our results. We specifically examine what are results means within the context of detecting malware and what our approach evinces.

8.1 Adversarial Attacks

As shown in this work, practical adversarial examples in both the black-box and white-box setting can be conducted on networks that make use of statistical information to detect malware. We showed in particular that an adversary can perform a black-box attack using only the returned labels for different types of flows. We acknowledge, however, that the adversary in this setting would need to know which statistical features are being used by the network and the size of the input. This technically makes our proposed black-box attacks somewhat 'grey'. This however is not a significant hurdle that an adversary would need to overcome, given that there are truly only a handful of feature subsets that an institution could use to classify flows. Furthermore, by incrementally changing key aspects of given flows until the adversary received different classifications of the flows, the adversary could determine what features malware detection system considered. In any case, using these 'grey' attacks, we found that we could transform upwards of 10% of the malicious flow test sets in all our datasets from being classified as malicious to being classified as benign. For CSE-CIC dataset in particular, an adversary could transform over 50% of malicious flows in appearing benign using a Jacobian-based data augmentation approach to train a substitute model. Similarly, we found that while we only transform around 5% of benign flows into appearing malicious on the CSE-CIC dataset, we found that could transform over 12% of the benign flow test set on the UNSW test set. Furthermore, on the USTC, we found that we could transform over 40% of the flows. These types of attack evince the power of adversaries to attack systems. By allowing 10% of attempted malware traffic to go undetected, an adversary could effectively cripple a network. In addition, by transforming a large amount of traffic (in this case upwards of 40%) into appearing malicious, the adversary could potentially overwhelm a system and prevent network analysts from being able to actually pinpoint real malware. Either of these methods could be devastating. Further, as we showed here the adversary does not actually need many capabilities to perform these attacks.

With more knowledge of the neural network while performing white-box attacks, we found that the adversary can perform even more devastating attacks on a given network. With an adversarial l_1-norm attack bound $L_{attack} = 0.5$, an adversary could transform over 30% of the UNSW malicious test set into appearing benign. Similarly, with a l_1-norm bound of $L_{attack} = 0.5$, an adversary could transform of 30% of UNSW benign traffic into appearing malicious. For the USTC dataset, an adversary could transform upwards of 10% of the malicious flows into appearing as benign with a $L_{attack} = 0.3$ l_1-norm attack bound, Further, with an l_1-norm attack bound of $L_{attack} = 0.3$, an adversary could transform over 80% of the benign traffic into appearing malicious. Finally, with a $L_{attack} = 0.1$ l_1-norm attack bound, an adversary could transform upwards of

75% of CSE-CIC dataset's malicious flows into appearing benign. Again, these types of attacks could have devastating effects on a given network meant to detect adversarial flows.

8.1.1 Differences in Effectiveness

We address here why we found such different effectiveness for transforming malicious traffic into appearing benign versus benign into appearing malicious. We think that this primarily had to do with the way that resampled the datasets. Within both the UNSW and the USTC datasets we used SMOTE in order to resample the statistical features of the malicious flows. This was because in both there was a severe imbalance between the benign and malicious flows. However, because of this resampling with SMOTE, our malicious flows were very well clustered, and each point was surrounded their high-dimensional space in many cases by other points. Since the malicious flows that were used as the basis of the adversarial examples were from the same distribution as the training set, large perturbations were needed to shift malicious flows into appearing benign. As a result, creating adversarial examples for these points was more difficult. In contrast, for the benign flows which did not undergo resampling (each point was not synthetically created), creating adversarial examples was relatively effective. We hypothesize that within their own high-dimensional space there was probably more areas where adversarial examples could be created. This is all in contrast with the CSE-CIC dataset which did not undergo sampling at all. In this dataset the benign->malicious adversarial examples were more difficult to create while malicious -> benign adversarial examples were much easier to synthesize. We suspect that this in particular had to do with how clustered the Botnet traffic for this dataset was (about a singular point in the t-SNE plot). As a result, this traffic could be easily moved outside of this cluster. In contrast, to transform a benign flow into a malicious flow was much more difficult since this would be changing much more of its placement in the 3D t-SNE space.

These are only our thoughts. In order to test this for Future Work we plan try other means of resampling in order to see if we get similar results for the USTC and the UNSW datasets. See our Future Work chapter for more details.

8.2 Interpretability

Of key importance within this work is the interpretability of the adversarial examples that we created. For example, what does it actually mean for an adversary to induct a 0.1 l_1-norm change in a given test flow to create an adversarial example? Other works like Siciu et al. [64] and Al-Dujail et al. [24] use either the malware binaries or binary encoded features of a given malware for detection; however, the techniques that they use to make their methods robust to adversarial examples are seemingly confounding. For example, it is unclear why adding specific byte sequences at the end of a malware would make it more robust to adversarial attacks. It is further unclear what the neural networks are actually learning. Similarly, Wang et al. [76] use the first 784 encrypted bytes of a given flow in order to classify it as malicious or benign. Here it is very unclear what neural network could be learning in this encrypted data. This was one of the main reasons that we chose to use statistical features within our work in order to classify the flow. This gives us a clear understanding of what the neural network learned from the training data and allows us to understand exactly what the adversarial attacks meant. For this reason, we were able to specify what the adversary had to exactly in order to create flows that would evade detection in Section 6.4. This dependence on statistical features for interpretability of course comes at the expense of some amount of accuracy. By only conducting adversarial attacks on the statistical features we somewhat limit the scope of possibilities for our attack and thus the effectiveness of some of adversarial examples.

Although we have immediate interpretability of what our adversarial examples mean since all we must do is correlate the changes back to the given statistical features, there is still the question of which adversarial metric is best to protect against: l_1-norm or the l_2-norm. As argued by both Lecuyer et al. [47] and Shamir et al. [63], protections against one type of norm attack do not correlate to attacks against another norm attack. Since we are dealing with statistical features the value of l_2-norm is not immediately clear. This is unlike for images, where l_2 is a good norm because it is a good proxy for the perceptibility a given change on an image. Here, we are mostly concerned with individual changes to given features that an adversary would use in order to evade detection. As such within this context, the l_1-norm makes the most sense. We still include the l_2-norm results in this work in order to give a full picture of our approach however. Furthermore, because the l_2-norm values can be correlated back to the l_1-norm through bounds, it still has some amount of utility

Thus, taken as a whole, we have shown that real networks are vulnerable to adversarial attacks, more specifically networks that use statistical features in order to classify flows as malicious or benign. We further have shown that targeted attacks evidence that real malware can be altered slightly to avoid detection.

8.3 How Our Defense Correspond to Actual Malware?

Here we consider what our approach means in terms of the features that we used. Because we normalized our features to be between 0 and 1, a l_1-norm difference *on a single feature* corresponds an increase of 10% of maximum value encountered for that given feature, a 0.3 l_1-norm difference *on a single feature* would correspond to a 30%

increase, etc. Obviously making a neural network robust to large changes would not always be desirable. However, for our networks, making our network robust to these changes did not lead to a large decrease in accuracy. The question of what a network should be robust to and how much it should tolerate is a question that should be answered on an individual basis. However, despite this our approach makes it so if an adversary would want to perform an attack using adversarial examples as an approach, for large values of Δ, he would be forced to change the malware in very large way. Once changed, this malware could then be detected using other means or could become more noticeable to a supplementary system. Simply our system would force an adversary to incorporate large changes into their malware which could then disrupt their malware flow, change their malware so significantly that it is no longer malware, or force them to find another means of attacking the system. This is seen most evidently from our section on targeted attacks. There we saw 0.1 l_1-norm attacks against both the UNSW and USTC CNN-0 models. In one example in order to evade detection, an adversary would have needed to raise their bps (bits per second) by 2.13Kb. Thus, for a model incorporating protection to 0.3 l_1-norm attacks, an adversary could need to increase their rate by 6.39Kb. This new adversarial malware could then be more easily detected using traditional means.

8.4 Our Defense And Its Uses

Our attacks illustrated that networks that make use of statistical features are vulnerable to adversarial attacks. However, based on our defense further showed that many of the smaller adversarial examples can be defended against without sacrificing a significant decrease in accuracy. We specifically found that by incorporating Laplace noise after the 1st layer of our autoencoder that we could manage relatively high robustness (depending on which volume of noise that we were trying to defense against). For example, we managed to achieve robustness levels above 0.3 at the expense of 5% accuracy for the USTC dataset by incorporating Laplace noise and our prediction procedure (see Fig. 7.4). This evinces that our approach can be used for small ϵ values in order to provide robustness to adversarial examples. Furthermore, this approach provides ***provable robustness***. Namely, this approach guarantees that adversarial examples under a given robustness level calculated for a given flow cannot be created. For network operators concerned about malware, this *guarantee* is outstanding.

In our results, we further see that in general for l_1-norm defense that the Laplace noise defense did better than the Gaussian noise defense. Accordingly, this makes sense due to correspondence between the Laplace distribution and the l_1-norm and the Gaussian noise with the l_2-norm. Throughout our analysis, we further saw continually that our systems did better with the Laplace noise defense to l_1-norm attacks than for l_2-norm attacks in general. Because our values are normalized to be within [0,1], this difference makes sense. This is because within this range the l_2-norm change values are always seemingly smaller. As a result, a small l_2-norm change might imply a large l_1 change. Despite this, for small l_2 attack vector sizes $L_{attack} \leq 0.3$, we see that we still get a reasonable trade-off between accuracy and the robustness

In our defense, for Poisson subsampling for privacy amplification, we saw mostly mixed results from which we could draw no definitive conclusion. As a result, further study of this property is needed in order to understand its trade-offs. By subsampling at different rates or with different methodologies we could get different results. However, here we draw no conclusions about its effectiveness.

Given our design, even when we do not achieve high robustness values, we can still use the robustness values returned in order to prioritize how when certain flows are investigated. For example, a flow with a robustness level of 0.01 could be investigated before a flow with a robustness of 0.3. Prioritizing responses to malicious traffic is a significant issue within network security. Adversarial examples make this prioritization even more difficult. This is because in most cases, a cut-off for the probability for being labelled malicious is used to prioritize flows. However, adversarial examples and flows of the sort can often completely switch a label (i.e. a flow that is nearly 89% malicious can be flipped to being 99% benign). As a result, the return robustness level can instead be used to prioritize response. It is a better metric because it truly gives a concrete distance metric before a malicious flow can be labelled as benign or vice versa. As a result, our proposed method of incorporating differential privacy into ensuring robustness has the added feature of improving response levels as well.

8.5 Limitations

There are several limitations to this work. Specifically, we only look at the final flows after they have completed. In other words, this work focused on detecting different types of malware/flows after they had already occurred. More difficult would be determining whether a flow was malicious or benign in the first few bytes and creating adversarial examples based on that statistical information. We leave the investigation of this area to future work.

Another limitation of this work is that we do not test these results in a real-world setting. It would be interesting to see how the implementation of our procedure could be used to improve the triaging approaches of real-world malware detection systems.

Lastly, we also do not explicitly consider the generalizability of systems to different types of malware that the systems were not trained to detect. Malware detection system must be able to detect unknown types of malware. We used our test sets as a proxy for these unknown types of flows. However, it would be an interesting area for us to test these malware detection networks on unseen types of malware and create adversarial examples from these flows.

Now that we have thoroughly discussed our results and their implications, we now discuss some of the work we wish to continue on this project in the future.

Chapter 9

Future Work

In this section, we delve into possible future areas that we can explore in the future.

9.1 Different and Real-World Datasets

The most obvious area that we would want to explore is the use of different datasets. Within this work, we used data from UNSW-NB15 [53], USTC-TFC2016 [75] (this dataset also took a subset of flows from the CTU-13 dataset), and CSE-CIC-IDS2018 [4]. However, it would be interesting to extend our work to more updated and recent datasets and to much larger ones as well. One possible dataset that we could use in the full CSE-CIC-IDS2018 dataset [4] . For this work, we considered a subset that only considered consisted of Botnet attacks. However, this dataset specifically includes Brute Force attacks, DoS attacks (Hulk, GoldenEye, Slowloris, Slowhttptest, Heartleech), Damn Vulnerable Web App web attacks, infiltration attacks, port scans and Botnet attacks Applying our methods to the full updated dataset would allow us to perceive how far our approach is applicable to different network settings and different types of attacks. Furthermore, the full dataset is much larger than the datasets considered in this work.

In addition to the above dataset, we would also like to consider the ISCX dataset [38]. The ISCX dataset published by Draper et al. consists of 7 days of encrypted synthetic network traffic that replicates real-world network attack traffic. The traffic was captured using Wireshark and consists of a 28GB of data, which is again much larger than what we considered in this work.

In addition to the above datasets, for future work we would apply our approach to real user data from the University of Oxford in order to truly see the applicability of the networks considered in a real-world setting.

9.2 Different Neural Network Models

Within this work we considered several different neural network models but focused specifically on the CNN-0 model that gave us the best initial results. See Section 6.1 for more details. For future work we would like to implement and experiment with several different neural network models. Specifically, we tested a model that characterized traffic using timing information in conjunction with statistical measurements, we would also like to attempt to use models that take into consideration the encrypted byte information of individual flows in conjunction with statistical measurements. For example, the methodologies of those like Wang et al. [70] and Wang et al. [73] consider the byte distribution in addition to temporal features within their approaches. Further within Wang et al. [73] specifically they make use of an (Long-Short Term Memory) LSTM to learn information from temporal data. Our initial forays into using LSTMs for our approach did not result in anything very fruitful. However, by making use of different features and using LSTMs to analyse timing information we could perhaps get better results for detecting malware. In all of the above portions however we would again like to focus on changing statistical features when attacking the networks. Again, for interpretability we would like to only attack features that we can potentially explain. However, in order to get better overall accuracy, these proposed models could perhaps lead to better results.

9.3 Different Feature Subsets

As noted earlier, we used a subset of Piskozub et al's. [59] features for the USTC dataset. However, in would be interesting to ascertain what feature subsets would lead to the best accuracy for our network and how changing the sizes of the given subset could potentially change how effective adversarial attacks would be on the network. Once this initially done, we would also like to apply a similar approach to the UNSW dataset and the CSE-CIC-IDS2018 dataset.

9.4 Different Ways of Adding Differential Privacy

In this work, we focused on adding differential privacy after the 1st layer of an autoencoder and directly on the features. However, despite this focus, there are several other ways of adding differential privacy to these networks to make them robust to adversarial examples. For example, the autoencoder could be scrapped and instead the model itself could be trained with noise after one of its initial layers. Furthermore, noise could be placed deeper within the network as well.

Lastly, we considered using Poisson subsampling in order to improve the results of our experiments. This ended up giving us mixed results with no clear conclusion being drawn on its benefits. Future work would include other mechanisms of subsampling besides Poisson and different sampling rates to see if they resulted in better results.

9.5 Different Ways of Sampling the Datasets

As mentioned earlier in this work, we used SMOTE in order to resample the malicious flows in our dataset due to the imbalance between malicious and benign flows. However, based on our results, this led to differing behaviours in terms of our network's susceptibility adversarial attacks (i.e. benign flows -> malicious flows were easier to create while malicious -> benign flows were more difficult). It would be interesting to see if using other methods of sampling really caused this behaviour or whether it was an inherent characteristic of the networks that we used. For this reason, we propose in future work to try other means of resampling including sampling with replacement, Poisson sampling with replacement, among other in order to see if we get different behaviours.

9.6 Randomized Smoothing for Certified Robustness

The last technique that we wish to try is randomized smoothing. Proposed by Cohen et al. [37], this smoothing has been found to provide certifiable protection to adversarial examples. After projecting our features to a 784 or 1024-dimensional space randomized smoothing (which we described in the Background chapter), we would apply randomized smoothing in order to provide certifiable robustness. This approach published during the summer of 2019 takes inspiration from Lecuyer et al. [47] so it would be interesting to compare results when this novel approach is applied in our own domain.

Chapter 10

Conclusion

We demonstrated in this work that (1) neural networks that detect malware are vulnerable to adversarial examples and (2) that by incorporating differential privacy as well as a multiple prediction scheme for labelling can make models that detect malware robust to these examples. More concretely we did the following:

1. We implemented several different neural networks that differentiate malicious traffic from benign traffic for the UNSW-NB15 [53] and CSE-CIC-IDS2018 datasets [4] and that differentiate malware traffic from benign traffic for the USTC-TFC2016 [75] dataset. Using our methodology, we manage to achieve 98%, 99.89%, and 99.80% on the UNSW,USTC, and CSE-CIC datasets respectively.

2. We implemented two different architectures to show that more fine-grained analysis of these datasets can be conducted in order to differentiate different types of malicious/malware flows.

3. We illustrated our best model's vulnerability to both black-box and white-box types of attacks. We particularly show that an adversary could potentially trans-

form over 40% of the benign flows in a test into appearing malicious within the USTC dataset using black box methods. This same adversary could also transform at least 10% of malicious flows into appearing benign using only black box methodologies in all considered datasets. For the CSE-CIC dataset we can transform over 75% of the malicious flows into appearing benign. We then further illustrate how a more knowledgeable adversary in a white-box setting could improve upon these results to transform over 30% of UNSW malicious flows into appearing benign, over 80% of USTC's benign flows into appearing malicious, and 5% of CSE-CIC benign flows into appearing malicious. We present attacks of this sort for both the l_1 and l_2 norms.

4. We illustrated targeted attacks on our networks to show that a knowledgeable adversary could attack single statistical features in order to create adversarial examples. We then correlated these changes in features to real malware.

5. We demonstrated how differential privacy can be used in order to make models robust to adversarial examples.

6. We demonstrated that other methods of adding differential privacy using Poisson subsampling to our models can lead to differing and sometimes better results.

7. We finally showed that using our approach, an adversary would be forced to change a given malware by a large amount in order to create an adversarial example against a given model.

8. We showed that a metric used within our differential privacy approach can be used a form of prioritization for investigating malicious flows.

This work showed that while maintain a relatively high accuracy that for small perturbation attack vector sizes $L_{attack} < 0.3$, we could make our models robust to adversarial attacks. We further illustrated in this work how adversarial the robustness calculation proposed by Lecuyer et al. [47] can also be used to manage prioritization of malware investigation. Namely because adversarial examples can easily flip SoftMax probabilities, they are not reliable metrics for determining which flows to investigate. However, our metric is entirely robust as it determines an actual Δ that characterizes the distance of a given test instance to being labelled differently. Lastly in this work, we proposed additional steps that can be taken in order to improve upon this work and continue to make malware detection systems robust to adversarial examples and more reliable systems overall.

References

[1] Adversarial example. https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjHnsr-kKXkAhVHzRoKHal1CPMQjRx6BAgBEAQ&url=https. Accessed: 2019-08-20.

[2] Cisco joy. https://github.com/cisco/joy. Accessed: 2019-08-20.

[3] Cridex. https://www.symantec.com/security-center/writeup/2012-012103-0840-99. Accessed: 2019-08-20.

[4] Cse-cic-ids2018 on aws. https://www.unb.ca/cic/datasets/ids-2018.html. Accessed: 2019-08-20.

[5] Encryption. https://arstechnica.com/tech-policy/2019/08/post-snowden-tech. Accessed: 2019-08-20.

[6] Flowmeter. https://github.com/ISCX/CICFlowMeter. Accessed: 2019-08-20.

[7] Geodo. https://cofense.com/dark-realm-shifting-ways-geodo-malware/. Accessed: 2019-08-20.

[8] Goldeneye. https://www.bitdefender.com/news/massive-goldeneye-ransomware-attack-affects-users-worldwide-3330.html. Accessed: 2019-08-20.

[9] Google colab. https://colab.research.google.com/.

[10] Htbot. https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor. Accessed: 2019-08-20.

[11] Machine learning. https://www.cs.ox.ac.uk/teaching/materials18-19/ml/index.html. Accessed: 2019-08-20.

[12] mirai. https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. Accessed: 2019-08-20.

[13] Miuref. https://www.symantec.com/security-center/writeup/2014-011518-0225-99. Accessed: 2019-08-20.

[14] Neris. https://apan.net/meetings/apan42/Sessions/77/SDN-1.pdf. Accessed: 2019-08-20.

[15] Nsis-ay. https://fortiguard.com/appcontrol/45139. Accessed: 2019-08-20.

[16] Probability and computing lectures. https://www.cs.ox.ac.uk/people/elias.koutsoupias/pc2018-19/lectures.html#Lecture. Accessed: 2019-08-20.

[17] Random forest feature importance. https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e. Accessed: 2019-08-20.

[18] Shifu. https://securityintelligence.com. Accessed: 2019-08-20.

[19] Target attack. https://www.wired.com/2014/01/target-malware-identified/. Accessed: 2019-08-20.

[20] Tinba. https://threatvector.cylance.com/en-us/home/blackberry-cylance-vs-tinba-banking-trojan.html. Accessed: 2019-08-20.

[21] Virut. https://www.symantec.com/security-center/writeup/2007-041117-2623-99. Accessed: 2019-08-20.

[22] Zeus. `https://www.symantec.com/security-center/writeup/2010-011016-3514-99`. Accessed: 2019-08-20.

[23] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[24] Abdullah Al-Dujaili, Alex Huang, Erik Hemberg, and Una-May O'Reilly. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 76–82. IEEE, 2018.

[25] Bushra A AlAhmadi and Ivan Martinovic. Malclassifier: Malware family classification using network flow sequence behaviour. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13. IEEE, 2018.

[26] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM workshop on artificial intelligence and security*, pages 35–46. ACM, 2016.

[27] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[28] Borja Balle, Gilles Barthe, and Marco Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *Advances in Neural Information Processing Systems*, pages 6277–6287, 2018.

[29] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 807–822, 2016.

[30] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-cnn: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, 2019.

[31] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[32] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

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

[34] Jianbo Chen and Michael I Jordan. Boundary attack++: Query-efficient decision-based adversarial attack. *arXiv preprint arXiv:1904.02144*, 2019.

[35] Yun-Chun Chen, Yu-Jhe Li, Aragorn Tseng, and Tsungnan Lin. Deep learning for malicious flow detection. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2017.

[36] Zhitang Chen, Ke He, Jian Li, and Yanhui Geng. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1271–1276. IEEE, 2017.

[37] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[38] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[39] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. Operational experiences with high-volume network intrusion detection. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 2–11. ACM, 2004.

[40] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Zhanglong Ji, Zachary Chase Lipton, and Charles Elkan. Differential privacy and machine learning: a survey and review. *CoRR*, abs/1412.7584, 2014.

[44] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[45] Nizar Kheir and Chirine Wolley. Botsuer: Suing stealthy p2p bots in network traffic through netflow analysis. In *International Conference on Cryptology and Network Security*, pages 162–178. Springer, 2013.

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

[47] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. *arXiv preprint arXiv:1802.03471*, 2018.

[48] Ninghui Li, Wahbeh Qardaji, and Dong Su. On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 32–33. ACM, 2012.

[49] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[50] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[51] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[52] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, volume 7, pages 94–103, 2007.

[53] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[54] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[55] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 10, 2016.

[56] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

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

[58] Mijung Park, Jimmy Foulds, Kamalika Chaudhuri, and Max Welling. Dp-em: differentially private expectation maximization. *arXiv preprint arXiv:1605.06995*, 2016.

[59] Michal Piskozub, Riccardo Spolaor, and Ivan Martinovic. Malalert: Detecting malware in large-scale network traffic using statistical features. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):151–154, 2019.

[60] Paul Prasse, Lukáš Machlica, Tomáš Pevný, Jiří Havelka, and Tobias Scheffer. Malware detection by analysing network traffic with neural networks. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 205–210. IEEE, 2017.

[61] Shahbaz Rezaei and Xin Liu. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine*, 57(5):76–81, 2019.

[62] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148. ACM, 2004.

[63] Adi Shamir, Itay Safran, Eyal Ronen, and Orr Dunkelman. A simple explanation for the existence of adversarial examples with small hamming distance. *arXiv preprint arXiv:1901.10861*, 2019.

[64] Octavian Suciu, Scott E Coull, and Jeffrey Johns. Exploring adversarial examples in malware detection. *arXiv preprint arXiv:1810.08280*, 2018.

[65] Liuying Sun, TS Anthony, Ho Zhe Xia, Jiageng Chen, Xuzhe Huang, and Yidan Zhang. Detection and classification of malicious patterns in network traffic using benford's law. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 864–872. IEEE, 2017.

[66] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[67] Florian Tramèr and Dan Boneh. Adversarial training and robustness for multiple perturbations. *arXiv preprint arXiv:1904.13000*, 2019.

[68] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. A deep learning based method for handling imbalanced problem in network traffic classification. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, pages 333–339. ACM, 2017.

[69] Derek Wang, Chaoran Li, Sheng Wen, Yang Xiang, Wanlei Zhou, and Surya Nepal. Defensive collaborative multi-task training - defending against adversarial attack towards deep neural networks. *CoRR*, abs/1803.05123, 2018.

[70] Pan Wang, Feng Ye, Xuejiao Chen, and Yi Qian. Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access*, 6:55380–55391, 2018.

[71] Shanshan Wang, Zhenxiang Chen, Lei Zhang, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2016.

[72] Shanshan Wang, Qiben Yan, Zhenxiang Chen, Bo Yang, Chuan Zhao, and Mauro Conti. Detecting android malware leveraging text semantics of network flows. *IEEE Transactions on Information Forensics and Security*, 13(5):1096–1109, 2017.

[73] Wei Wang, Yiqiang Sheng, Jinlin Wang, Xuewen Zeng, Xiaozhou Ye, Yongzhong Huang, and Ming Zhu. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access*, 6:1792–1806, 2017.

[74] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017.

[75] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717. IEEE, 2017.

[76] Yao Wang, Jing An, and Wei Huang. Using cnn-based representation learning method for malicious traffic identification. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pages 400–404. IEEE, 2018.

[77] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.