Spring 5-4-2021

Detection and Prediction of Distributed Denial of Service Attacks using Deep Learning

Christopher Freas

Detection and Prediction of Distributed Denial of Service Attacks using Deep Learning

by

Christopher B. Freas

Under the Direction of Robert W. Harrison, Ph.D.

ABSTRACT

Distributed denial of service attacks threaten the security and health of the Internet. These attacks continue to grow in scale and potency. Remediation relies on up-to-date and accurate attack signatures. Signature-based detection is relatively inexpensive computationally. Yet, signatures are inflexible when small variations exist in the attack vector. Attackers exploit this rigidity by altering their attacks to bypass the signatures. The constant need to stay one step ahead of attackers using signatures demonstrates a clear need for better methods of detecting DDoS attacks.

In this research, we examine the application of machine learning models to real network data for the purpose of classifying attacks. During training, the models build a representation of their input data. This eliminates any reliance on attack signatures and allows for accurate classification of attacks even when they are slightly modified to evade detection. In the course of our research, we found a significant problem when applying conventional machine learning models. Network traffic, whether benign or malicious, is temporal in nature. This results in differences in its characteristics between any significant time span. These differences cause conventional models to fail at classifying the traffic. We then turned to deep learning models. We obtained a significant improvement in performance, regardless of time span. In this research, we also introduce a new method of transforming traffic data into spectrogram images. This technique provides a way to better distinguish different types of traffic. Finally, we introduce a framework for embedding attack detection in real-world applications.

INDEX WORDS: Application Level Intelligence, Attack Detection, Deep Learning, Denial of Service, Distributed Denial of Service, Flow analysis, Machine Learning, Networks, Security

Detection and Prediction of Distributed Denial of Service Attacks using Deep Learning

by

Christopher B. Freas

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2021

Detection and Prediction of Distributed Denial of Service Attacks using Deep Learning

by

Christopher B. Freas

Committee Chair: Robert Harrison

Committee: Xiaojun Cao

Rajshekhar Sunderraman

Yichuan Zhao

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

May 2021

DEDICATION

I dedicate this work to my family, my friends, and anyone who needs just a bit of inspiration to take the steps they need to get their goals accomplished.

ACKNOWLEDGEMENTS

I would like to first thank my family. For my wife, Alexandra, I love you more every day. Your unending love and support for me as I spent uncountable nights and weekends studying is why I've made it to this point. Thank you for everything you do for our family. For my son Evan and my daughter Madison, Daddy loves you more than you can imagine. Thank you for your endless energy and enthusiasm. I smile from ear to ear every time I think of you two. The three of you have changed me for the better in so many ways.

To my advisor, Dr. Robert Harrison, thank you for everything. I'm forever grateful to you for agreeing to be my advisor. I'm thankful for your patience and willingness to work with me as a part time student. Under your tutelage, I've become a computer scientist and a published researcher. Given where I started, that's no small feat. Your boundless knowledge and keen eye have helped me to open new doors, see things I would otherwise miss, and make connections between things that would otherwise seem unrelated.

To my lab mates, both past and present - Dr. Andrew Rosen, Dr. Brendan Benshoof, Dr. Chinua Umoja, Dr. Erin Elizabeth-Durham, Michael J. McDermott, Dhara Shah, David Bolding, and their families - thanks for your humor, your wisdom, and the community you provided me.

I would like to thank my work colleagues - Abdallah El Kadi, Randoll Revers, Robert Barnes, and Tyler Killian - I'm grateful every day I get to work with such an outstanding team. I would especially like to thank my manager, Dr. Marco Valero. I will always cherish your constant encouragement, support, and friendship.

I would also like to acknowledge my committee members - Dr. Rajshekhar Sunderraman, Dr. Xiaojun Cao, and Dr. Yichuan Zhao. It has been the privilege of my life to work with you all. Finally, thanks to Adrienne Martin, Celena Pittman, Tammie Dudley, and Venette Rice. Though I don't get to see you every day, I'm grateful to you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- ANN - Artificial Neural Network
- AS - Autonomous System
- ASN - Autonomous System Number
- CART - Classification And Regression Trees
- CDN - Content Delivery Network
- CNN - Convolutional Neural Network
- DBN - Deep Belief Network
- DDoS - Distributed Denial of Service
- DNS - Domain Name System
- DoS - Denial of Service
- Gbps - Gigabits per second
- IANA - Internet Assigned Numbers Authority
- IAT - Inter-Arrival Time
- ICMP - Internet Control Message Protocol
- ID3 - Iterative Dichotomiser 3
- IDS - Intrusion Detection System
- IP - Internet Protocol
- IRC - Internet Relay Chat

- KDD - Knowledge Discovery and Data mining
- kNN - k-Nearest Neighbors
- LSTM - Long Short-Term Memory
- MLP - Multilayer Perceptron
- NP - Nondeterministic Polynomial time
- NTP - Network Time Protocol
- PCA - Principal Components Analysis
- RBM - Restricted Boltzmann Machine
- RNN - Recurrent Neural Network
- SNMP - Simple Network Management Protocol
- SVM - Support Vector Machine
- TCP - Transmission Control Protocol
- UDP - User Datagram Protocol

PART 1

INTRODUCTION

The Internet is an indispensable part of our social, economic, and political well-being. Indeed, as Tim Berners-Lee, the father of the World Wide Web, put it, "There was a time when people felt the Internet was another world, but now people realise it's a tool that we use in this world.". Along with this success has come a threat in the form of denial of service (DoS) attacks, where a malicious user attacks some resource to deny the service or services it provides to other users. Denial of service attacks are generally either a "centralized" attack where the malicious user attacks a target (or targets) from a single source, or a so-called "distributed" denial of service (DDoS) attack, where the attacker can use many hundreds or thousands of compromised computer systems to launch an attack. The focus of this research is on attacks of the latter type. The magnitude of these attacks can range from benign pranks to very costly down time. Though commercial and non-commercial solutions exist to mitigate these attacks, they typically rely on attack signatures to mitigate known attacks. Similar to computer malware and viruses, DDoS attacks tend to change over time in order to defeat these signatures. This means that any system for mitigating them must be able to cope with these changes in order to detect anomalous traffic behavior.

1.1 An Overview of Denial of Service Attacks

Fundamentally, a denial of service attack can be thought of as any attack on a resource that is meant to deny other, legitimate use of that resource. Denial of service attacks generally take one of two forms: a "centralized" attack where the malicious user attacks a target (or targets) from a single source, or a so-called "distributed" denial of service (DDoS) attack, where the attacker can use many hundreds or thousands of compromised computer systems to launch an attack. In both methods, the attacker conceals his identity and location

in order to maximize the lifetime of the attack. The following sections provide an overview of each method.

1.1.1 Traditional Denial of Service Attacks

Traditional denial of service attacks can be thought of as an attack meant to disrupt or crash a service. There are two general forms of DoS attacks: those that crash services and those that flood services[2]. A buffer overflow attack is an attack in which the attacker exploits code which does not properly sanitize the input given to it. The result is that the bounds of a finite memory buffer are exceeded, allowing the attacker to crash the service or simply take control of it. Examples of flooding services are a SYN[1] attack, where the attacker floods a target with TCP SYN packets in order to exhaust the target's memory, the ICMP[2] "ping of death" attack, in which the attacker sends an extremely large payload in an ICMP packet to crash operating systems that are unable to handle them, a Teardrop attack, where the attacker sends fragmented packets whose fragment offsets overlap [3], to a simple ping flood, where the attacker tries to consume the target's bandwidth with large amounts of ICMP packets. With the exception of the ping flood attack, all modern operating systems have been patched against these sorts of attacks. The ping flood attack, when carried out by a large number of attackers, can be particularly effective. In this scenario, it is considered to be a distributed denial of service attack, and is discussed in the next section.

1.1.2 Distributed Denial of Service Attacks

When several attacking systems are coordinated and directed at a target, the attack is known as a distributed denial of service attack. A DoS attack is shown in Figure 1.1 and a DDoS attack is shown in Figure 1.2 for comparison.

---

[1]The SYN, or synchronize packet, is the first packet in the TCP "three-way handshake" and is used to establish a connection to a remote system.

[2]ICMP is used for a variety of functions, including checking whether a system is reachable.

[3]Packet fragmentation is used when a packet is too large to be transmitted on one or more segments along the path from its source to its destination. Fragment offsets allow the receiver to reassemble the received fragments correctly.

Figure 1.1. A DoS attack.

The DoS attack in Figure 1.1 is meant to be an abstraction - these types of attacks do not necessarily require the attacker to send malicious packets to the target. In the case of a buffer overflow attack, the attacker could even be a valid user on the target system!

The DDoS attack shown in Figure 1.2 is by definition reliant on the attackers sending malicious packets to the target, however, the packets are not known a priori to be malicious.

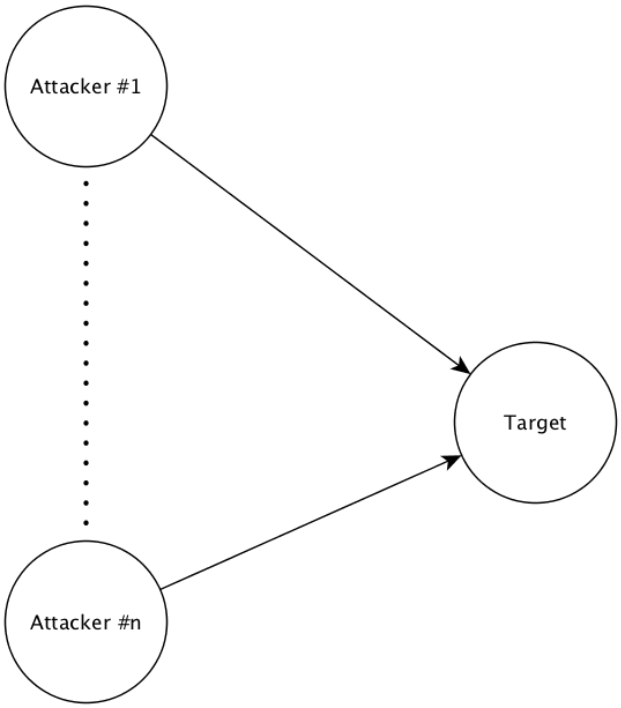

Figure 1.2. A DDoS attack.

The coordination of attacking systems for these kinds of attacks can be managed by a single attacker employing one of two known methods [3]. The first is the Agent-Handler model where the attacking systems are compromised computers with agent software installed to carry out the attack. The attacker controls and monitors the attacking systems via the

handlers, which each control a subset of the attacking systems. The second is a so-called "botnet", which is a collection of compromised computers controlled by the attacker and directed to attack a target [4]. Each of the attacking systems connects to a private IRC channel created for the specific purpose of coordinating an attack. In both architectures, the attacker issues commands to either a handler or directly to an agent on a compromised system to launch an attack.

Amplification Attacks DDoS amplification attacks exploit the characteristics of certain packet types and protocols in order to provide volumetric increases in malicious attack traffic in response to small amounts of legitimate query traffic. An example of an amplification attack is shown in Figure 1.3.

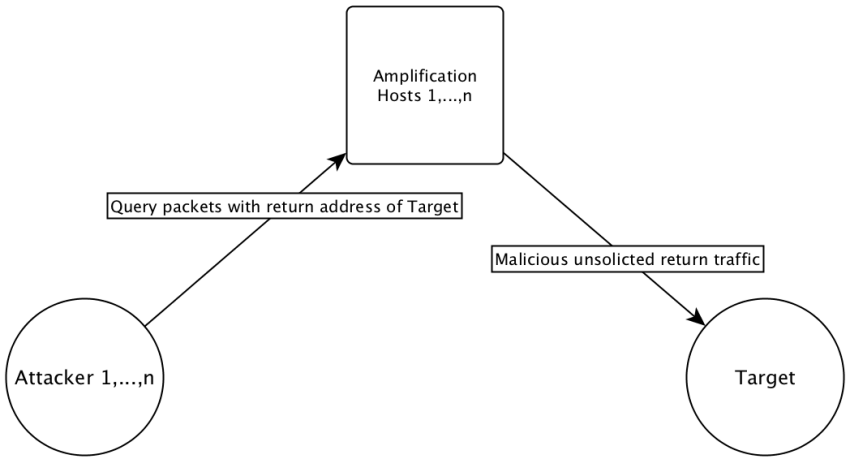

Figure 1.3. A DDoS amplification attack.

In the figure, the attacking systems send a small payload consisting of a query of some type to a set of systems running a service whose response traffic is known to be larger than the query traffic. The query packets have forged IP destination address headers, which are set to the target's IP address. Depending upon the service running on the amplification hosts, an amplification factor of several hundreds of times the size of the query traffic can be achieved. In the case of DNS reflection, the amplification factor is 8x, meaning the attack generates eight times more response traffic, which is then sent to the target, than query

traffic. However, in the case of NTP and SNMP reflection, amplification can be over 200x and 650x, respectively [5].

Perhaps the most well known amplification attack is the NTP reflection attack carried out in 2014 against CloudFlare, a company that provides CDN services for popular websites. As detailed in [5], the attackers exploited a flaw in NTP whereby an attacking system queries an exposed NTP server for a list of systems that have recently queried the NTP server (called a "MONLIST") [6]. Because the size of the MONLIST is larger than the original query, the response traffic has thus been "amplified". The attack against CloudFlare was measured at over 400 Gigabits per second (Gbps) with an amplification factor of 200x [5].

1.1.3 Impacts of Denial of Service Attacks

The impacts caused by a denial of service attack vary. For traditional DoS attacks, the impact is usually a brief disruption for a single service. For DDoS attacks, the impact is much greater. Because these attacks typically target large networks which host services or users in a shared environment, when a single end host is targeted, it often takes down services for other users sharing the environment. In extreme cases such as the CloudFlare attack mentioned previously, entire networks can become unreachable for the duration of the attack.

1.2 Research Outline

Throughout this document, a chronology of tools and information that details this research will be presented. This project was born out of my many years of experience as a professional network engineer and architect. I had several occasions where I had to help remediate DDoS attacks and the lessons learned were invaluable. However, I lacked a deep understanding of what is really needed for useful attack detection. This research intends to fill that knowledge gap for myself and anyone that may be interested in DDoS attacks.

The most salient goal of this research is to develop a machine learning tool (or set of tools) that can analyze network traffic in order to detect and even predict these sorts of

attacks with a reasonable level of accuracy. The first goal is to develop a strong understanding of DDoS attacks. The next goal is to develop a model of normal versus anomalous traffic behavior. Various machine learning models will be evaluated against real traffic data sets. This allows us to determine if machine learning is appropriate and which model or models work best. Finally, we apply what we've learned to create a framework that can be applied in real-world settings to detect malicious behavior.

1.3 Existing Approaches to Detection

Broadly, existing approaches to the detection of DDoS attacks can be placed in one of two categories: network-based detection and application-level detection. These can be further classified into centralized detection schemes and distributed detection schemes. Application-level detection of DDoS attacks is not of interest for this research and is not discussed further.

Chen et al. propose in [7] the Distributed Change Point (DCP) detection architecture, which uses a system of distributed collection points to monitor traffic traversing several networks. Once the traffic level exceeds a predefined router threshold, β, the traffic is considered suspicious. This is performed using a metric called *deviation from average* (DFA), a ratio of historical traffic averages and the current traffic level. DFA is defined formally as the following:

$$DFA_{in}(t_m, i) = \frac{S_{in}(t_m, i)}{\overline{X}(t_m, i)} \tag{1.1}$$

In Equation 1.1, the values t_m and i refer to the number of packets received by a router at discrete time interval m on physical port i. The function $\overline{X}(t_m, i)$ is then the historical average number of packets received at discrete time interval m on physical port i. The function $S_{in}(t_m, i)$ is the deviation from the average defined in $\overline{X}(t_m, i)$.

This architecture performs well but is limited to the detection of so-called flooding attacks. Additionally, it requires cooperation between several network operators residing in

different autonomous systems[4] for attack remediation. Speaking from practical experience, network operators are frequently reluctant to add any additional software or hardware to their networks based on security and operational concerns.

Similar statistical approaches exist, such as the D-WARD system proposed by Mirković et al. in [8], which rate limits traffic considered to be non-compliant with a set of predefined models for normal traffic. In contrast to DCP, D-WARD is placed only at the source of traffic, and not egress points between networks. Several unanswered questions exist, namely how to deal with the possibility of attacks within a network, updating the models of normal traffic, and how to effectively deal with high speed routers (that is, terabit-capable routers).

Other purely statistical approaches, such as [9], rely on increasing or decreasing a rate limit based on end system feedback, or modeling the TCP SYN arrival rate to detect attacks [10].

In [11], Niyaz et al. use deep learning techniques as well as a software-defined networking (SDN) approach for DDoS detection. For deep learning, a stacked autoencoder is used, which, when trained, feeds data into a softmax[5] classifier. For their SDN approach, a Traffic Collector and Flow Installer (TCFI), Feature Extractor, and Traffic Classifier module are used to automatically program remediation flows in the SDN controller. These modules rely on the stacked autoencoder for detection. This approach achieves a 95.65% accuracy with a false-positive rate of 5%. However, the authors chose to capture raw packet data as it arrives instead of relying on a network flow collection protocol. Due to this, the application of this approach will be limited to smaller networks.

Finally, in [12], Fiore et al. combine a discriminative RBM (DRBM) with feature-rich training data. The discriminative RBM shows promise, however, it does suffer from long training times and an inability to cope with data that is widely different than the network data that training was performed on. This suggests that the DRBM may be overfitting the

[4]An autonomous system (AS) is a 32-bit integer value assigned to network operators by the IANA and delineates all of the devices under the control of a single network operator, such as Google, Level3, and so on.

[5]Softmax is the term for generalized multi-class classification using the sigmoid function discussed in section 2.1.2.

data. The authors acknowledge these issues and propose future research to determine a way to generalize anomaly detection using a DRBM.

This small sample of existing literature is by no means comprehensive. It is merely meant to provide a taste for the breadth of existing DDoS attack detection research. Additional references to relevant literature will be made throughout the remainder of this dissertation as appropriate.

PART 2

ATTACK DETECTION USING CONVENTIONAL MACHINE LEARNING MODELS

The detection of DDoS attacks fits well within the umbrella of big data. Big data problems are characterized by five Vs: Volume, Velocity, Variety, Veracity, and Value [13]. Reliable estimation of the threat from a network attack requires rapid (Velocity) and accurate (Veracity, Value) estimation of a non-homogeneous threat (Variety) in the presence of terabytes of data (Volume). Forecasts on both the Volume and Velocity of network traffic show a doubling in the next three years [14].

In this chapter, we examine the use of conventional machine learning models for classifying attacks on network flow data. The data contain a combination of benign traffic and several types of malicious traffic. We applied the Decision Tree, Random Forest, and k-nearest neighbors algorithms to the full data set. We then broke the data set into several non-overlapping sets based on the time of collection to determine how the models would perform with a time difference. The accuracy significantly deteriorates when attacks are present in the test data but not the training data which demonstrates that the system is non-stationary. Finally, we applied feature reduction techniques to determine the most relevant features in the data. We then retested the models to observe the impact of a reduced feature space. Since conventional machine learning algorithms converge poorly on non-stationary data, we conclude that generalizing on network flow data requires more advanced machine learning algorithms.

The next section provides a brief overview of machine learning. Readers with a background in machine learning can skip to Section 2.2 if desired.

Size	Color	Texture	Shape	Name
Medium	Red	Firm	Round	Red Apple
Medium	Orange	Soft	Round	Orange
Small	Purple	Soft	Round	Grape
Large	Yellow	Firm	Crescent	Banana

Table 2.1. A sample labeled data set.

2.1 An Overview of Machine Learning

Machine Learning can be roughly defined as a set of methods and techniques that can be used to help make sense of a set of data. Under this definition, machine learning is almost indistinguishable from the related field of data mining. Indeed, many data mining tool kits and software make heavy use of machine learning models[1]. A better definition of machine learning as it relates the the field of artificial intelligence was coined in 1959 by machine learning pioneer Arthur Samuel: *"Machine learning gives computers the ability to learn without being explicitly programmed."*. This definition is, of course, not entirely accurate since a computer can't learn to drive a car by simply reading a book. Instead, a learning algorithm is used with a set of data to make decisions, infer certain values, or group the data in some meaningful way.

The most common way to make decisions using machine learning is with classification, which is discussed in the next section. When the requirement is to infer future values based on previous observations, regression is most commonly used. Together, classification and regression are referred to as *supervised learning*, since the particular learning algorithm must first be *trained* with a set of data containing *labels* for the correct output. During training, the model builds an internal representation of the training data. Once the algorithm has been trained (using a training data set), data with the same features (called a test or validation data set) and no labels can be given to the algorithm and it will label the data.

If no labels exist in the data, *unsupervised learning* is used to gain some insight about

[1]A great example of this is the Apache Spark cluster computing framework. See the following link for more information: `https://spark.apache.org/`.

the data (possibly for the purpose of adding labels!). The most common approach to unsupervised learning is clustering, which is discussed in Section 2.1.3.

2.1.1 Classification

The goal of classification algorithms is to place an unknown piece of data into a known category or class [15]. Table 2.1 shows an example data set. The columns provide the *features* (sometimes referred to as *attributes*) for each row of data. Each feature describes an aspect of the data that is considered the most relevant to the classification task at hand. In the table, the features are Size, Color, Texture, and Shape. The last column provides the *label* for each row, which is the category or class the particular row belongs to. Each row is referred to as an *instance*. The labels are Apple, Orange, Grape, and Banana. This table illustrates a data set used to train a learning algorithm. A sample (or test) data set would be the same, but the Name column would not be present. Table 2.1 is an example of multiclass classification. A simpler classification method, binary classification, is used to determine membership in one of two classes. For DDoS detection, it would suffice to label the traffic as either benign or not benign. Doing so provides the advantages of simpler models as well as the ability to detect new attacks before they are recognized by signature-based systems (so-called "zero-day" attacks).

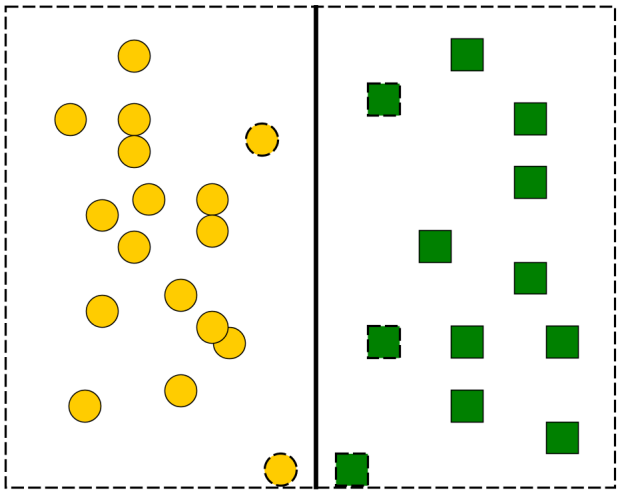

Figure 2.1. A linearly separable set of points.

For many machine learning models, the input data is mapped to some arbitrary n-dimensional space. Figures 2.1 and 2.2 illustrate several properties of data mapped to a set of points in Euclidean space. The set of points in Figure 2.1 can be easily separated by the bold line, called the *separating hyperplane*, into two distinct classes (circles and squares in this example). The points with dashed borders are are the closest to the separating hyperplane. For models like the Support Vector Machine (SVM), these points are considered the *support vectors*. The support vectors provide a "boundary" with which to classify instances from a data set. An important concept with regards to support vectors is *margin*. The margin is the distance from the closest point to the separating hyperplane. In Figure 2.1, the two points at the bottom of the figure are the closest points to the separating hyperplane.

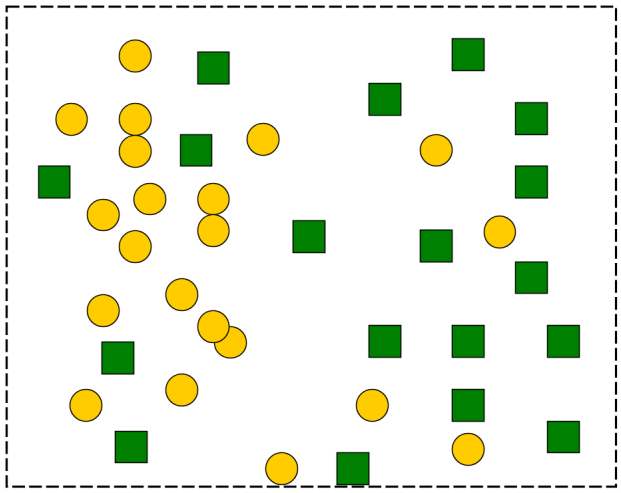

Figure 2.2. A set of points that is not linearly separable.

The points in Figure 2.2 are not linearly separable (at least not to a human observer). However, most classification models have methods to cope with this. The SVM can make use of a *kernel* for points that are not linearly separable. The kernel maps the points from their two-dimensional Euclidean space to some other higher dimensional space so that the SVM can classify the points [15]. The most common kernel used is the Gaussian radial basis function, defined as [16]:

$$k(\overrightarrow{x}) = exp\left(-\frac{||\overrightarrow{x}||^2}{2\sigma^2}\right) \tag{2.1}$$

In the equation, the σ variable influences the number of support vectors chosen for the data set. A smaller σ requires more support vectors and a larger σ requires fewer. The net effect of this is to change the resulting separating hyperplane.

Other examples of classification algorithms include k-nearest neighbors, decision trees, naive Bayes, and gradient descent, to name a few. Of those, the decision tree is of most interest to this research. A brief description of the decision tree follows.

Decision Trees The decision tree is a powerful, yet simple classifer that can be used with numerical data as well as nominal data (such as the data in Table 2.1). Decision trees work by creating rules for splitting nodes based on the features in the data. These rules are analogous to asking a series of yes/no questions on the data. The predicted class of the input data is that of the leaf node once reached.

Several algorithms exist for constructing a decision tree (e.g. ID3, C4.5, CART, etc...). In this research we use the CART algorithm [17]. CART runs in logarithmic time and uses Gini impurity on features to split nodes. Gini impurity is the probability of obtaining two different output predictions for a given input. The Gini impurity of node t for $j = \{1, \cdots, k\}$ possible classes is given by Equation 2.2:

$$1 - \sum_{j=1}^{k} p^2(j|t) \tag{2.2}$$

The tree generated by the decision tree algorithm is easy to understand. This simplicity makes decision trees very popular for many machine learning tasks. Effective attack classification requires that the machine learning model generalize on the data.

It is possible to combine decision trees in order to improve accuracy and robustness. Such a combination is called a Random Forest, and is part of a machine learning technique called *model ensembling*. In an ensemble, the individual decision trees are called estimators.

Two model ensemble techniques exist. The first combines and then averages the results of the individual estimators. This method is called *bagging*, and in the case of decision trees, it is called a Random Forest. The other method, *boosting*, places the estimators sequentially with the goal of improving the performance of the combined estimator. In this chapter, we make use of the Random Forest (bagging) to compare performance with single estimators.

2.1.2 Regression

Regression is used when unknown values need to be inferred from a data set. The predicted values can be continuous, as in the case of linear regression. Other regression techniques exist, including ordinary least squares, weighted, and non-linear regression, to name a few. Logistic regression, which forms the basis of artificial neural networks (discussed in section 3.1.1), is the most applicable to this research and is discussed in the next section.

Logistic Regression Like all regression methods, logistic regression finds the best-fit coefficients for some function. In the case of logistic regression, this function is nonlinear and is called the sigmoid (or logistic) function. The sigmoid function (so-named because of its s-shaped graph) is expressed as the following:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

The output of this function is a real number between 0 and 1 that acts as a probability estimate for the input. Logistic regression imposes a threshold whereby output over 0.5 is classified as 1 and output below 0.5 is classified as 0. This enables logistic regression to take both real-valued and discrete input values and perform binary classification on them.

The input to the sigmoid function is a vector $\overrightarrow{x} = w_0x_0 + w_1x_1 + ... + w_nx_n$. Each $w_ix_i \in \overrightarrow{x}$ is a weight attached to a feature and the weights are determined by some optimization function. The optimization function is chosen based on the objective (minimization or maximization). Several optimization functions exist, however, the gradient descent algorithm is the most commonly used in machine learning.

Based on the input vector $\overrightarrow{x}$, gradient descent finds the set of weights that minimizes the error (or cost) between the features and their correct classification[2] given a set of training data. Once the error has been minimized, logistic regression is then able to classify data from a sample data set. The gradient descent algorithm can be expressed in pseudocode as shown in Algorithm 1.

Algorithm 1: Gradient Descent

```
Set all the weights in the weights vector to 1
while i < maxiterations do
    Calculate the gradient of the entire data set
    Update the weights vector with the current weights added to product of the step size
    and the new gradient
end while
Return the weights vector
```

It is useful to imagine gradient descent performing a search for the function minimum on a two-dimensional surface. For each iteration, a step size parameter (typically denoted α) controls how quickly the gradient moves along the surface and the computed gradient determines the direction that is taken along the surface. The value chosen for α is important; if α is too small the algorithm could produce a value that is nowhere near the minimum since the loop will be exited too early and if α is too large, it may pass right over the minimum. The *maxiterations* value prevents the algorithm from oscillating indefinitely in the event it gets stuck in a local minima.

Gradient descent suffers from performance problems when training on large data sets containing a large number of features. Because it must compute the gradient for *all* the points in the data set *before* updating the set of weights, the convergence time can be quite large. An alternative algorithm, *stochastic gradient descent*, changes this by calculating the gradient and updating the weights on a single instance at a time. Training using stochastic gradient descent can be done in a fraction of the time compared to gradient descent. This

[2]Linear regression, which finds a best-fit line for a set of points, uses the squared error function $\sum_{i=1}^{M}(y_i - (mx_i + b))^2$ for computing the error between the best fit line and the given set of points.

technique of training as new instances arrive is called *online learning* since the set of weights is adjusted incrementally instead of the all at once approach of gradient descent. That is, the values of the weights are retained even as new training data arrive instead of simply throwing them away and starting over.

Though we do not make use of the gradient descent algorithm in this chapter, it is heavily used by artificial neural networks (ANN). We will explore several neural network models in Chapter 3.

All of the machine learning techniques discussed so far share one common shortcoming: they can only classify data given a set of known labels. Sometimes we would like to take a set of unlabeled data (network flows) and give some meaning to it. Unsupervised learning accomplishes this and can even classify unlabeled data using clustering, which is discussed in the next section.

2.1.3 Clustering

Clustering is a type of unsupervised learning that automatically forms clusters (classes) of similar things [15]. Many clustering algorithms exist, but the k-means algorithm is the most relevant to this research and is discussed briefly in the next section.

k-means Clustering The k-means algorithm finds k clusters from a given data set, where the value of k is provided as input. Each cluster is centered around a so-called *centroid*, which is a point that lies at the center of the cluster. Pseudocode for k-means [15] is shown in Algorithm 2.

Importantly, the distance metric used to calculate the distances between the points and the centroids is arbitrary, so any metric space can be used. This also implies the use of numeric values for distances; the development of a distance metric $\in \mathbb{R}$ which maps mostly nominal network flow data values to a metric space is expected to be a crucial part of this research work.

The k-means algorithm differs from the previously discussed learning algorithms since

Algorithm 2: k-means clustering

```
Create k points for initial centroid point assignments
while any point has changed cluster assignment do
  for all points in the data set do
    for every centroid do
      Calculate the distance between the centroid and point
      Assign the point to the cluster with the lowest distance
    end for
  end for
  for every cluster do
    Calculate the mean of the points in that cluster
    Assign the centroid to the mean
  end for
end while
```

finding an optimal solution for k clusters[3] in d dimensions is NP-hard [19]. Several heuristic algorithms exist, such as Lloyd's [20] algorithm. Additionally, dimensionality reduction techniques such as principal component analysis can speed up the convergence time of k-means [21] by retaining only the most important features in the data set.

2.2 Methodology

We explored the use of conventional machine learning approaches on experimental network flow data in order to determine baseline performance as the first step to developing better approaches. In particular, we analyzed the Intrusion Detection Evaluation Data set published by Sharafaldin et al in [22]. This data set consists of 15 different attacks over a week of collection. The data set also includes benign background traffic which mimics typical user behavior. The Intrusion Detection Evaluation Data set is hereafter referred to as the CIC-IDS data set.

We also examined the KDD '99 network intrusion data set [23]. This data set is often used for machine learning applications. The KDD and CIC-IDS data set differ in several important ways:

[3]Determining the correct value for k is NP-hard; when k and d are fixed, k-means converges in polynomial time [18].

- The KDD data consists of *connections* between endpoints and not whole flows
- Much of the KDD data has connections with a duration of zero
- The KDD data has no source or destination Internet Protocol (IP) addresses
- The KDD data has no source ports
- The KDD data has destination ports, but they are the nominal service names instead of numeric values

The most glaring difference between the data sets is that the KDD data set is much older. There is a well known truism that "Attacks always get better; they never get worse." [24] Modern attacks and attack methods have evolved since the publishing of the KDD data set. It is possible that the KDD data set may no longer represent common attacks. We included this data set to show how well machine learning methods work for attack classification in disparate data sets.

2.3 Experiment

Once we cleaned the two data sets, we applied the Decision Tree, Random Forest, and kNN classifiers to the full data sets. We found excellent cross-validated accuracy when we looked at the entire data set. When the data set was segmented by time to study the effect of novel attacks the results were less impressive. We also looked at the effects of removing various features from the data to determine the minimal set of features needed to achieve high accuracy.

2.3.1 Accuracy of Machine Learning

Table 2.2 shows the performance of the decision tree for both data sets. We removed the IP addresses from the CIC-IDS data for two important reasons. First, attackers spoof IP addresses to hide themselves and defeat IP filtering systems. Second, the high variance of the IP addresses "crowded out" the other attributes when applying feature reduction. This

	CIC-IDS			
Split (train % / test %)	Accuracy/Std Dev	Precision	Recall	F1 Score
67/33	97.881/0.010	97.881	97.881	97.881
50/50	97.837/0.033	97.837	97.837	97.837
33/66	97.759/0.026	97.759	97.759	97.759
	KDD			
Split (train % / test %)	Accuracy/Std Dev	Precision	Recall	F1 Score
67/33	99.982/0.0004	99.982	99.982	99.982
50/50	99.981/0.001	99.981	99.981	99.981
33/66	99.973/0.008	99.973	99.973	99.973

Table 2.2. Cross-validated performance on the entire CIC-IDS and KDD data sets. The IP addresses are excluded from the CIC-IDS data set. All scores are percentages.

complicates efforts to find a set of features that best characterize attacks. We applied k-fold cross-validation with $k = 3$ in all cases. The accuracy scores shown are averages taken over the folds. Table 2.3 shows the running time of the decision tree. The running times shown are averages taken over the folds.

Though the results in Table 2.2 and Table 2.3 are appealing, they obscure a problem we found with the data. Because the amount and type of traffic varies by day (for the CIC-IDS data) and by connection (for the KDD data) the data are not stationary. This lack of stationarity means that generalizing on the data is difficult. Traditional machine learning algorithms such as decision trees, support vector machines, and so on will not work without modifications because they expect the underlying data to have a stationary distribution.

To test this, we first trained a decision tree on the CIC-IDS Monday data set. We then checked its accuracy on the rest of the week. We obtained scores of 96.897, 63.524, 99.517, and 71.172 for Tuesday, Wednesday, Thursday, and Friday, respectively. We tested retrospective learning by training on Friday's data. We then checked accuracy on Thursday, Wednesday, Tuesday, and Monday and obtained scores of 95.264, 64.117, 96.811, 99.714, respectively. We obtained similar results when we tested using a random forest with 10 estimators.

We further tested decision tree and random forest generalization on a power set of the days of the CIC-IDS data. The decision tree result is shown in Table 2.4. The random forest result is shown in Table 2.5. The average score is 84.407% and the standard deviation is

	CIC-IDS	
Split (train % / test %)	Train Time/Std Dev	Test Time/Std Dev
67/33	25.259/2.651	0.093/0.002
50/50	16.176/2.234	0.067/0.002
33/66	10.422/1.783	0.042/0.007
	KDD	
Split (train % / test %)	Train Time/Std Dev	Test Time/Std Dev
67/33	10.439/0.797	0.175/0.029
50/50	6.712/1.264	0.120/0.0006
33/66	3.780/0.934	0.0869/0.0134

Table 2.3. Computational performance on the entire CIC-IDS and KDD data sets. The IP addresses are excluded from the CIC-IDS data set. All times are in seconds.

8.530% for the decision tree. The average score is 85.289% and the standard deviation is 9.541% for the random forest.

We also tested classification using the kNN algorithm. Like the decision tree algorithm, kNN is a supervised machine learning model. Many accuracy and speed improvements in anomaly detection have come from the use of kNN for classification [25]. Figure 2.3 provides a comparison between the three models we tested. The left side of the figure shows how the decision tree and random forest compare as a function of kNN. The right side of the figure shows how random forest and kNN compare as a function of the decision tree. Random forest showed better generalization than the decision tree and kNN. The decision tree performed better than random forest. kNN had the worst run time at over 660 seconds on average. This run time is over 9 times longer than the decision tree, which was almost 4 times longer than random forest.

2.3.2 Feature Reduction

Raw network data are quite verbose. Reducing the data to relevant features is critical for obtaining high classification accuracy. This process comprises feature reduction techniques. The idea is to drop features which do not provide adequate variance in the data. The remaining features then provide enough variance to achieve reasonable classification accuracy.

There are benefits and drawbacks to feature reduction. Benefits include simpler models, improved accuracy, and a reduction of the effects of high dimension data. High dimension

		Full Dataset	Reduced Dataset
Training Days	Test Days	Score	Score
Tue	Wed, Fri, Mon, Thu	80.877	78.008
Wed	Tue, Fri, Mon, Thu	88.927	85.649
Fri	Tue, Wed, Mon, Thu	87.455	90.464
Mon	Tue, Wed, Fri, Thu	79.509	79.509
Thu	Tue, Wed, Fri, Mon	80.317	80.163
Tue, Wed	Fri, Mon, Thu	85.924	85.383
Tue, Fri	Wed, Mon, Thu	83.097	88.793
Tue, Mon	Wed, Fri, Thu	75.356	75.082
Tue, Thu	Wed, Fri, Mon	83.761	76.353
Wed, Fri	Tue, Mon, Thu	97.38	94.021
Wed, Mon	Tue, Fri, Thu	85.357	82.477
Wed, Thu	Tue, Fri, Mon	85.525	84.445
Fri, Mon	Tue, Wed, Thu	86.78	84.854
Fri, Thu	Tue, Wed, Mon	84.747	87.733
Mon, Thu	Tue, Wed, Fri	74.943	74.493
Tue, Wed, Fri	Mon, Thu	96.635	95.410
Tue, Wed, Mon	Fri, Thu	81.212	78.537
Tue, Wed, Thu	Fri, Mon	81.822	81.738
Tue, Fri, Mon	Wed, Thu	75.948	79.835
Tue, Fri, Thu	Wed, Mon	79.997	84.532
Tue, Mon, Thu	Wed, Fri	67.535	67.405
Wed, Fri, Mon	Tue, Thu	96.674	93.566
Wed, Fri, Thu	Tue, Mon	98.197	96.038
Wed, Mon, Thu	Tue, Fri	79.075	78.537
Fri, Mon, Thu	Tue, Wed	77.142	80.640
Tue, Wed, Fri, Mon	Thu	92.343	92.620
Tue, Wed, Fri, Thu	Mon	99.958	98.669
Tue, Wed, Mon, Thu	Fri	68.913	69.368
Tue, Fri, Mon, Thu	Wed	80.21	72.858
Wed, Fri, Mon, Thu	Tue	96.613	93.392

Table 2.4. Generalization on the power set of per-day full and feature-reduced CIC-IDS data sets using the decision tree. IP addresses are excluded from the full data set. All scores are percentages.

data suffers from the so-called "curse of dimensionality" [26]. This problem affects machine learning when the data lacks enough samples for combinations of all the features. Removing unnecessary features eases the effects of an insufficient amount of samples. Feature reduction can also enable the use of other machine learning models. Removing unnecessary features can remove the noise from data, but can remove the signal as well. This can cause a close grouping of the data, destroying the variance and reducing accuracy.

Reducing the size of the data set is a practical advantage of feature reduction. Net-

		Full Data set	Reduced Data set
Training Days	Test Days	Score	Score
Tue	Wed, Fri, Mon, Thu	80.820	77.797
Wed	Tue, Fri, Mon, Thu	88.279	86.399
Fri	Tue, Wed, Mon, Thu	89.428	90.706
Mon	Tue, Wed, Fri, Thu	79.509	79.509
Thu	Tue, Wed, Fri, Mon	80.290	80.197
Tue, Wed	Fri, Mon, Thu	86.265	86.010
Tue, Fri	Wed, Mon, Thu	88.634	89.313
Tue, Mon	Wed, Fri, Thu	75.341	75.343
Tue, Thu	Wed, Fri, Mon	76.382	76.361
Wed, Fri	Tue, Mon, Thu	98.760	95.674
Wed, Mon	Tue, Fri, Thu	84.551	83.806
Wed, Thu	Tue, Fri, Mon	87.385	84.629
Fri, Mon	Tue, Wed, Thu	87.852	87.679
Fri, Thu	Tue, Wed, Mon	92.390	88.274
Mon, Thu	Tue, Wed, Fri	74.098	74.512
Tue, Wed, Fri	Mon, Thu	99.686	98.333
Tue, Wed, Mon	Fri, Thu	82.148	80.776
Tue, Wed, Thu	Fri, Mon	83.580	82.038
Tue, Fri, Mon	Wed, Thu	79.568	82.069
Tue, Fri, Thu	Wed, Mon	86.313	84.840
Tue, Mon, Thu	Wed, Fri	67.393	67.393
Wed, Fri, Mon	Tue, Thu	95.620	95.392
Wed, Fri, Thu	Tue, Mon	98.590	96.190
Wed, Mon, Thu	Tue, Fri	78.830	78.508
Fri, Mon, Thu	Tue, Wed	82.956	82.808
Tue, Wed, Fri, Mon	Thu	99.656	97.864
Tue, Wed, Fri, Thu	Mon	99.963	98.879
Tue, Wed, Mon, Thu	Fri	69.378	69.629
Tue, Fri, Mon, Thu	Wed	68.704	71.761
Wed, Fri, Mon, Thu	Tue	96.316	93.523

Table 2.5. Generalization on the power set of per-day full and feature-reduced CIC-IDS data sets using a random forest with 10 estimators. IP addresses are excluded from the full data set. All scores are percentages.

work flow data can easily exceed gigabytes per day, and threat detection requires rapid and accurate response to be useful. Therefore, minimizing the size of the data has important practical effects.

We applied Principal Components Analysis (PCA) [27] and the entropy and information gain metrics from the ID3 algorithm [28] to both data sets and were able to significantly reduce the number of features while retaining the same level of accuracy. Applying PCA can obscure the connection between the reduced data set and the original. PCA can also “load”

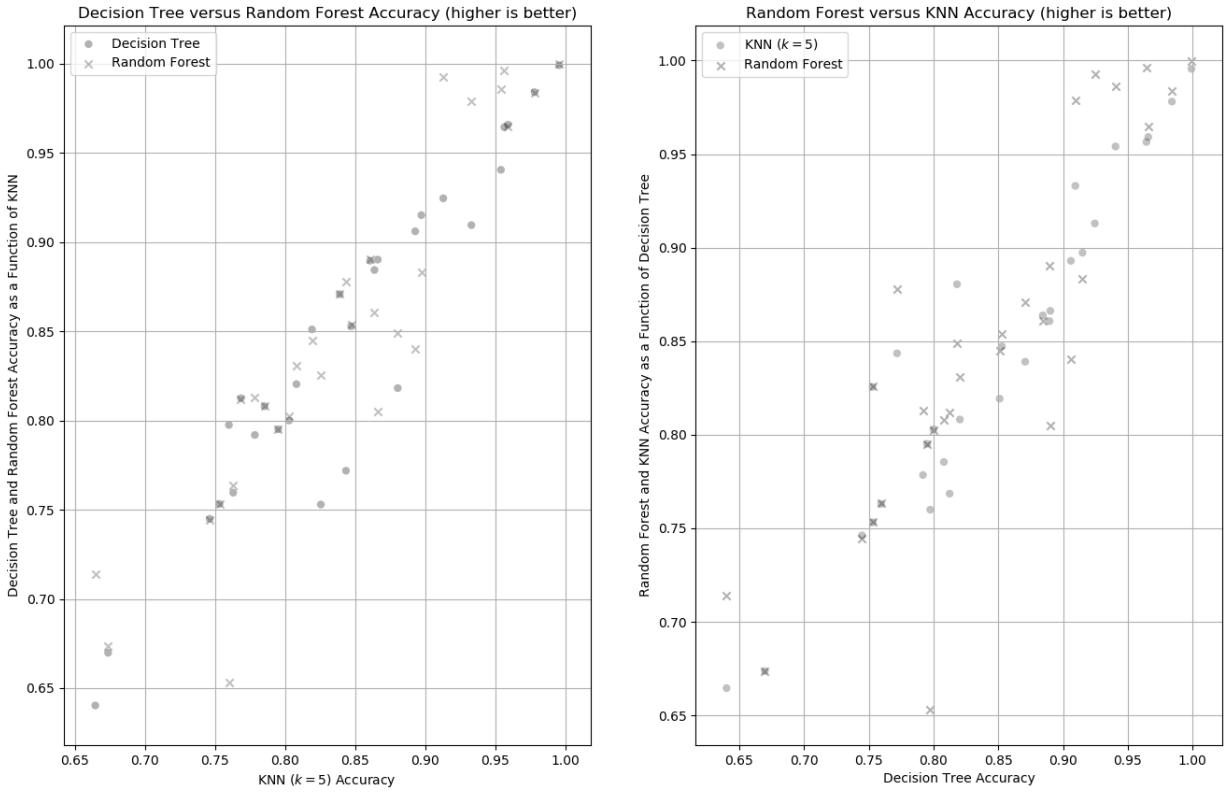

Figure 2.3. Performance comparison for the different models. Random forest performs the best, but only by a small margin.

the first principal component with the highest variance which makes the reduced data set non-optimal. The remaining components thus no longer provide an accurate interpretation of the original data. Measuring information gain is useful for working around these limitations. In doing so, we can arrive at a set of features that are the most characteristic of attacks.

Table 2.6 shows the results of applying the ID3 entropy and information gain metrics to varying sample sizes for both data sets. The KDD data does not show an interesting result since the initial feature chosen is the same for all three sample sizes. Both algorithms select the same initial feature for the first split. The CIC-IDS result varies as the sample size increases until the initial feature is ultimately the same for all data sets.

Figure 2.4 shows the per-feature variance for all five days of the CIC-IDS data. All features have been scaled to the range [0,1] so that the largest value of each feature is scaled to unit size. It is obvious that many features have a small variance and can thus be

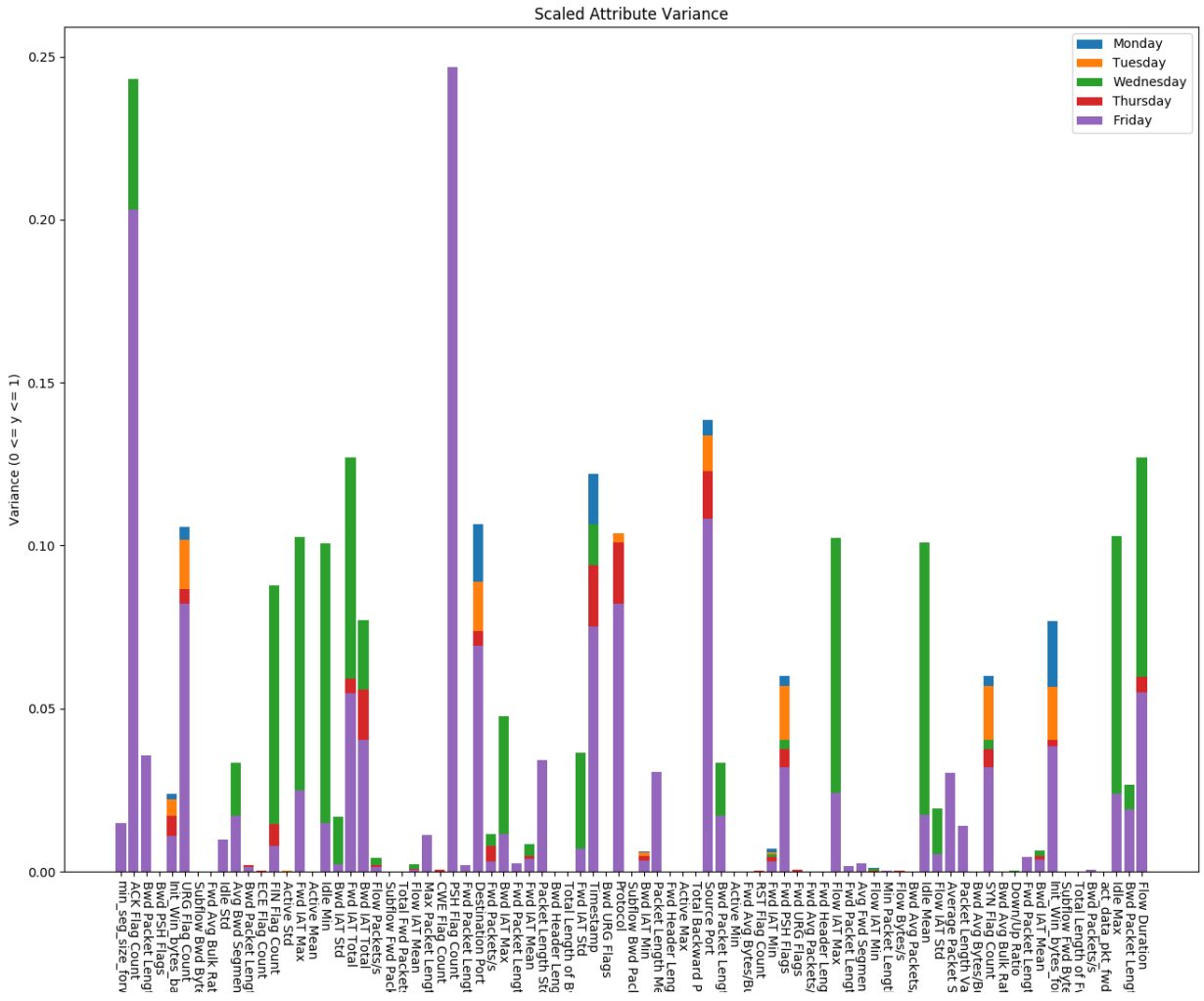

Figure 2.4. The variance of all flow features. All values have been scaled to the range $[0, 1]$.

eliminated. Further feature reduction could be based on a variance threshold or some other criteria.

We applied PCA to the CIC-IDS data and reduced the number of features from 85 to 5. The largest eigenvalue corresponded with the Backward Inter Arrival Time (IAT) Total feature. The second largest was Flow Bytes per second. The third largest was Flow Duration. The fourth largest was Forward IAT Total. The last was Forward IAT Max. We reduced the KDD data set from 41 features to 2. The largest eigenvalues corresponded with the Source Bytes and Destination Bytes features. It is worth noting that the ID3 splitting algorithm always chose Destination Bytes as the best feature to split on in Table 2.6.

Data set	1,000 Samples	5,000 Samples	10,000 Samples
CIC-IDS Monday	Flow Bytes/second	Flow Bytes/second	Flow Bytes/second
CIC-IDS Tuesday	Flow Packets/second	Source Port	Flow Bytes/second
CIC-IDS Wednesday	Source Port	Source Port	Flow Bytes/second
CIC-IDS Thursday	Flow Bytes/second	Flow Bytes/second	Flow Bytes/second
CIC-IDS Friday	Source Port	Source Port	Flow Bytes/second
CIC-IDS Full data set	Fwd Packets/second	Flow Bytes/second	Flow Bytes/second
KDD Full data set	Destination Bytes	Destination Bytes	Destination Bytes

Table 2.6. Feature reduction on the CIC-IDS and KDD data sets using the ID3 entropy and information gain metrics. The given feature for each data set is the initial feature to split on.

We retrained the decision tree on both reduced data sets. Table 2.7 shows the results in a format comparable to Table 2.2. The accuracy on the CIC-IDS data set increased by an average of 0.097% while the accuracy on the KDD data set decreased by an average of 2.79%.

	CIC-IDS			
Split (train % / test %)	Accuracy/Std Dev	Precision	Recall	F1 Score
67/33	98.005/0.020	98.005	98.005	98.005
50/50	97.929/0.014	97.929	97.929	97.929
33/66	97.828/0.032	97.828	97.828	97.828
	KDD			
Split (train % / test %)	Accuracy/Std Dev	Precision	Recall	F1 Score
67/33	97.176/0.044	97.176	97.176	97.176
50/50	97.182/0.031	97.182	97.182	97.182
33/66	97.203/0.048	97.203	97.203	97.203

Table 2.7. Cross-validated performance on the feature reduced CIC-IDS and KDD data sets. The features used in the CIC-IDS data set are Bwd IAT Total, Flow Bytes/sec, Flow Duration, Fwd IAT Total, and Fwd IAT Max. The features used in the KDD data set are Source Bytes and Destination Bytes. All scores are percentages.

We then tested generalization on the reduced per-day CIC-IDS data set for the decision tree and random forest. The rightmost column in Table 2.4 shows the decision tree results. Accuracy decreased by an average of 0.72% with a standard deviation of 9.541%. The rightmost column in Table 2.5 shows the random forest results. Accuracy decreased once again, this time by an average of 0.74% and a standard deviation of 8.728%.

2.4 Conclusions

We've shown that decision trees provide good performance in estimating attacks. Decision trees can yield accuracy as high as 95% [29]. Like all machine learning models, decision trees are prone to overfitting [30]. To avoid this, we applied dimensionality reduction techniques as discussed in Section 2.3.2. The lack of stationarity in the data means that traditional machine learning models will not generalize well. This presents a problem when an attack signal is present only in certain features. When using a decision tree, the result is that features near the top of the tree for one day are lower down for other days. There are two possible approaches to fixing this. The first is to increase the size of the training set to more than a day and retrain the model after some time. This way, the model is aware of malicious traffic and the structure of the tree can adapt. We observed this lack of awareness when the accuracy on the model trained on the Monday data dropped from 96.897% for Tuesday to 63.524% for Wednesday. The cause of this was the fact that Monday's data were all benign and no traffic for Wednesday was benign. We intend to explore ways of determining the optimal size of this training set. The second is to explore models that do generalize well. Based on our previous work in [31], the Restricted Boltzmann Machine (RBM) is a good choice.

Feature reduction provides a powerful means of reducing the data to the features that are most characteristic of attacks. 85 features were reduced to 5 significant ones with this data, which suggests that the data are highly redundant. This process must be applied iteratively to the data. Doing so systematically eliminates high-variance features that do not contribute to the discrimination between benign and malicious traffic.

The results shown in Table 2.7 suggest two possible findings related to our feature reduction efforts. First, the reduced CIC-IDS data set features are close to those most characteristic of attacks. Second, the reduction in features for the KDD data set was too aggressive. It is obvious that both reduced data sets need further feature engineering. For this paper, our goal was to show the effects of feature reduction. We intend to explore

ways of adding useful features without adding redundancy and retaining the performance improvement that comes with feature reduction.

The overall decrease in accuracy shown in Tables 2.4 and 2.5 compared to the increase for cross-validated training on the entire data set (shown in Table 2.7) strongly suggests that the non-stationarity affects the accuracy of the results. The changes of the accuracy in the power set data between the reduced and full data sets are not uniform, sometimes increasing and sometimes decreasing. This suggests a complicated interaction between feature selection and stationarity. It may be interesting to explore this effect on larger and more complete data sets in future work.

PART 3

ATTACK DETECTION AND GENERALIZATION USING DEEP LEARNING

In Chapter 2, we showed that conventional machine learning models are unable to cope with the lack of stationarity in the network flow data. The relationship between time and whether benign or attack traffic is present is a feature that is not explicitly present in the data; that is, this feature exists in the *latent space* of the data. Conventional models are able to learn latent features. However, as we showed in the previous chapter, these models are unable to generalize on the data. Further, even applying feature reduction, a common practice used to speed up training, provided no benefit. We concluded that more powerful models are needed.

Deep learning makes use of Artificial Neural Networks (ANNs) and provides a powerful way to learn from nearly any type of data imaginable. Deep learning models can work with or without the notion of time. Recurrent neural networks (RNN) are a good model if the recent past matters more than the distant past. This is useful when certain DDoS attacks exhibit a "ramp up" period. A Convolutional Neural Network (CNN) is a good model if specific attack attributes are more important than temporality. CNNs learn feature maps, which are translation invariant. Once a CNN learns the features of benign traffic, detecting malicious attack traffic is made easier.

In this chapter, we analyze the DDoS Evaluation Data set published by Sharafaldin et al. in [1] (hereafter referred to as the CIC-DDoS data set). This data set consists of 13 different attacks over two days of collection. We also analyze the Intrusion Detection Evaluation Data set published by Sharafaldin et al. in [22]. This data set consists of 15 different attacks over a week of collection. Both data sets include benign background traffic, which mimics typical user behavior. The Intrusion Detection Evaluation Data set is hereafter

referred to as the CIC-IDS data set.

Time series analysis is an important machine learning problem. When a variable is measured sequentially in time over or at a fixed interval, the resulting data form a time series [32]. Both the CIC-DDoS and CIC-IDS data sets are organized into time series data. Time series data includes trends and seasonality. Trends are increases and decreases of the observed variable over time. Seasonality is the presence of repeating patterns or cycles in the data.

In this research, we found that a variety of attacks could be identified with high accuracy compared to previous approaches. We show that a CNN can be implemented for this problem that is suitable for large volumes of data while maintaining useful levels of accuracy. We also propose a new technique for representing flow data that is suitable as input for a CNN.

The next section provides an overview of deep learning. Readers with a background in deep learning can skip to Section 3.2 if desired.

3.1 An Overview of Deep Learning

All of the previously discussed machine learning algorithms and techniques are considered "shallow learning", since they consist of a single algorithm which is used for classification, regression, or clustering. In contrast, in deep learning, networks of algorithms (typically logistic regression units) are connected in a hierarchical structure where each subsequent layer learns higher-level features from the data set. Several architectures for deep learning exist, such as convolutional neural networks, deep belief networks, and recurrent neural networks, to name a few.

Deep learning networks are functionally similar to existing artificial neural networks, and an overview of artificial neural networks is provided in the following section.

3.1.1 Artificial Neural Networks

Artificial neural networks (commonly referred to simply as "neural networks" or "neural nets" in the context of machine learning) are sets of computational units (neurons) which

form a connected network, similar to the structure of the human brain. A typical artificial neural network is shown in Figure 3.1.

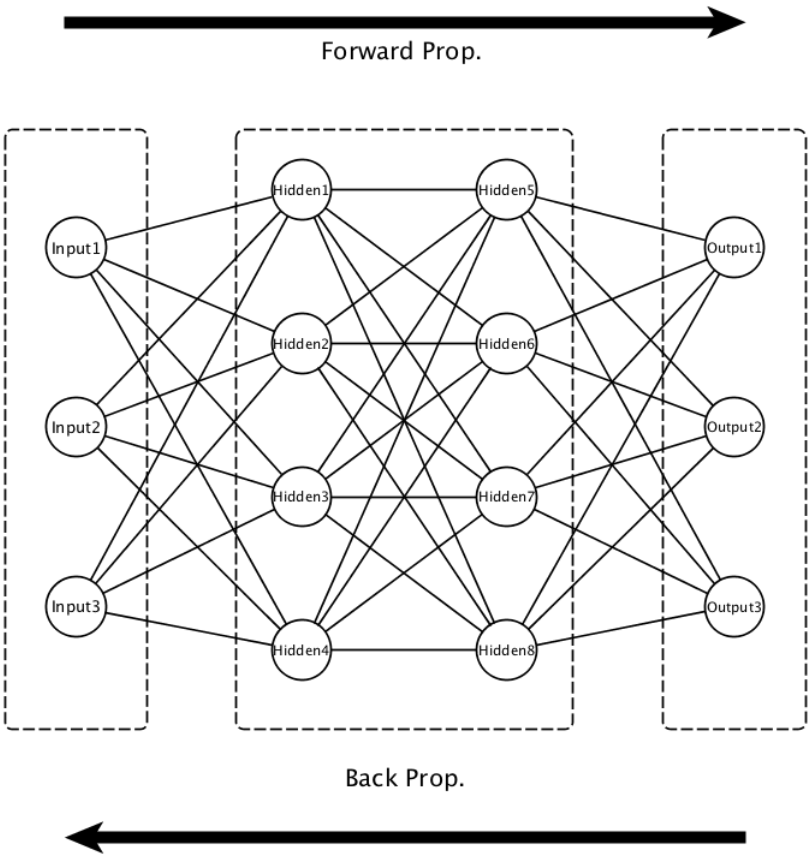

Figure 3.1. An Artificial Neural Network.

Several important properties of neural networks are shown in the figure. First, three layers exist in the network, and are separated by the dashed boxes. These are, from the left of the figure to the right, the *input layer* (sometimes called the "visible" layer), *hidden layer*, and *output layer*. Note that the hidden layer contains two sets of hidden units; in larger networks, it could contain thousands of hidden units. The second property of the network is that no two neurons in the same layer are connected to each other. Thus, connections between layers are bipartite graphs. Lastly, the bold arrows at the top and bottom of the figure indicate the directions taken by forward propagation and back propagation, respectively. Forward propagation is performed when data arrives at the input layer and is "fed" along the network until it reaches the output layer. Back propagation is performed after an error

measurement is taken between the output of the network and the training data set. In this step, the outputs are fed backwards into the network towards the input layer until the hidden neurons arrive at an acceptable error level. This is analogous to gradient descent, but is performed in parallel on all hidden neurons until they can accurately represent the input data.

The perceptron is the most basic unit in a neural network and consists of an input vector, a weight vector, a bias, and a nonlinear activation function. An example of a simple perceptron is shown in Figure 3.2.

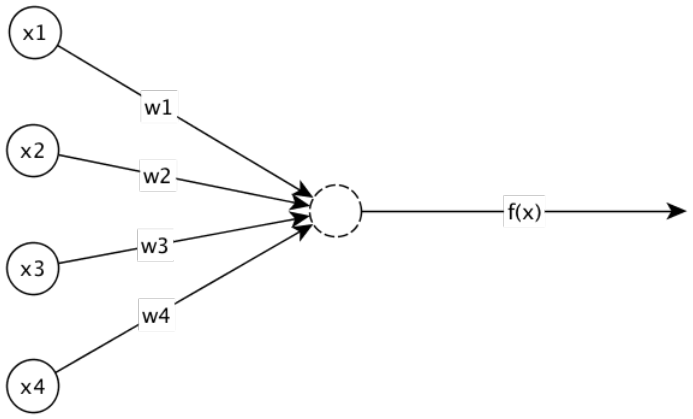

Figure 3.2. A simple perceptron.

In the figure, the set of $x_1, ..., x_4$ denotes the input vector, the set of $w_1, ..., w_4$ denotes the weight vector, and $f(x)$ is the output of some nonlinear activation function, G. Not shown in the figure are the biases attached to the input nodes. These perform the same function as described in Section 2.1.2. The output of the perceptron is either 0 or 1, making it a binary classifier for linear inputs. The perceptron uses the Heaviside step function as an activation function. The perceptron is often referred to as a "single-layer" perceptron to distinguish it from the more commonly used multilayer perceptron. The term multilayer perceptron is a bit of a misnomer - it is not simply several single-layer perceptrons connected together. Instead, an MLP is a network including an input layer, a hidden layer, and an output layer that can perform classification or regression and can use any available activation function. The sigmoid function is the most commonly used activation function and is the

same function used in logistic regression as described in Section 2.1.2. Pictorially, the MLP resembles Figure 3.1.

The model for a single-layer perceptron can be expressed as the following:

$$f(x) = G(W^T x + b) \tag{3.1}$$

In Equation 3.1, G is the activation function (the Heaviside step function), W^T is the transpose of the weight vector, and b is the bias. Importantly, the x variable is a *vector*, not a single value.

3.1.2 Deep Learning Networks

As previously mentioned, several architectures for deep learning exist: convolutional neural networks, deep belief networks, and recurrent neural networks, to name a few.

Convolutional neural networks modify the connectivity shown in Figure 3.1 so that only contiguous neurons from a previous layer are connected to successive layers. This arrangement is called a receptive field [33]. A receptive field of size 2 in Figure 3.1 would then connect neurons **Input 1** and **Input 2** to **Hidden 1**, and **Hidden 1** and **Hidden 2** to **Hidden 5**, for example. The convolutional property of this deep learning network has some application to this research and is expected to be featured in at least one publication.

Deep belief networks are a modification of restricted Boltzmann machines. The typical RBM consists of a visible layer of neurons and a hidden layer of neurons which model the joint distribution of both layers. A DBN modifies this by adding additional hidden layers to the network. These new hidden layers serve as visible layer inputs to successive hidden layers. As shown by Hinton, et. al in [34], these DBNs can more effectively reduce the dimensionality in a set of data compared to techniques like principal components analysis. This is important for this research since network flow data has a large number of features and it is desired to eliminate those that don't contribute to accurate anomaly detection.

Recurrent neural networks are well suited for processing sequential data such as time series data. This makes them a natural fit for processing network flow data, which is inher-

ently sequential and time series-based. RNNs differ from traditional deep learning networks since they do not use the back propagation algorithm. Instead, a feedback loop from the previous layer affects the output of the current layer [35]. This feature is paired with a type of memory called Long Short-Term Memory. This differs from the forward propagation in typical networks since LSTM is dynamic and lives outside the network, whereas forward propagation is a part of the network and produces a static representation of the data [35]. It is expected that RNNs will be a part of this research due to their capability to model sequential data.

3.2 Methodology

In this research, we show that deep learning achieves far greater accuracy compared to shallow models. We show that contemporary deep learning models can be trained on large data sets in a reasonable amount of time. We show that attack detection accuracy improves in some situations when using a per-destination classifier. We introduce a new technique to turn individual traffic flows into images for 2D CNNs. This technique provides a way to better distinguish different types of traffic. This technique helps detect even low rate attacks. Low rate attacks can go undetected by conventional mitigation systems, yet they remain a valid threat. We show that DDOS attacks can be recognized in advance (forward prediction) with robust high accuracy. The general IDS problem still remains difficult, probably because of the complexity of the attacks and the relatively small volume of each IDS attack.

Our goal is not to compare our work with the state of the art in deep learning. Rather, we aim to show that deep learning models significantly outperform shallow models while still performing well enough to handle huge volumes of data. Deep learning models and techniques learn powerful representations of their input data. This includes relationships that may exist outside the explicit feature space. We conclude that intrusion and DDoS detection using shallow models is impractical.

3.3 Experiment

The CIC-DDoS data set contains several comma-separated value (CSV) files spanning two days of collection. The training set consists of traffic captured on January 12th, 2019. The validation set consists of traffic captured on March 11th, 2019. A taxonomy of the attacks present in the data is shown in Figure 3.3.

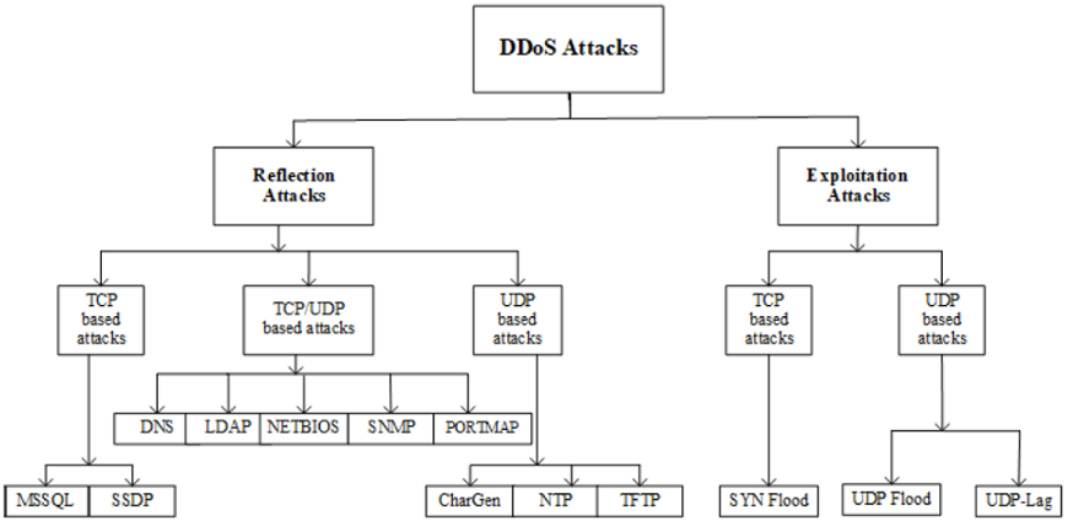

Figure 3.3. A taxonomy of the attacks present in the CIC-DDoS data set [1]

.

A number of issues were present in the data that required fixes before applying machine learning. We dropped two columns, "Unnamed: 0" and "SimillarHTTP", from the data since they did not contain useful information. In some cases, columns such as "Flow Bytes/s" and "Flow Packets/s" contained a few non-numeric values with floating point values for the remaining data. The March 11th "UDP.csv" and "UDPLag.csv" files seemed to be corrupted and were omitted from all testing. We applied similar techniques to clean and prepare the CIC-IDS data set. Since we used binary classifiers, we changed the class name of any attack flows to "attack". The flows with a class name of "benign" were not changed.

We created three different sample data sets with data drawn from both the CIC-DDoS data set and CIC-IDS data set. Each sample data set was further broken down into training

Name	Sample Size	Purpose
DDoS-Random	1,600,000	Random data from all attack types, sources, and destinations
DDoS-Single-Destination	1,600,000	Attacks aimed at a specific target
DDoS-Multiday	2,100,000	Generalization across multiple capture days
IDS-Random	700,000	Random data from all attack types, sources, and destinations
IDS-Single-Destination	645,592	Attacks aimed at a specific target
IDS-Multiday	2,100,000	Generalization across multiple capture days

Table 3.1. Sample data sets for the CIC-DDoS and CIC-IDS data sets. For each source CSV file, 100,000 random samples were taken without replacement.

and testing data sets. The first data set consists of random samples drawn from all collection days with no replacement. The second data set consists of all flows with the most common destination IP address. For the CIC-DDoS data set, the IP address is 192.168.50.1. For the CIC-IDS data set, the IP address is 192.168.10.3. The third data set consists of data from different days. All three sample data sets are intended to assess the generalization power of the deep learning models. The third data set in particular presents a test of temporality. Details of the evaluation data sets are shown in Table 3.1.

3.3.1 Neural Network Details

We created four different neural network models for testing: A feed-forward Artificial Neural Network (ANN), a RNN, a 1-dimensional CNN (1D CNN), and a 2-dimensional CNN (2D CNN). These models are adept at learning different characteristics of the input data. Table 3.2 provides a summary of the common hyperparameters used by all models.

The ANN learns the global characteristics of the entire training data set. That is, specific representations of the data are extracted at each layer. The loss function allows the network to learn which features are relevant. Shallow learning techniques, by comparison, typically require feature engineering before machine learning is applied. Our ANN model is comprised of three densely connected layers. The first two layers have 128 neurons each. The last layer serves as the output layer, where a classification decision is made. The ANN serves as the baseline for performance comparisons. By using the ANN as a baseline, the utility of the other models is established. This justifies the added complexity and run time

of the other models.

ANNs, although simple compared to other models, are still quite powerful for DDoS detection. NG et al. approach the identification of DoS and DDoS attacks by ANN feature learning using the radial basis function [36]. In this approach, errors from layers of ANN are computed on the fixed number of network flow features as an additional regulator on the radial loss function.

Parameter	Value
Batch Size	128
Neurons Per Layer	32-512 depending on model
Optimizer	RMSProp
Hidden Layer Activation Function	ReLU
Output Layer Activation Function	Sigmoid
Loss Function	Binary Cross Entropy
Training Epochs	5
Train/Test data split	66% / 33%

Table 3.2. Hyperparameters common to all models.

The RNN is desirable for sequence classification and time-series prediction. Our RNN model consists of four layers. The first layer is an embedding layer. It embeds the input vectors into a Euclidean space. Similar vectors tend to lie closer while dissimilar vectors are more distant. The next two layers consist of Long Short-Term Memory (LSTM) cells. We compared the performance of LSTM cells to Gated Recurrent Unit (GRU) cells. We were specifically interested in whether the vanishing gradient problem (described in [37]) might appear. We found no difference in performance between LSTM and GRU. The last layer serves as the output layer.

The 1D CNN consists of eight layers. The first layer is an embedding layer similar to the one used by the RNN described previously. The next two layers consist of a 1D convolution layer and pooling layer. The convolution layer uses a kernel size of 5 for generating feature maps. In our testing, kernel sizes between 5 and 7 worked best. Smaller values caused a decrease in accuracy and larger values showed negligible improvements in accuracy. The pooling layer sub-samples the filters produced by the 1D convolution layer. The next two

layers consist of similar convolution and pooling layers. The next layer performs regularization using dropout at a 50% dropout rate. As before, the last layer serves as the output layer.

The 2D CNN consists of twelve layers. The first eight layers are 2D convolution and pooling layers, respectively. The convolution layers use 3×3 kernels for generating feature maps. The pooling layers use a pool size 2×2. The next layer flattens the output of the pooling layer into 1D vectors. The next layer is a densely connected layer consisting of 512 neurons. The next layer performs regularization using dropout at a 50% dropout rate. Finally, the last layer serves as the output layer.

For all models, the sigmoid activation function is used at the last layer for binary classification. All models use the Root Mean Square Propagation (RMSProp) optimizer. The densely connected layers use Rectified Linear Unit (ReLU) activations. The loss function is binary cross entropy. The cross entropy function [38] is defined in Equation 3.2.

$$\begin{aligned} CE &= -\sum_{i}^{N} t_i log(s_i) \\ t &\in \{i, 1-i\} \\ s &\in \{\hat{i}, 1-\hat{i}\} \end{aligned} \tag{3.2}$$

In Equation 3.2, t is the set of ground-truth labels and s is the set of predictions. The sum is over N classes.

3.3.2 Spectrogram-based Flow Representation

Spectrograms provide a visualization of the frequencies contained in a signal over time. The most common way to generate a spectrogram is by using the fast Fourier transform (FFT). The resulting image can be thought of as a heat map of the signal magnitude over time. The heat map provides a visual representation of the frequency space (or spectrum) of the input signal.

The underlying discrete Fourier transform (DFT) provides better output for periodic

signals. This is due to the assumption of periodicity in the resulting basis functions. We first tried generating spectrogram images with no modifications to the flow data. The resulting images were difficult to distinguish. This was due to the small frequency domain represented by the real-valued feature columns in the flow data. We thus introduced artificial periodicity by repeating the flow data for each flow. Introducing artificial periodicity does not expand the resulting frequency domain. Instead, it provides a larger sample space from which to derive the frequency domain. A spectrogram image was generated from the resulting flow. The conversion process is detailed in Algorithm 3.

Algorithm 3: Flow Conversion to Spectrogram Image

```
1  D ⟵ read input flow data
2  train_split ⟵ ⌈0.66 × D_rows⌉
3  test_split ⟵ ⌈0.33 × D_rows⌉
4  F_s = 200.0 /* the sampling frequency                              */
   /* This loop is run on the training and test data                 */
5  for i ⟵ 0 to train_split do
6      row ⟵ D_i
7      S ⟵ smooth the data in row using exponential smoothing
8      S_neg ⟵ flip the signs on the data in S
9      S ⟵ concatenate S and S_neg
10     NFFT ⟵ length of S
11     S' ⟵ repeat S twice
12     S_out ⟵ spectrogram of S' using NFFT data points sampled at F_s
13     Save S_out as an image with a unique file name (a version 4 UUID value) under a
         directory with the flow's label
```

The algorithm works as follows. For each sample data set listed in Table 3.1 the sizes of the training and test data sets are computed. The sampling frequency is set at 200 samples per second. Each row in the training and test data sets has several transformations applied. The data in the row is first smoothed. A copy of the row is made and the signs of each value are flipped to introduce artificial periodicity (i.e. to approximate a periodic signal). The original row and negated row are concatenated together. The sample size (NFFT) is computed from the concatenated result. The concatenated result is repeated twice to create a larger input signal. A spectrogram image is generated on the result. This image is given

a unique file name and saved under a directory named with the flow's class name (either "benign" or "attack").

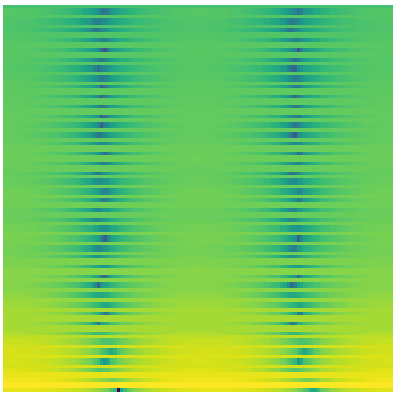

Figure 3.4. A Spectrogram Image of Good Traffic

We generated 150x150 pixel spectrogram images from the sample data sets listed in Table 3.1. Each image represents a single unique traffic flow from the data. A sample of "good" and "bad" traffic flows is shown in Figures 3.4 and 3.5, respectively.

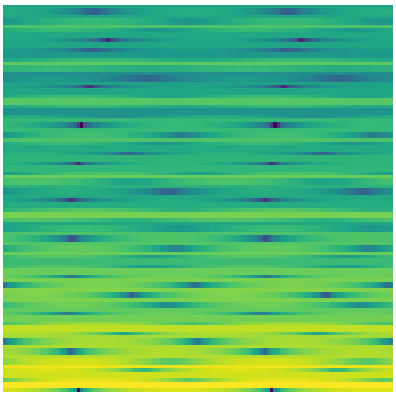

Figure 3.5. A Spectrogram Image of Bad Traffic

Images of the same class showed minor differences in appearance. A few of the images

in different classes showed some common features. Most had very distinct features, an important quality for the 2D CNN. The resulting image data for the CIC-DDoS and CIC-IDS data sets was over 100 Gigabytes in size.

3.4 Results

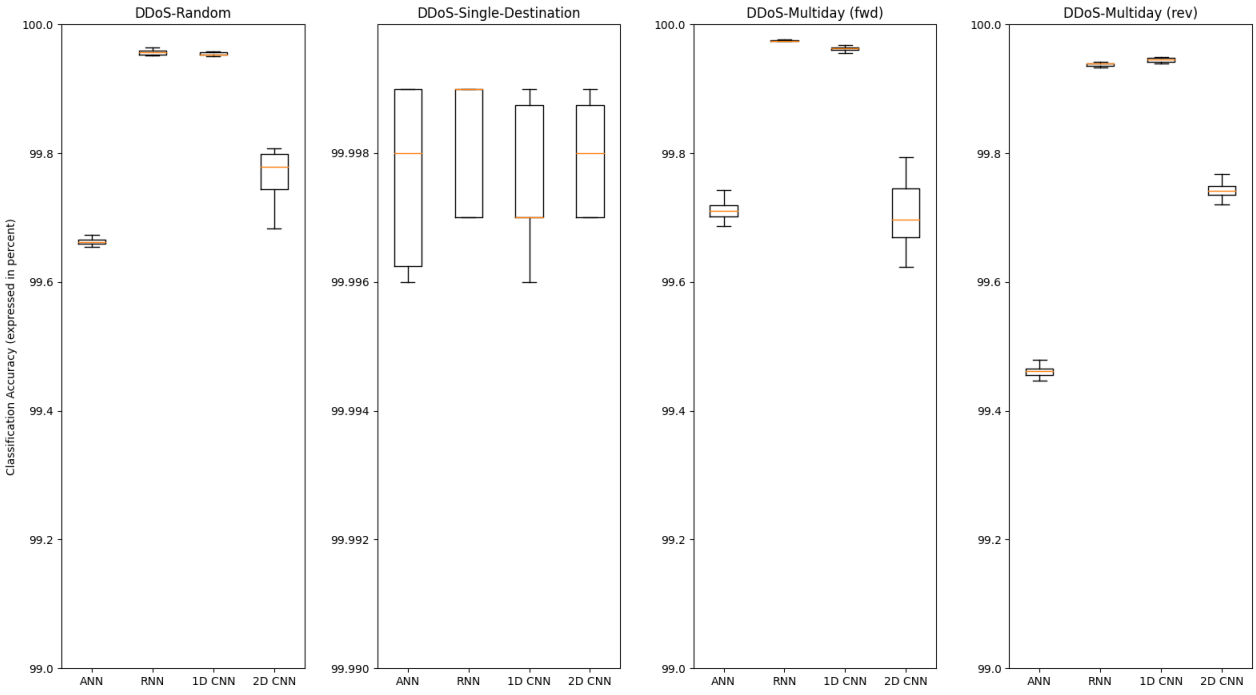

Figure 3.6. A summary of model performance on the CIC-DDoS sample data sets. For the multiday samples, (fwd) refers to predictive learning and (rev) refers to retrospective learning.

We applied the neural networks described in Section 3.3.1 to the text and image sample data sets (as described in Table 3.1) as appropriate. We found that each model began overfitting each of the sample data sets after only a few training epochs. We therefore ran each model for five epochs as a single "iteration". Each model was run for 10 iterations. All models achieved at least 90% training accuracy on all the sample data sets with some achieving 100% accuracy.

A summary of per-model performance on the CIC-DDoS data set is shown in Figure 3.6. A summary of per-model performance on the CIC-IDS data set is shown in Figure 3.7.

The CIC-DDoS results show that the binary baseline accuracy is met and exceeded by the other models, justifying their additional complexity.For the ANN, a global representation is learned. The RNN learns relationships between items in a sequence. Finally, the CNN learns local features (so-called "feature maps"). Our use of more complex models is to determine whether a more granular representation of the data yields better performance. The ANN runtimes are the lowest compared to the other models. This is expected since it is the least complex model.

The CIC-IDS results also show that the baseline accuracy is exceeded by all classifiers. The runtimes for all models are similar to the CIC-DDoS results.

3.4.1 Class percentages

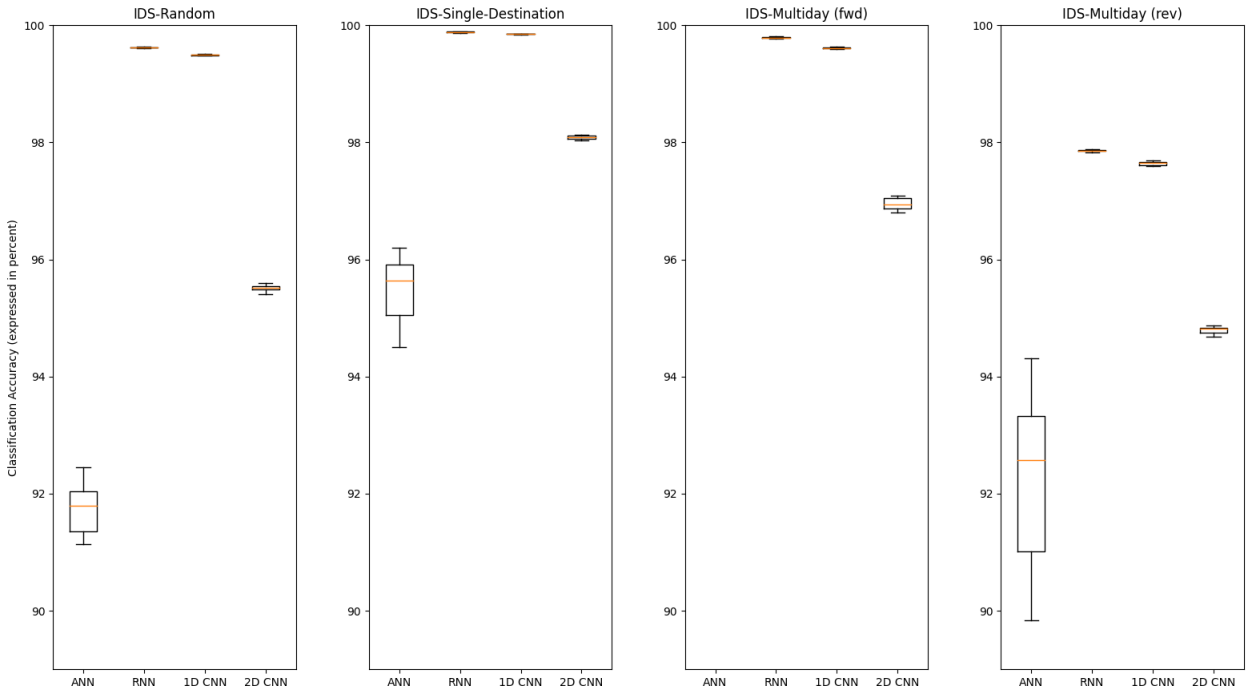

Figure 3.7. A summary of model performance on the CIC-IDS sample data sets. For the multiday samples, (fwd) refers to predictive learning and (rev) refers to retrospective learning.

The performance discrepancies between the CIC-DDoS results and CIC-IDS results were concerning. The same models and methods were used on both sample data sets. We would

thus expect to see similar results.

We investigated the distribution of classes in all of the sample data sets. For the CIC-DDoS samples, the "syn" attack comprises 37.4% of the total traffic, followed by the "netbios" attack at 21%. The percentage of "benign" class traffic (less than 1% of the total traffic) is extremely low relative to the other classes. We conclude from this that the models do not have sufficient training data to learn the "benign" class. Therefore, classification performance suffers.

The class distribution in the CIC-IDS samples is more balanced. The "benign" class comprises 83% of all traffic. This gives all models sufficient samples to create a representation of benign versus attack traffic. The "portscan" attack was the second most prevalent at 10% of the total traffic. Most of the remaining attack classes have low percentages. Others, such as the "web attack" classes, have no samples. These conditions are more representative of real world conditions since they can represent low volume and ramp-up attack traffic.

3.4.2 Accuracy comparisons using the Matthews Correlation Coefficient

Training on the unbalanced data will result in a classifier that is skewed towards the majority class. The Matthews correlation coefficient (MCC) attempts to remedy this [39]. The MCC is a good alternative to F1 scores and ROC curves for comparing training quality [40]. It provides a more concise measure of a classifier's performance compared to a confusion matrix. The MCC value is the normalized determinant of the confusion matrix. The MCC measure is expressed in Equation 3.3.

$$MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{3.3}$$

We captured the MCC values for all the models on all of the sample data sets. The results are shown in Figure 3.8 and Figure 3.9. For the CIC-DDoS data set, the ANN and RNN provide the best classifications, respectively. For the CIC-IDS data set, the RNN significantly outperforms the other models. These results provide a very different perspective compared to the other measures. The class imbalances between the data sets described in

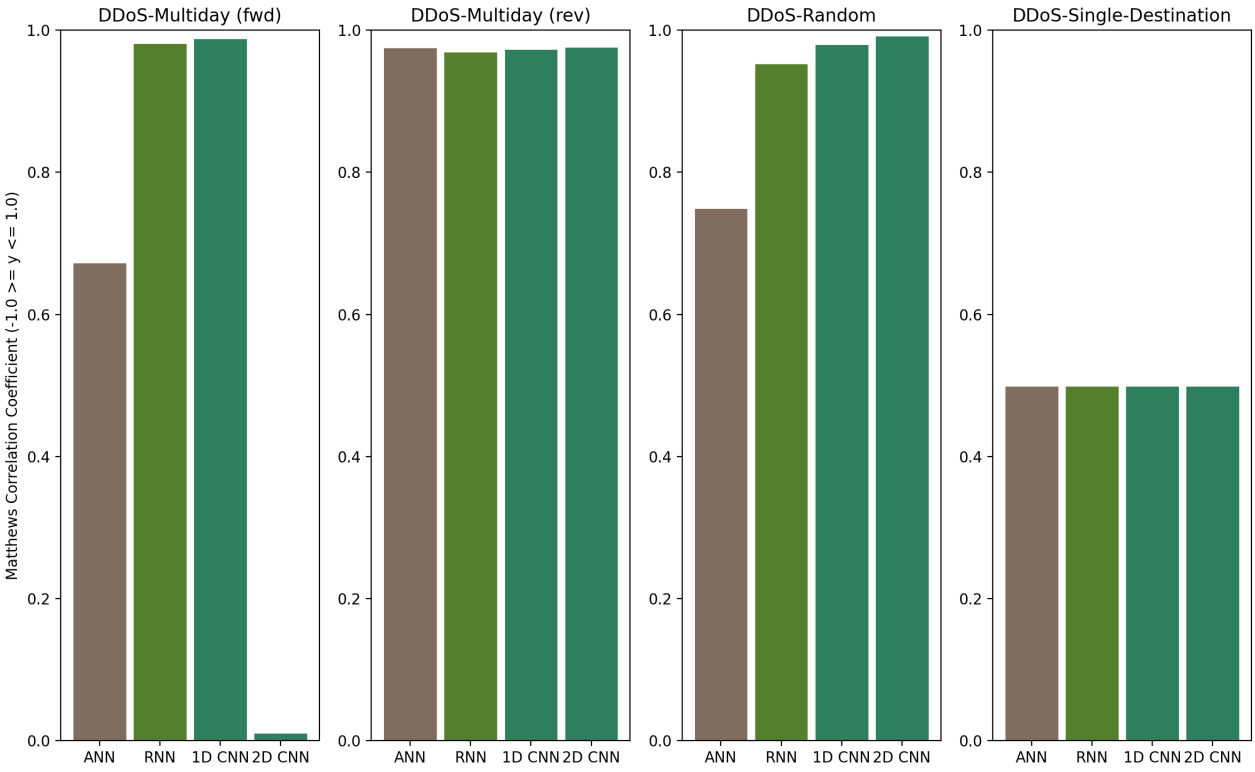

Figure 3.8. Matthews correlation coefficients for all models on the CIC-DDoS sample data sets. Higher values are better. For the multiday samples, (fwd) refers to predictive learning and (rev) refers to retrospective learning.

Section 3.4.1 are less of a factor in the MCC results.

3.4.3 Generalization

How well a given model generalizes to new and unseen data is important. This is especially true for new attack types. The MCC results in Section 3.4.2 provide some insights into how each model performs. Table 3.3 shows more granular per-sample and per-model performance results.

For each data set sample and model, the training accuracy, loss, and runtime are shown. We also captured multiday performance values for each data set (denoted as "M.D."). These values show how well each model handles the temporal nature of the attack data. This aspect of the data is important since network traffic patterns tend to vary based on time and day. The multiday results capture both predictive and retrospective (denoted by *fwd* and *rev*,

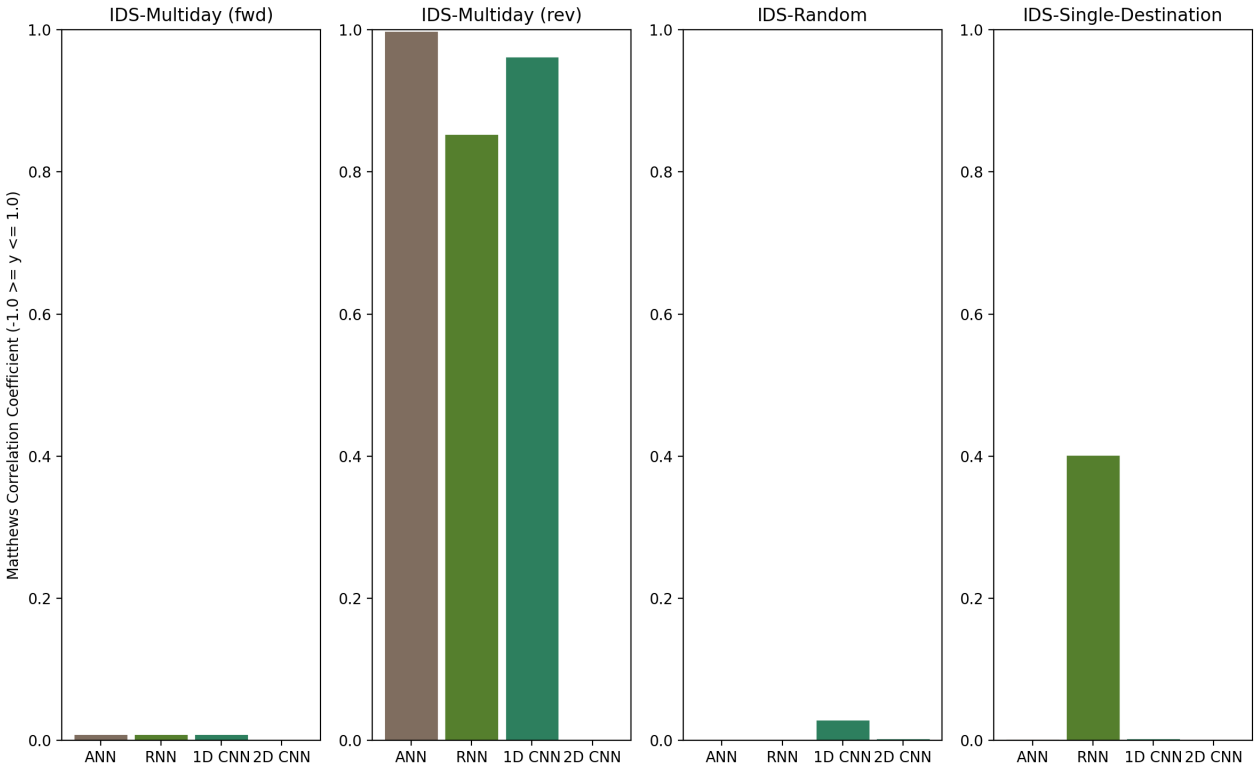

Figure 3.9. Matthews correlation coefficients for all models on the CIC-IDS sample data sets. Higher values are better.

respectively) learning.

For the CIC-DDoS data set samples, the average predictive accuracy for all models is 90.8%. For retrospective learning, the accuracy drops to 85.5%. For the CIC-IDS data set samples, the average predictive accuracy for all models is 91.4%. For retrospective learning, the accuracy is slightly increased to 91.6%. We therefore conclude that the models are capable of generalizing on the data.

These results are far better than our previous work using shallow learning methods [41], where in some cases, accuracy dropped by 30% or more. Dimensionality reduction caused a further loss of generalization accuracy which showed that the effects were due to an inability of the shallow algorithms to capture the essential features of the data.

	ANN			RNN		
Sample Name	Acc.	Loss	Train Time	Acc.	Loss	Train Time
DDoS-Rand	84.8	27	1102	99.4	29	34853
DDoS-S.D.	99.4	26	1060	100.0	0	34954
DDoS-M.D. (fwd)	99.6	49	1440	99.9	0	50593
DDoS-M.D. (rev)	94.0	61	1001	95.7	60	24791
IDS-Rand	82.3	53	469	91.5	51	15172
IDS-S.D.	73.7	34	423	86.9	50	14186
IDS-M.D. (fwd)	83.8	43	1692	88.1	63	50276
IDS-M.D. (rev)	80.5	25	1779	85.7	44	56411
	1D CNN			2D CNN		
Sample Name	Acc.	Loss	Train Time	Acc.	Loss	Train Time
DDoS-Rand	77.3	53	1481	90.6	51	79237
DDoS-S.D.	100.0	0	1482	100.0	10	79671
DDoS-M.D. (fwd)	99.9	21	2021	99.4	15	15712
DDoS-M.D. (rev)	91.5	39	1410	92.6	39	17948
IDS-Rand	91.1	55	629	83.8	30	28941
IDS-S.D.	83.6	31	580	75.9	36	26321
IDS-M.D. (fwd)	87.4	58	2333	85.1	50	49793
IDS-M.D. (rev)	86.9	60	2508	80.8	32	54929

Table 3.3. Performance results for the CIC-DDoS and CIC-IDS sample data sets. Accuracy and loss are expressed in percent. Training time is expressed in seconds.

3.4.4 Low Rate Attacks

Low rate attacks are carried out by sending malicious packets at a much lower rate than a typical high rate attack. The attacker's goal is to carry out the attack while avoiding detection. Attackers use a variety of techniques in low rate attacks. Examples include opening partial connection requests or sending small packets to maintain open connections, to name a few.

We examined the "Flow Packets per second" column present in both the CIC-DDoS and CIC-IDS data set samples. For each attack, we computed the average of the Flow Packets per second. Packets per second (pps) is a good measure of the sending rate since it is independent of the packet size. For the CIC-DDoS data set samples, the NTP attack had the lowest average pps rate at 186,061. All of the attacks in the CIC-IDS data set samples had an average pps value of 1,209,995. Due to this, only the CIC-DDoS data set samples were considered.

Table 3.4 shows the MCC values for the NTP attack for all models and all sample data sets. All models show reasonable performance since there are no negative MCC values. The ANN outperforms the other models, especially with the multiday samples. The RNN provides the second best result in all cases. The 1D CNN provides good multiday performance, suggesting that the NTP attack does not vary significantly over time. The 2D CNN shows average performance. This suggests that while the spectrogram approach works, the NTP images are not distinct enough from the other classes. Further refinement of Algorithm 3 could yield better distinction between classes.

lightgray Data set	Model	MCC
DDoS-M.D. (rev)	ANN	0.666
	CNN1D	0.332
	CNN2D	0.000
	RNN	0.518
DDoS-M.D. (fwd)	ANN	0.993
	CNN1D	0.603
	CNN2D	0.001
	RNN	0.814
DDoS-Rand	ANN	0.197
	CNN1D	0.003
	CNN2D	0.002
	RNN	0.162
DDoS-S.D.	ANN	0.189
	CNN1D	0.000
	CNN2D	0.003
	RNN	0.168

Table 3.4. MCC values for each model and each sample DDoS data set for the low rate NTP attack. Higher values are better. For the multiday samples, (fwd) refers to predictive learning and (rev) refers to retrospective learning.

3.5 Conclusions

We showed that a variety of neural networks provide good classification performance against network attack data. The generalization capabilities of neural networks exceed those of shallow learning methods. Each of the models outperformed the shallow models in our previous work.

Figure 3.8 shows that DDOS attacks can be recognized prior to observing samples of the individual attack. This suggests that there is a common temporal or structural feature to DDOS attacks which can be learned and used to effectively protect networks from DDOS.

Figure 3.9 shows that the general IDS problem is more difficult. While reasonable accuracy can be obtained, shown in figure 3.7, the classifier over predicted IDS resulting in a poor Matthew's correlation coefficient. Still, as shown in figure 3.9 IDS Multiday (rev), the classifiers were able to learn to identify attacks they had seen before.

We introduced a new data representation technique in using spectrograms. The 2D CNN that makes use of these images performed well in most tests. Further refinement of the conversion process detailed in Algorithm 3 could yield additional performance improvements.

CNNs are finding more utility in anomaly detection. In [42], Doriguzzi-Corin et al. also use a CNN for attack detection. The CNN is combined with a data preprocessing algorithm that transforms traffic flows into an embedding space. The embedding space is based on traffic attributes. By transforming the input data, resource utilization and attack detection times are decreased. Their resulting model is called LUCID. Cheng et al. use network flow binary images combined with deep CNNs to predict DDoS attacks in [43]. Creation of the grayscale images used as input to the CNN is based on extraction rules. The result is called a Grayscale Matrix Feature (GMF). Their model obtains high accuracy and low false positive and error rates.

We validated the generalization capabilities of our models using predictive and retrospective learning. We also tried using a bidirectional RNN to achieve a similar result, but the performance difference was negligible. Generalization for all of the neural network models outperformed the shallow models.

Binary classification is better suited for real-world scenarios. Multiclass classification requires that all possible classes are known a priori. Such a requirement limits the effectiveness of the classifier since new or unknown attacks may mimic some properties of known attacks. Additionally, low volume attacks may exhibit characteristics similar to other attacks. As demonstrated in Section 3.4, an imbalance of classes can significantly affect model

performance.

All of the models we tested were sequential neural networks. Other topologies, such as multi-input or multi-output models, could provide additional capabilities. Model ensembling could improve performance. The 1D CNN, whose runtime was only slightly longer than the ANN, could be used to preprocess the data before feeding it to the RNN. In doing so, the higher level features learned by the CNN are fed as input sequences to the RNN. The short runtime for the 1D CNN means that it is suitable for implementation on large volumes of data. Generative learning is another possibility.

The CIC-DDoS and CIC-IDS data sets we tested had very different properties. The most prominent is the class imbalance between them. Most real world networks will have network traffic patterns similar to the CIC-IDS data set. However, it is quite possible that an attack detection framework deployed on live network traffic will encounter an irregular distribution of network traffic that it must cope with. It may be interesting to explore even more powerful models that are capable of handling such unfavorable environments.

PART 4

A FRAMEWORK FOR ATTACK DETECTION

The research presented in Chapters 2 and 3 demonstrates conclusively the feasibility of detecting DDoS attacks using machine learning. In particular, Chapter 2 showed that conventional machine learning models can learn to discriminate between benign traffic and traffic containing a DDoS attack. However, the inability to generalize necessitated the use of deep learning models, which was discussed in Chapter 3.

This chapter takes the knowledge obtained from the previous chapters and attempts to create intelligent threat detection agents. These agents can be integrated into the infrastructure at the network level on up to the application level. In order to distribute the intelligence of the agents, we propose a peer-to-peer network in which agents participate and share threat information.

4.1 Methodology

This research explores the factors required to create intelligent distributed threat detection agents. These agents are capable of performing the roles of the hardware appliances at the application level. They are tightly integrated into the various components of the web application stack. They distribute valuable information on attacks as they are mitigated. The agents integrate modern machine learning into web servers and web applications.

We describe several web-based attacks to provide context to our proposed solution. We discuss two popular Python-based machine learning frameworks. These frameworks can be used to build intelligent threat detection agents. We propose a distributed threat detection model for the contemporary web application stack. Finally, we discuss the potential challenges of such a model.

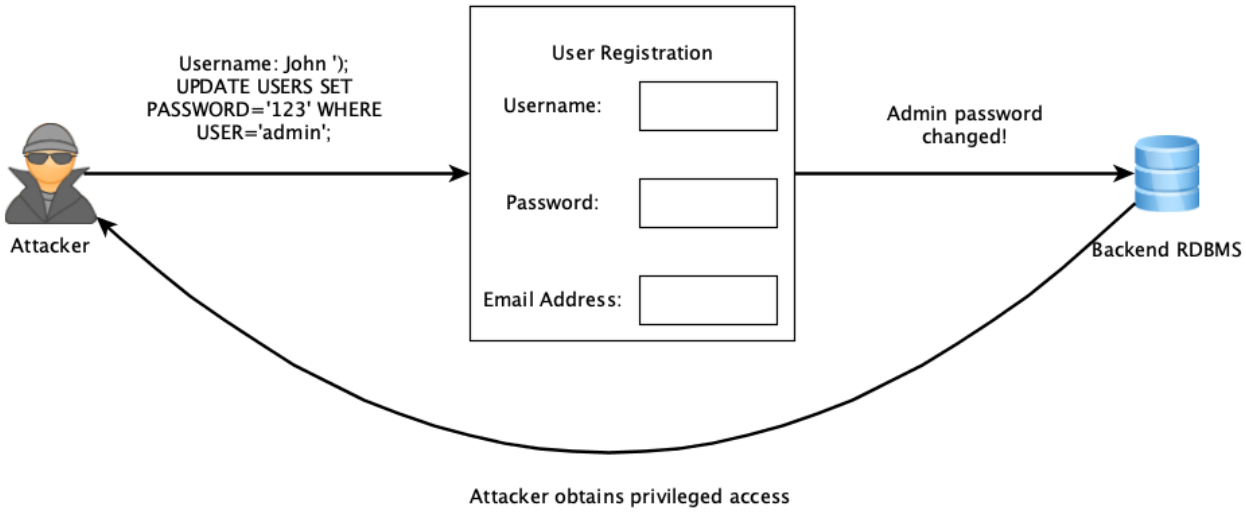

Figure 4.1. A SQL injection attack. The attacker sends a crafted payload and obtains privileged access to the back end database.

4.2 Threat Analysis

It is important to understand common threats to understand the potential use of distributed intelligence. The number of web-based attacks is numerous and growing, so we describe a few common attacks here.

One of the most common web attacks leverages improper or altogether missing form validation in a web application. A number of potent attacks are possible when form inputs are not properly validated [44]:

- Injection of SQL code, which enables an attacker to run arbitrary SQL commands on the backend database of the website
- Cross-site scripting, which enables an attacker to post arbitrary data to a website
- Header injection, which allows an attacker to exploit forms in order to send spam

An illustration of a SQL injection attack is shown in Figure 4.1. To carry out the attack, the attacker includes valid SQL code in the form submission. The result of the attack depends on the payload. The attacker could corrupt the database. Sensitive data

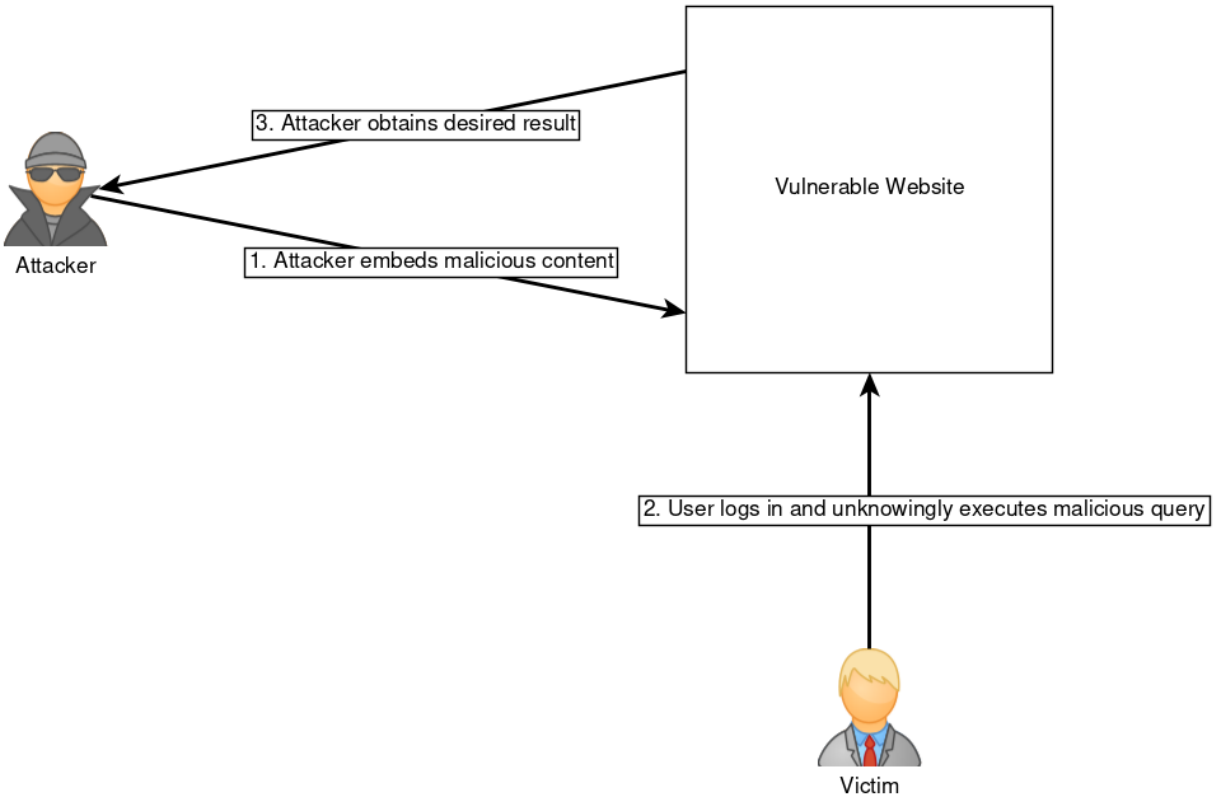

Figure 4.2. A CSRF attack. The attacker embeds a malicious payload into the website. When the user logs in, their web browser automatically executes the malicious query. The attacker obtains the result.

could be stolen. The attacker could gain privileged access to carry out further attacks. Form validation restricts the type of data that can be entered into a form. Input checks ensure the input data is valid for what is being requested in a given form field. Research into the prevention of such attacks is active [45] [46].

Another common attack is a Cross-Site Request Forgery (CSRF) attack. CSRF attacks are a type of confused deputy attack. The forged requests leverage the authentication and authorization of the victim [47]. A CSRF attack adds extra commands to a user's request. The extra commands perform any actions for which the user is authorized. Attacks can change the user's credentials. If the current user has sufficient privileges, other users can be impersonated. These actions are performed without the user's knowledge or consent.

Figure 4.2 shows a generalized CSRF attack. The attacker first embeds a malicious

payload into the website's Hypertext Markup Language (HTML) code. The code could be as trivial as adding a password change request query as the source to an HTML image tag. Once logged in, the victim's web browser fetches the contents of the image tag. The victim's password is changed to the password specified by the attacker in the query. The victim has no knowledge that this has occurred. The victim is a valid user so the password change request appears legitimate from the perspective of the website.

There are many defenses against CSRF attacks [48]. The most common is to embed a secret validation token in any requests. If a request is missing the token or the token does not match the expected value, the server rejects the request.

These two brief examples share a caveat with the hardware appliances mentioned before. Because the mitigations work only with known attacks, any unknown attack is likely to succeed. The use of machine learning can help to mitigate known and unknown attacks.

4.3 Using Machine Learning Frameworks

Machine learning at the application layer can reduce many of the drawbacks of hardware and software solutions. Security can be moved from the network perimeter to host systems. The software solutions described in Section 4.2 gain greater protection. Training application-layer security against application layer threats improves the quality of the decisions.

Host-based application-layer security can substantially reduce license and operational costs for hardware appliances. The physical footprint of the network can thus be scaled down. Throughput scaling becomes less of a concern since host systems perform traffic inspection. Constant signature updates on security devices is reduced or eliminated altogether. However, our proposed model requires updates to traffic that is considered benign. Generative machine learning models such as the restricted Boltzmann machine (RBM) are a solution to this. Application throughput is thus limited to the host system's bandwidth.

Many popular Open Source frameworks exist and are compatible with the application layer. We briefly describe two popular frameworks.

Scikit-learn supports many supervised and unsupervised machine learning models.

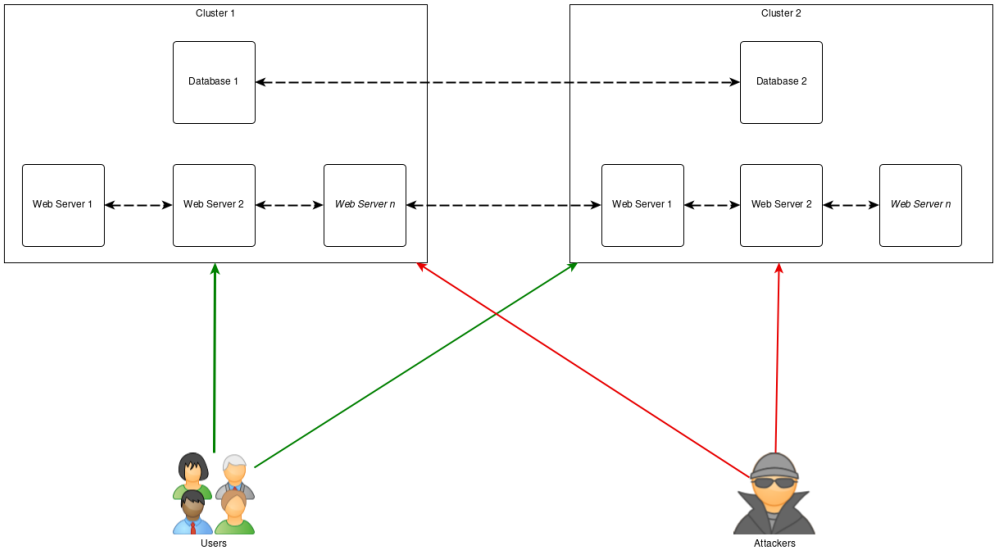

Figure 4.3. An intelligent threat mitigation model for a web application. The dashed lines indicate the distribution of threat intelligence. User traffic is permitted while attacker traffic is not.

Other supporting functionality includes data preparation and model validation. Scikit-learn provides a Python API and is easy to learn. The use of Graphics Processing Units (GPU) is not supported. CPU-optimized versions of the Numpy library can provide some performance gains. Limited support for basic neural networks is available. Neural networks are not the main focus of the library.

TensorFlow was released by the Google Brain team in 2015. In contrast to Scikit-learn, TensorFlow operates with or without a GPU. Using the GPU results in significant speedups in learning. Tensorflow is intended to be a general-purpose numerical computation library. Like Scikit-learn, TensorFlow can execute "shallow learning" tasks. Support for all major neural network models is included.

4.4 A Mitigation Model

Intelligent agents can be incorporated into any component of the web application infrastructure. Figure 4.3 illustrates intelligent agents in multiple layers of the web application infrastructure. The intelligent agents run on each of the components. The dashed lines

indicate the distribution of threat intelligence among the components. Note that there is no inter-component communication. Different components face different threats. Threat detection models thus protect specific components. A compromise of any single component's agents does not affect the agents for other components.

The inclusion of multiple component clusters highlights the distributed nature of the model. This enables a geographically dispersed infrastructure for sharing threat information.

The next sections describe intelligent agents for specific web application components. The use of the Python-based frameworks mentioned in Section 4.3 make the serialization of object instances (i.e. "pickling an object") trivial. Mitigation systems using either of the frameworks could be updated asynchronously while still running.

Training and validation of machine learning models could be carried out in a non-production environment, and the updated models supplied to the applications. It will be necessary to sign the updates to ensure security. Python libraries exist that support modern digital signature algorithms.

4.4.1 Flask

Flask is a Python-based web framework used to build APIs, websites, and more. In the model shown in Figure 4.3, an instance of Flask would run on every web server. Building an intelligent agent using either of the two frameworks in Section 4.3 is trivial. An intelligent agent need only check incoming requests to see if they're potentially malicious. Requests with suspicious content are dropped and an error is returned to the user. This results in just two classes of requests and a linear model on which to learn.

Stochastic gradient descent (SGD) is a simple binary classification model that uses convex optimization as a loss function. SGD outperforms models such as Support Vector Machines (SVM), especially on large data sets. The intelligent agent is first trained using SGD. Once trained, it transforms an incoming request into a feature vector, classifies it, and returns the result. If the feature vector belongs to the malicious class, the request is dropped and an error is returned to the user.

4.4.2 Databases

The database intelligent agent checks incoming SQL queries before running them. Unusual or never-before-seen queries are potentially malicious. For example, if no user management exists in the web application, the agent should never receive queries against such tables. Queries of this type are malicious and are dropped.

Building an intelligent agent for the database component using either of the two frameworks in Section 4.3 would require more effort. The agent would receive incoming SQL queries, classify them, then pass along queries classified as benign to the SQL database. The intelligent agent thus acts as a reverse proxy between the client applications (e.g. the web servers in Figure 4.3) and the database.

The number of possible requests is small relative to those seen by the Flask component. As a result, most database queries are benign. In this case, the use of an SVM is appropriate. As before, the intelligent agent is first trained. Once trained, it transforms an incoming SQL query into a feature vector, classifies it, and returns the result. If the feature vector belongs to the malicious class, the SQL query is not passed along to the database and an error is returned to the user.

4.4.3 Web servers

Web servers are the most outward facing component of the web application. Protecting them against attacks is vitally important. Attacks against web servers tend to focus on the web server itself instead of the content it serves. Fuzzing is a common technique where invalid or unexpected data is provided to an application and the result is observed [49]. Attackers combine fuzzing and known vulnerabilities into attack tool kits. Automated scanners and bots use these tool kits to exploit known and unknown vulnerabilities.

Web servers are typically written in languages such as C to maximize performance. Python-based frameworks such as Flask typically use library functions, written in C and integrated into Python, to optimize performance by speeding up critical bottlenecks to performance. Therefore integrating a Python-based machine learning package, one that also

uses low-level C libraries, into a Python-based web-server is fully viable.

An intelligent agent could either be integrated into the web server software itself or act as a reverse proxy like the database model. An agent acting as a reverse proxy only checks incoming Hypertext Transfer Protocol (HTTP) requests. An integrated agent would need to check incoming network packets as well since attacks might target lower level functionality of the web server.

The attack surface of the web server could be quite large depending on how the agent is integrated. In both cases, SGD would be a good fit for a machine learning model.

4.4.4 Web browsers

The focus of this paper is on the web application infrastructure. We mention web browsers only briefly here to provide a more thorough discussion of the web application ecosystem. A web browser presents a large attack surface on the user's device. This is due to the complexity of modern web browsers. They parse HTML and Javascript code, render audio and visual content, cope with third party extensions, and talk securely to web sites. Most contemporary web browsers include protections for users. While this paper focuses on Python frameworks, machine learning frameworks exist for Javascipt, including TensorFlow [50] and mljs [51].

- Tracking protection prevents a user from being profiled
- Protection against dodgy downloads prevents the user from installing malware
- Blocking pop-up windows protects the user against potentially malicious advertisements
- Disabling third party cookies prevents some user tracking and CSRF attacks
- Private browsing sandboxes the user's session and prevents user cookies from being stolen

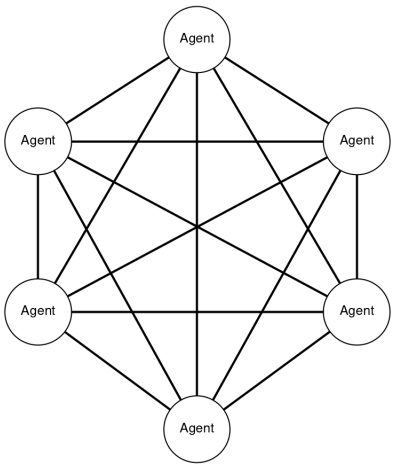

Figure 4.4. A DHT for distributing threat intelligence.

Even with the above protections, web browsers still have drawbacks. The above features mostly guard against known attacks. New attacks can wreak havoc until the browser vendor updates the software. Web browsers offer no protection against potentially malicious content in web pages. A good example of this is the CSRF attack detailed in Section 4.2. A web browser agent would see the unusual content in the image tag and refuse to execute it. Modern firewalls block all communication by default. Trusted communication, defined by policy, may pass through. Web browsers allow all communication by default until the user configures content blocking.

4.5 Distributing the Intelligence

Trained models can be easily distributed using peer-to-peer (P2P) network technology. Modern P2P networks implement a distributed hash table (DHT). This provides robust, fault-tolerant, distributed delivery of resources [52] [53]. However, some environments need strict security controls. In such environments, a private bootstrap host can be used to connect the private peers to each other.

A DHT for distributing threat intelligence is shown in Figure 4.4. The intelligent agents discussed in Section 4.4 are depicted in the figure. The DHT can serve several functions. It can distribute signed trained model data. It can distribute the public keys used to validate

the signed training data. Potential threats that agents have seen but could not correctly classify can be distributed and analyzed.

The types of threats that could be handled using the DHT depend on the placement of the intelligent agents. In the model shown in Figure 4.3, each layer of components runs a separate DHT. In general, any anomalous behavior can be identified and remediated. The default response of the components should be to drop the anomalous traffic. Requests that could not be clearly identified could be quarantined similar to the model seen in email spam filtering. The anomalous request is moved to a secure sandbox until an administrator can identify it. Once identified, the request becomes part of the training data and shared with the agents via the DHT.

4.6 Challenges

From a software engineering and implementation viewpoint, the use of integrated machine learning at the application level is attractive. Several challenges remain.

Obtaining training data, and especially labeled training data, is problematic. A combination of honeypots, monitoring software, and unsupervised classification could be used to find classes of data. The data would then require evaluation to see if it is malicious or benign. Partially trained models could be used to "bootstrap" a system by flagging suspicious, but uncertain, data for further evaluation.

User adoption is another major challenge. It will be necessary to make the addition of the models to existing systems as seamless as is possible and to convincingly demonstrate that the systems provide real benefits to the users. Since, even with advanced machine learning, the problem of deciding what code actually does without executing it, being isomorphic to the stopping problem is formally undecidable, it is critical to tune the machine learning to give a small number of false positives without giving too many false negatives. A system that "cries wolf" too often will inhibit security as users will learn to ignore it.

Another challenge is the validation and update of training models. Distributed and custom training on individual machines may seem an ideal model, but it introduces the

possibility of a "false oracle" attack where the adversary spoofs the machine learning until it ignores real threats while focusing on noise. Therefore, training probably should be separate from the online application so that the training can be supervised and performed in a secure and reliable manner. The models themselves would need to be updated, asynchronously and securely, to the users. ClamAV [54] is an example of an open-source anti-virus program which demonstrates that this is an achievable goal.

4.7 Conclusions

This research has examined the factors needed to integrate machine-learning into application level security. While technical challenges remain, there is no fundamental architectural reason why this could not be done. What remains to be demonstrated is that this can be achieved with a useful level of accuracy while not degrading computational performance.

PART 5

CONCLUSIONS

This research has shown that detecting DDoS attacks using deep learning models and techniques is not only feasible, but can achieve high accuracy with reasonable training times. Importantly, we showed how time affects the accuracy of conventional machine learning models. The lack of stationarity in the data means that more powerful models are needed to achieve good classification performance. This relationship between time and whether the traffic is malicious exists outside of the explicit feature space in the so-called latent space.

We also showed how such an attack detection framework could be applied in real-world settings using intelligent agents and a peer-to-peer network for distributing threat intelligence. These agents can be embedded in any part of the network including into the network devices themselves. By sharing the threat intelligence among agents, the need to retrain individual agents is reduced since the learned representation can be updated as new threats appear and existing threats evolve.

5.1 Future Research

Though many other deep learning models and techniques exist, we wanted to show that contemporary models are more than capable of good performance.

We intend to explore the many unanswered questions this research has generated. One important open question is how the network flow data could be turned into a metric space. Flow data can be quite verbose. Defining a distance function to convert flows into a spatial representation could yield performance improvements and open new research possibilities.

Generative models are useful for sampling from the latent space of the data they're given. Such models could be used for generating new training data and even predicting what new attacks *could* look like.

More powerful models and techniques could improve performance. Multi-input and multi-output models can allow for deeper analysis of the flow data. We alluded to this in Chapter 3 but it's worth repeating here: it's possible to use the 1D CNN as a layer in a larger model. The fast training times mean that a meaningful representation (in the form of feature maps) can be generated by the CNN and used by the larger model.

Word embedding models could be another approach. Word2Vec is a very popular embedding algorithm and could provide a more meaningful representation of the flow data.

The spectrogram-based approach described in Chapter 3 provided some interesting results. The algorithm used to generate the images can certainly be improved. Certainly other spectral analysis techniques could be applied.

Another unanswered question is how well the classifiers described in Chapter 3 hold up against adversarial techniques. This is a crucial issue since the lack of attack signatures means that the classifiers must be robust and general enough to not mistake deceptive inputs as benign so as to fool the model.

Finally, how deep learning for DDoS detection might be applied in real-world settings is yet to be answered. There are many papers in the literature that attempt to answer this. However, the problem remains that any real-world solution must be able to analyze the data at extremely high rates. Network hardware at Internet scale is rapidly expanding in capacity and performance. Contemporary Internet routers make heavy use of interfaces capable of 100 Gigabits per second throughput. This means that a typical Internet Service Provider can potentially have usable capacity more than 10 times greater, or in the Terabits per second range. Further, development of 1,000 Gigabit per second, that is, 1 Terabit per second, network interfaces is proceeding quickly. These speeds suggest that any attack detection models must be embedded into the hardware they run on. This can be accomplished using application-specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs).

PART 6

PUBLICATIONS

This chapter includes publications that I had a role in, but were not related to this research. For each publication, a brief description is provided.

6.1 Fuzzy Restricted Boltzmann Machines

In [31], we developed a deterministic RBM training algorithm and showed how to use that algorithm to automatically derive fuzzy membership classes. RBMs are inherently fuzzy and well suited for situations where only one class is well-formed. We modified several aspects of the conventional RBM. First, we replaced the typical binary values in the visible layer with -1 and 1. Doing so simplifies gradient and energy calculations. Next, we replaced the gradient calculation (typically done using contrastive divergence [55]) with an analytic approximation. This analytic approximation is able to train much faster since it is not iterative.

Fuzzy training extends the standard RBM by adding a belief function to each layer. The layer with the best reconstruction ratio is selected during classification and the most likely value (if discrete) or expected value (if continuous) for the class assignment is returned along with an accuracy estimate.

6.2 Analysis of drug resistance in HIV protease

In [56], we combined dimensionality reduction techniques and generative machine learning (using an RBM) to predict drug resistance profiles from genomic data. Generative machine learning models trained on one inhibitor could classify resistance from other inhibitors with varying levels of accuracy. Generally, the accuracy was best when the inhibitors were chemically similar. Restricted Boltzmann Machines are an effective machine learning tool for

classification of genomic and structural data. They can also be used to compare resistance profiles of different protease inhibitors.

6.3 Evolution of drug resistance in HIV protease

In [57], we evolved the techniques described in the previous paper (namely, developing models for predicting the resistance to single inhibitors) to techniques for predicting the resistance to multiple inhibitors. The previous paper showed that there was significant cross-prediction accuracy where models trained on one inhibitor predict the response to other inhibitors. This suggests that there are commonalities in resistance mechanisms and the first step to studying these commonalities is to build a machine learning model that describes them. This model can then be used to select sequences for expression, characterization, and structural analysis. We developed a method of using minimum spanning trees (MST) to estimate the evolutionary properties of HIV response to drugs.

6.4 Illicit Activity Detection in Large-Scale Dark and Opaque Web Social Networks

In this research [58], we used natural language processing (NLP) techniques to detect illicit activity on the so-called "dark net". We examined conversations on the Telegram network, since many criminal networks exist there. Importantly, we attributed conversations to users even when their "handle" changes (as often happens as a way to try to remain semi-anonymous). We found that we could classify illicit activity, advertisements, and bot activity with high accuracy when using as little as 10% of the words from the corpus. We further tested our model by using it to determine whether a message came from Telegram or Twitter and again obtained high accuracy.

REFERENCES

[1] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1–8.

[2] J. Erickson, *Hacking: the art of exploitation*. No Starch Press, 2008.

[3] S. M. Specht and R. B. Lee, "Distributed denial of service: Taxonomies of attacks, tools, and countermeasures." in *ISCA PDCS*, 2004, pp. 543–550.

[4] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[5] Lucian Constantin, "Attackers use NTP reflection in huge DDoS attack," http://www.computerworld.com/article/2487573/network-security/attackers-use-ntp-reflection-in-huge-ddos-attack.html, 2014, [Online; accessed 6-December-2016].

[6] ICS-CERT, "NTP Reflection Attack," https://ics-cert.us-cert.gov/advisories/ICSA-14-051-04, 2014, [Online; accessed 6-December-2016].

[7] Y. Chen, K. Hwang, and W.-S. Ku, "Distributed change-point detection of ddos attacks over multiple network domains," in *Proceedings of the IEEE International Symposium on Collaborative Technologies and Systems*, 2006.

[8] J. Mirkovic, G. Prier, and P. Reiher, "Attacking ddos at the source," in *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*. IEEE, 2002, pp. 312–321.

[9] A. Saifullah, "Defending against distributed denial-of-service attacks with weight-fair router throttling," 2009.

[10] C.-L. Chen, "A new detection method for distributed denial-of-service attack traffic based on statistical test." *J. UCS*, vol. 15, no. 2, pp. 488–504, 2009.

[11] Q. Niyaz, W. Sun, and A. Y. Javaid, "A deep learning based ddos detection system in software-defined networking (sdn)," *arXiv preprint arXiv:1611.07400*, 2016.

[12] U. Fiore, F. Palmieri, A. Castiglione, and A. De Santis, "Network anomaly detection with the restricted boltzmann machine," *Neurocomputing*, vol. 122, pp. 13–23, 2013.

[13] B. Marr, "Why only one of the 5 vs of big data really matters," *IBM Big Data\& Analytics Hub. Available online at www. ibmbigdatahub. com/blog/whyonly-one-5-vs-big-data-really-matters (last accessed February 29*, 2015.

[14] V. N. I. Cisco, "The zettabyte era: Trends and analysis," *Updated (07/06/2017), https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html*, 2017.

[15] P. Harrington, *Machine learning in action*. Manning Greenwich, CT, 2012, vol. 5.

[16] A. J. Smola, B. Schölkopf, and K.-R. Müller, "The connection between regularization operators and support vector kernels," *Neural networks*, vol. 11, no. 4, pp. 637–649, 1998.

[17] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[18] M. Inaba, N. Katoh, and H. Imai, "Applications of weighted voronoi diagrams and randomization to variance-based k-clustering," in *Proceedings of the tenth annual symposium on Computational geometry*. ACM, 1994, pp. 332–339.

[19] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine learning*, vol. 75, no. 2, pp. 245–248, 2009.

[20] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[21] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 29.

[22] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of fourth international conference on information systems security and privacy, ICISSP*, 2018.

[23] ACM SIGKDD, "KDD Cup 1999 : Computer network intrusion detection." [Online]. Available: http://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data

[24] P. Hoffman and B. Schneier, "Attacks on cryptographic hashes in internet protocols," Internet Requests for Comments, RFC Editor, RFC 4270, November 2005, https://tools.ietf.org/html/rfc4270#section-6. [Online]. Available: https://tools.ietf.org/html/rfc4270#section-6

[25] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection1," *Computers & security*, vol. 21, no. 5, pp. 439–448, 2002.

[26] M. Köppen, "The curse of dimensionality," in *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*, vol. 1, 2000, pp. 4–8.

[27] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[28] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[29] M. Zamani and M. Movahedi, "Machine learning techniques for intrusion detection," *arXiv preprint arXiv:1312.2177*, 2013.

[30] M. Bramer, "Avoiding overfitting of decision trees," *Principles of data mining*, pp. 119–134, 2007.

[31] R. W. Harrison and C. Freas, "Fuzzy restricted boltzmann machines," in *North American Fuzzy Information Processing Society Annual Conference*. Springer, 2017, pp. 392–398.

[32] A. V. Metcalfe and P. S. Cowpertwait, *Introductory time series with R*. Springer, 2009.

[33] LISA Lab, "Convolutional Neural Networks (LeNet)," http://deeplearning.net/tutorial/lenet.html, 2010, [Online; accessed 9-January-2017].

[34] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[35] Skymind, "Tutorial: Recurrent Networks and LSTMs," https://deeplearning4j.org/recurrentnetwork, 2016, [Online; accessed 9-January-2017].

[36] B. A. NG and S. Selvakumar, "Deep radial intelligence with cumulative incarnation approach for detecting denial of service attacks," *Neurocomputing*, vol. 340, pp. 294–308, 2019.

[37] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[38] K. P. Murphy, *Machine learning: a probabilistic perspective*. The MIT Press, 2012.

[39] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[40] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, no. 1, p. 6, 2020.

[41] C. B. Freas, R. W. Harrison, and Y. Long, "High performance attack estimation in large-scale network flows," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5014–5020.

[42] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincon, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for ddos attack detection," *IEEE Transactions on Network and Service Management*, 2020.

[43] J. Cheng, Y. Liu, X. Tang, V. S. Sheng, M. Li, and J. Li, "Ddos attack detection via multi-scale convolutional neural network," *CMC-COMPUTERS MATERIALS & CONTINUA*, vol. 62, no. 3, pp. 1317–1333, 2020.

[44] Syronex, "Why is Form Validation Needed?" [Online]. Available: https://formsmarts.com/form-validation

[45] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "Preventing input validation vulnerabilities in web applications through automated type analysis," in *2012 IEEE 36th annual computer software and applications conference*. IEEE, 2012, pp. 233–243.

[46] W. Xu, S. Bhatkar, and R. Sekar, "Practical dynamic taint analysis for countering input validation attacks on web applications," *Technical Report SECLAB-05–04, Department of Computer Science*, 2005.

[47] I. Muscat, "What is cross-site request forgery?" Jun 2017. [Online]. Available: https://www.acunetix.com/blog/articles/cross-site-request-forgery/

[48] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 75–88.

[49] P. Oehlert, "Violating assumptions with fuzzing," *IEEE Security & Privacy*, vol. 3, no. 2, pp. 58–62, 2005.

[50] Google, "Tensorflow for javascript," accessed: 2019-05-26. [Online]. Available: https://www.tensorflow.org/js

[51] M. Zasso, "Machine learning and numerical analysis tools in javascript for node.js and the browser," accessed: 2019-05-26. [Online]. Available: https://github.com/mljs

[52] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[53] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[54] ClamavNet, "ClamAV is an open source antivirus engine for detecting trojans, viruses, malware & other malicious threats," accessed: 2019-05-26. [Online]. Available: https://www.clamav.net/

[55] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[56] S. D. Pawar, C. Freas, I. T. Weber, and R. W. Harrison, "Analysis of drug resistance in hiv protease," *BMC bioinformatics*, vol. 19, no. 11, pp. 1–6, 2018.

[57] D. Shah, C. Freas, I. T. Weber, and R. W. Harrison, "Evolution of drug resistance in hiv protease," *BMC bioinformatics*, vol. 21, no. 18, pp. 1–15, 2020.

[58] D. Shah, T. Harrison, C. B. Freas, D. Maimon, and R. W. Harrison, "Illicit activity detection in large-scale dark and opaque web social networks," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 4341–4350.