Improving Detection of Malicious Network Traffic Using Machine Learning and Expert Systems

By

Samuel Carroll

A thesis submitted to the Graduate Division
in partial fulfillment of the requirements for the degree of

Master of Science in Computational Sciences and Robotics

South Dakota School of Mines and Technology
Rapid City, South Dakota

Defended: July 3, 2018

Approved by:

Major Professor — Mengyu Qiao, Ph.D., Department of Mathematics and Computer Science	Date
Graduate Division Representative — Jon J. Kellar, Ph.D., Department of Materials and Metallurgical Engineering	Date
Committee Member — Christer Karlsson, Ph.D., Department of Mathematics and Computer Science	Date
Head of the Mathematics and Computer Science Department — Kyle L. Riley, Ph.D.	Date
Dean of Graduate Education — Maribeth H. Price, Ph.D.	Date

Abstract

In breach detection and threat mitigation there are certain indicators of compromise that may indicate a threat actor has breached a network. This threat actor may be either currently launching attacks or gathering information for an upcoming attack. Some of these indicators can be scans and beacons.

By detecting scans and beacons a system administrator can make a network more secure. This is achieved by limiting the actions a threat actor can make. However a single system will have over 64,000 ports to watch, a network will have hundreds of thousands or millions of potential breach points. This means people are unable to watch every possible attack avenue, and require an expert system to evaluate. These systems can take time to evaluate attacks, the more time it takes to remediate a breach the more that breach costs.

A study is presented, to classify network traffic gathered using Bro Network Analyzer and labeled using an expert system. This labeled data will be used to train various machine learning classifiers to aide the expert system in anomalous network traffic detection. The expert system that is used to label the network data was the RITA program developed by Active Countermeasures (ACM).

To accomplish this it will be necessary to test various classifiers against each other to determine the strength of each for labeling anomalous network traffic. The classifiers will be compared in accuracy and speed for labeling. A single machine learning suite should be used to compare each classifier.

This study will also build and test a random forest classifier in GoLang to ensure that the classifier will maintain its accuracy and speed. This is important because GoLang will be used to integrate with the expert system later. This classifier should check how well the classifier catches both anomalous and normal data.

Building a random forest in GoLang, guided by expert system labels, will allow a system administrator to check potential threats as they come in and reduce breach detection latency significantly. This method remains in the 90% range for detecting anomalies and an overall accuracy of about 80%. Because a random forest has a relatively fast run time it should be possible to reduce the detection time of anomalies further (possibly even to real time).

Acknowledgments

I would like to thank my family, friends and all the professors who have endured my complaints over this process. I would not be here without you.

Special thanks to Лиза Woody for LaTeX help, remember ∼∼C==B.

I would also like to thank Geddy Lee, Alex Lifeson, and Neil Peart for being companions unobtrusive.

Additional thanks to the UCI Machine Learning Repository for the wine dataset used to develop the Random Forest code and to the team who created the CTU-13 Dataset.

Many exuberant thanks to John Strand, Paul Asadoorian, Chris Brenton, Ethan Robish and the entire development team at Active Countermeasures.

Finally, to my thesis committee (Dr. Mengyu Qiao, Dr. Christer Karlsson, and Dr. Jon Kellar) for your guidance and advice.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

A	Set of boolean features (including association rules)
ACM	Active Countermeasures
$ANOVA$	Analysis of Variance
$BHIS$	Black Hills Information Security
$C2$	Command and Control
CA	Classification Accuracy
C	The set containing all classes
c	The Count of classes
CF	Click Fraud
CSV	Comma Separated Values
CTU	Czech Technical University
D	The set containing all features for a piece of data
d	The number of dimensions a piece of data has (feature dimension)
D_b	The set of boolean features for data
$DDoS$	Distributed Denial of Service
DoS	Denial of Service
FF	Fast Flux
FP	False Positive
FPR	False Positive Rate
FN	False Negative

FNR	False Negative Rate
$HTTP$	Hypertext Transfer Protocol
$HTTPS$	Hypertext Transfer Protocol Secure
$ICMP$	Internet Control Message Protocol
IDS	Intrusion Detection System
IOC	Indicators of Compromise
IRC	Internet Relay Chat
kNN	k-Nearest Neighbors
ML	Machine Learning
N	Set of all data elements
n	Number of all data elements
N_v	Subset of elements that goes to a child node
p_v, q_v	The v^{th} attribute of elements p and q
$p(i\|t)$	Number of elements of class i at node t
$P2P$	Peer-to-Peer
PS	Port Scan
RDP	Remote Desktop Protocol
RF	Random Forest
$RITA$	Real Intelligence Threat Analytics
ROC	Receiver Operator Characteristic
SVM	Support Vector Machine
t	Number of trees in a forest
TN	True Negative
TNR	True Negative Rate or Specificity
TP	True Positive

TPR	True Positive Rate or Sensitivity
UDP	User Datagram Protocol
w	Maximum height for a tree
$\mathbf{w} \cdot \phi(\mathbf{x}) + b$	Linear decision boundary
	(note $\mathbf{w} \neq w$)
λ_i, λ_j	The regularization parameters for an SVM
$\phi(\mathbf{x})$	Transformation to a space where a decision boundary is linear

List of Keywords

Random Forest

Support Vector Machine

k-Nearest Neighbors

Big Data Analytics

Machine Learning

Data Mining

Network Analytics

Bro Network Analyzer

Threat Hunting

Expert System

Dedication

For my goddaughter Margaret Day Carroll

and in loving memory of my grandfather Lee Marlin Peterson.

Chapter 1

Introduction

Computers and computer systems continue to become more prevalent in our society, and more connected. Today there are more computer systems than ever before, and more avenues for attack. Minihane et al. (2017) found that in quarter three of 2017 there was an all time high of new malware created. Meaning, threat actors are generating malware much more quickly than defenders can find ways to analyze and detect these new attack vectors.

To look at all the different attack vectors a threat actor might use is a very large domain space to evaluate. For the purposes of this thesis only port scans and beacons are evaluated, collectively called 'anomalous traffic'. These are important and representative attacks to evaluate because they can often be early warnings to a larger, impending attacks. It is proposed that machine learning can detect these attack vectors when they are logged using Bro Network Security Monitor and labeled with RITA. One thing to keep in mind is a threat actor will attempt to 'cruise under your radar' by adding noise to these attack vectors. For example, a treat actor will not likely use the same list of ports in their scan of an organization every time they scan. Instead they will scan a range of ports on a system, and then use a slightly different range of ports on another system owned by the same organization. Beacons can often be detected by a pulse, or a call back to a remote system every predefined time units. The threat actor can instead send the signal after a variable wait time, and use this noise to throw off the 'pulse' of their beacon. Instead of sending a message to the threat actor's machine every fixed length of time (e.g. 30 minutes) the compromised system will likely add or subtract a random length of time to an average 'fixed' value.

A human will likely make connections at erratic or random intervals. A deeper look into these attacks is found in Section 2.2. For the purpose of this thesis, noise simply refers to the variance in the list of ports scanned and the difference of the time a beacon sends its pulse.

This thesis has the following structure. First a deeper look into the background of the problem, followed by similar work done to solve this problem (both in the private sector, academic sector, and how this thesis differs). This is followed by an overview of the data used, including basic testing data, the actual data used for our testing, and the preprocessing steps taken to get the data ready for classification. After the data overview, a look into the classification algorithms used is explored (including the theory of how they work and what each is good at) and the measures used to determine if a classifier works well. After that chapter is a look into the experiments run and their results. Including classification accuracy and runtime to determine if the three algorithms compared in Orange would be good to transfer to Go in order to further experiments. This is followed by a conclusion of the test results and a discussion on concerns readers may have. Finally the future of this project is explored.

Chapter 2

Background of the Problem

As the world moves more toward Internet-based computing, commerce, entertainment, and communication the need to develop, and implement, better security measures remain essential for researchers, programmers and system administrators. Lee and Lee (2016) argued "prevention systems alone are insufficient to counter focused human adversaries who know how to get around most security and monitoring tools". In light of Lee's statement one realizes while some security is achieved using signature-based and anomaly-based detection, it does not work alone. There may still lurk a mysterious, invisible entity inside an organization's network. Lee and Lee (2016) go on to say "[threat hunters] are actively searching for threats to prevent or minimize damage." However, analysts "need search and visualization tools to help..." effectively hunt.

Black Hills Information Security (BHIS) and Active Countermeasures (ACM) have built the Real Intelligence Threat Analytics (RITA) platform specifically to aid in this threat hunting problem domain. RITA can be found at BHIS's website at https://www.blackhillsinfosec.com/projects/rita/ and is maintained on GitHub at https://github.com/activecm/rita. RITA preforms daily analysis of a network to inform a treat hunter or hunt team where to look for a potential breach of security. Using daily Bro logs, RITA, and indicators of compromise (IOCs) to determine if traffic into and out of a network is a potential threat to that network. Before diving too deep into how this expert system can help and before getting too deep into the problem, it is important to investigate existing solutions to prevent a threat actor from using a system or detecting that threat actor inside the system, as well as some

attack vectors a threat hunter might use to gain access or launch against a system.

2.1 Detecting Network Intrusions

One of the most common ways malicious traffic is found on a computer system is by an intrusion detection system (IDS). An IDS will detect, and log malicious traffic as well as report it to a system administrator. IDSs have two main methods of detection, signature-based and anomaly-based (Albayati and Issac 2015). Both methods have advantages and disadvantages compared to the other. Also some IDSs will attempt to prevent malicious or anomalous traffic. Remember Lee and Lee (2016) stated that alone these systems are not enough to prevent more sophisticated attacks from threat actors.

2.1.1 Signature-Based IDS

A signature-based IDS uses what Albayati and Issac (2015) call "a signature database" in order "to detect suspicious activity, where each signature represent a print of [a] known attack". One easily spotted problem with a signature-based IDS is the need for a database that is maintained and updated regularly. The maintenance of this database of signature takes time, money and can limit the accuracy of the IDS based on the thoroughness of its content. However, it will have a lower false-positive rate than an anomaly-based IDS. (Albayati and Issac 2015)

2.1.2 Anomaly-Based IDS

Albayati and Issac (2015) explained that an anomaly-based IDS works by building a 'normal' profile of network traffic, and any deviation from this behavior is flagged as a potential attack. The limitation of the anomaly-based IDS is the high false positive rate, but it does have an advantage over the signature- based IDS because it can

detect "non-trivial attacks" (attacks that do not conform to the signatures in the signature database).

2.1.3 Intrusion Prevention System

An IDS might also take steps to prevent the traffic it detects as malicious by preventing the traffic to go through the system, in this case the IDS is called an intrusion prevention system (IPS). If the malicious connection is ongoing, the IPS would attempt to stop the connection after it has been going for a little. Again this are just an additional function that some IDSs implement, though it is not required of an IDS, these systems should be noted too.

2.2 Scans and Beacons

As mentioned in Chapter 1 there are a lot of different attacks vectors a threat actor might launch against a system, this section will have a brief look at a some of those attack vectors and some possible IOCs that are important for this thesis.

2.2.1 Scans

Scans (sometimes port scans) may be an indicator of compromise, they do exactly what the name suggests. Normally scans are done by a program that sends a request to multiple ports to see if those ports are open for use. If a port responds affirmatively it is open and can be used by the user. Notice, in this case it is the user and not a threat actor, because there are times when a scan can be done for legitimate reasons (e.g. a network administrator seeing if the ports are in accordance with policy). In Figure 2.1, there are a lot of connections on ports 80 and 443, this is not surprising since those ports are the defaults for HTTP and HTTPS connections. (Fielding and Reschke 2014) It also shows connections to other ports, as it may be necessary to use those ports, this is normal behavior. In Figure 2.2 it is easy to see a large number of

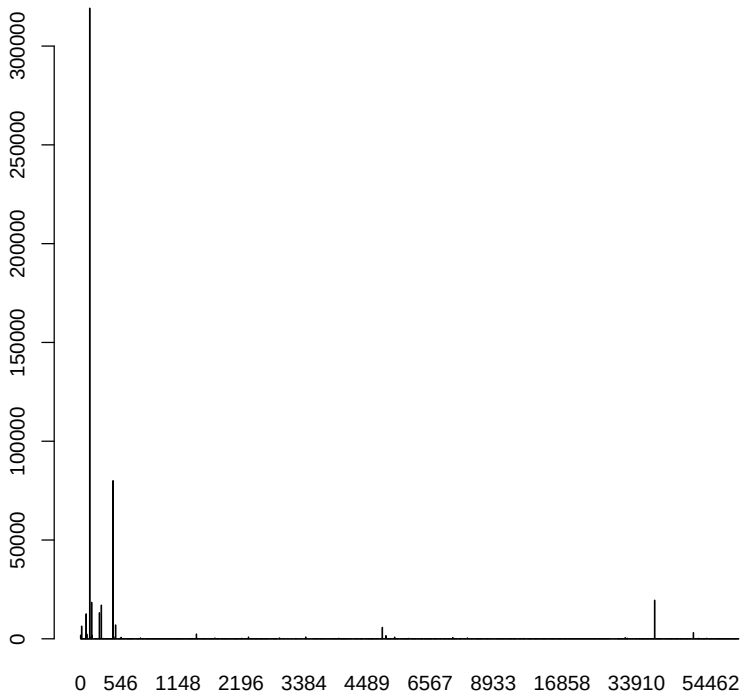

Figure 2.1: *Example of standard network connections across all ports*

ports are being scanned, and these scans are happening with no noise, meaning it is the same list of ports being scanned repeatedly. In Figure 2.3, the scan is obfuscated a little bit, there is more variance in ports being scanned, but some ports are hit almost every time. A threat actor might do this so a basic scan detector does not detect the similarities in the scans. The purpose of a port scan is to see which ports are open and which are closed, a threat actor will use scans as reconnaissance for future attacks. In the scope of this thesis, it is better that all scans are detected, including legitimate ones, so a human responder (threat hunter) can decide if action is required.

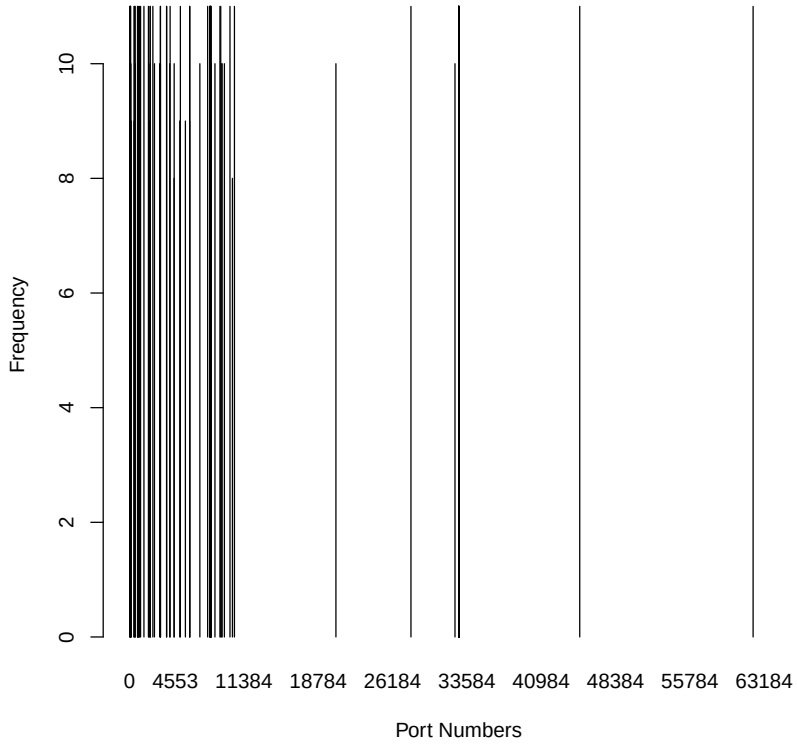

Figure 2.2: *Example of a scan with little to no noise*

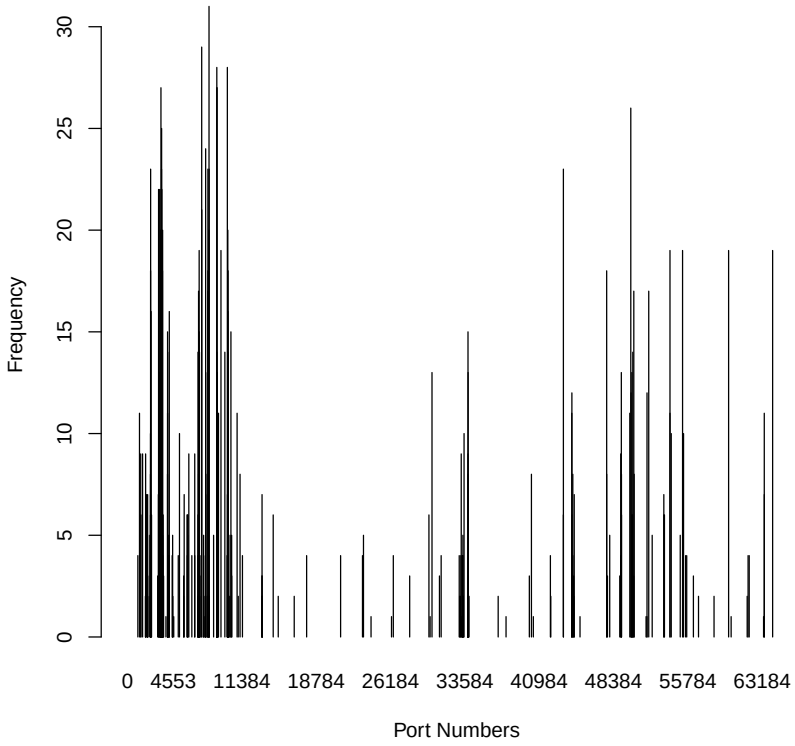

Figure 2.3: *Example of a scan with noise added in*

2.2.2 Beacons

Another attack vector, beacons are multiple connections made from one machine (host) to another machine (destination, inside or outside the network). Normally this is preformed on the host machine by a program that will send a connection every x time units, where x is any integer. Normally the time frame is in minutes, or hours. Rarely will a beacon will send a request over a course of days. Some call this beacon connection a heart beat, because it will make a request on a predictable schedule like a heart beat. A threat actor might add in a random value to the base x time units, in order to throw off detection of the beacon. This is effective because if a network administrator or expert system just checks for a connection with a predictable pulse, some random noise will make it look just normal enough to evade detection. However this random noise is commonly normally distributed, so slight alteration can be made to catch these special cases. Like scans there are times when beacons can be used legitimately, for example an operating system will likely check for updates every couple hours. In the scope of this thesis, it is better that all beacons are detected, including legitimate ones, so a threat hunter/hunt team can decide if action is required.

2.2.3 Denial of Service Attacks

Denial of service attacks include both denial of service (DoS) and distributed denial of service (DDoS). The basic concept of both is the same, a threat actor (or threat actors) will attempt to flood a system with so many requests that the system becomes bogged down with all the requests it will eventually begin stopping legitimate users from using that system for what it was intended for. The difference between DoS and DDoS comes from how the attack vector is enacted, a DoS attack will send all requests from a single machine while a DDoS attack will send the requests from multiple machines. The purpose of this attack is to disrupt the normal operation of a system.

2.2.4 Command and Control

Command and control (C2 or C&C) is an attack vector where a threat actor places a program on a target machine to beacon back to a system that the threat actor owns, then using the connection to the target machine they can send malicious commands or malware to the target machine, thus sending commands to a system the threat actor effectively 'controls'. The purpose of this attack is to infiltrate an organization and launch further attacks from inside the target organization.

2.2.5 Fast Flux

Fast flux is a technique used to obfuscate the source of an attack or malware, which uses multiple IP addresses and a single domain name. As time passes, the domain name switches which IP address it points to. This causes the DNS records to change and makes it harder to attribute the domain name with a malicious site. This also makes it harder to report or warn a web user that they are accessing a malicious site.

2.2.6 Phishing

Phishing is a social engineering attack vector that a threat actor might use to get information about or access to a target network or system. This is normally done by contacting a target user and claiming to be a person of authority needing the target user to do some action that will compromise the target network. For example the threat actor may call the target user pretending to be an employee at the same company, and needs them to try logging into a new system. The target user happily complies, reports any errors they encounter, and then goes on their way never knowing they just provided their login information to a threat actor. Since this technique is a social engineering attack it is defiantly outside the scope of this thesis.

2.3 Importance of this Project

Currently RITA needs to wait a full day to determine if an attack is happening. While it is a big step forward from the current detection rate, normally several months on average. (Ponemon Institute 2017) It was also found that a single data breach costs $3.62 million between all the factors. (Ponemon Institute 2017) A part of that expense comes from the time it takes to discover and fix the breach. With that knowledge if a threat hunter is enabled to detect a breach in hours instead of a day they could save companies huge sums of money. So the question becomes how can the detection time move from a day to an more ideal real time (or near real time) detection.

By combining an expert system (classification tools) already used by threat hunters with the flexibility of machine learning (ML) algorithms it may be possible to further reduce the detection of, and therefore the reaction to, a network breach. It is proposed that leveraging the advantages of machine learning and expert systems a better advising tool for threat hunters could be created (since time is a critical factor, reducing time reduces cost). The ML algorithm can learn what similar attributes the data labeled by the expert system have for every class. While RITA uses a day for statistical evaluation, a ML algorithm could learn how to do it on live streaming data.

Chapter 3

Related Work

There are many threat hunting projects similar to this work, in both the academic and commercial spheres, that should be looked at. Though the terms treat hunting and hunt teaming might not appear in academic circles, work related to detecting traffic anomalies in a network using machine learning algorithms do appear quite frequently, and should be acknowledged for their impact on this project.

3.1 Commercial Products

Real Intelligence Threat Analytics (RITA) and Sqrrl both detect threats or breaches into a network. Some companies (such as Carbon Black and Paladion Security) provide detection products and other related services. These related services include contract hunt teams (where a cybersecurity company can be hired to run hunt teams for a client company) and continual analysis of client systems.

3.1.1 RITA and AI-Hunter

RITA uses an statistical analytical approach to finding potential breaches. RITA uses a command line interface and will execute and report the results from those queries. RITA uses the experience of experts in the cybersecurity field to build its queries. RITA is open source and has received a patent to detect scans and beacons using statistical analysis. The primary customer of this is the open source community and people with some knowledge of running command line tools (which is presumed to be most system administrators). Rita will use Bro logs to capture the network data it analyzes.

A project closely related to RITA is the Actual Intelligence Hunter (AI-Hunter) which builds off RITA's logic core to find threats it will then display on a web page. AI-Hunter has additional analytics for further investigation into scans, beacons and other attack vectors of the network data for a system administrator or hunt team to evaluate and report on. Unlike RITA, AI-Hunter is a closed source project.

Both RITA and AI-Hunter are maintained by Active Countermeasures, and both have success in detecting threat vectors other products miss.

3.1.2 Sqrrl

Sqrrl claims to have a machine learning approach to detecting threats, in their blog they discuss forest, but since it is a closed source project, it is not clear if this is their approach. Also Sqrrl requires some knowledge of its query language to display results. Sqrrl has a graphical front end to aid the usage of their product, and has some graphical controls to show their results, but still requires some expertise in the field to use their product. They report grumblings that machine learning in security tools is none explainable. This means that when a classifier makes a decision it cannot be seen by a user how that classifier came to label some piece of data as either normal or anomalous.

3.1.3 Carbon Black

Carbon Black (formally Bit9) sells many different products and services that fulfill the threat hunting need. The core product seems to be CB Defense, which is advertised as using predictive modeling to form a model of normal network behavior and automatically block and prevent any of the attack vectors. Carbon Black also offers Security as a Service, in the CB Threatsight service. Security as a Service is discussed in more depth in Section 3.1.5. They also develop their product in closed source so what they say is believed to be the true operation of their logic core.

3.1.4 Paladion Security

Paladion Security advertises using over 25 artificial intelligence models to predict the presence of a threat actor in network endpoints. They call their approach Managed Detection and Response, and the AI is behind both the detection of and the reaction to a threat, having their experts training the appropriate reaction to a threat. Paladion Security has their product in closed source, and they do not say which 25 AI models they use.

3.1.5 Security as a Service

Security as a service (SecAAS) is similar to software as a service models, the security is handled by a third party, normally a cybersecurity company. This third party provides a range of options for a client's security, these can include hunt teams, intrusion detection, and various other security options. These services will vary from company to company. BHIS offers hunt team options and as mentioned in Section 3.1.3 Carbon Black offers a SecAAS option called Threatsight.

3.2 Academic Projects

It is also important to look at the academic approaches to detecting network anomalies. Network anomaly detection is a very active field that has been on the minds of experts for decades. Below a selection of research on machine learning approaches for the detection of anomalies is presented.

Goldstein and Uchida (2016) explored the application of unsupervised learning techniques on unlabeled data to see which classifier would work best for a wide range of data (including network, life-sciences, and hand-writing detection). The use of the wide range of data is to get a range of "size, dimensionality, and anomaly percentage" to compare the classifiers. In their findings for the KDD'99 dataset (an artificial network dataset with labels) the best classifier was what they call the kth-NN algorithm,

which is the method that is used in Orange (the alternate name is to distinguish between two k-nearest neighbor algorithms). This finding will be used later. Goldstein and Uchida (2016) have a section about data normalization, this is something that is not covered in our section about data preprocessing (Section 4.2.3), the reason why normalization was not used is covered in Section 8.4.

Reddy, Reddy, and Rajulu (2011) also looked into various machine learning (ML) techniques and the application of those techniques in detecting anomalies. They differ from Goldstein and Uchida (2016) in that Reddy, Reddy, and Rajulu (2011) look at the application of ML specifically in relation to IDSs. Their paper is an overview of many related papers and studies that have been published with a quick overview of the projects.

Dhanabal and Shantharajah (2015) look at the use of the NSL-KDD dataset (an advancement of the KDD'99 dataset) for benchmarking. They specifically look at how well support vector machines, J48, and naive Bayes classifiers work to classify the new dataset. They use the WEKA program, and its classifiers to compare the classification methods. They also provide breakdowns of the KDD'99 and NSL-KDD datasets. They decide that the NSL-KDD dataset is better for testing the effectiveness of a classifier than the KDD'99 dataset. They also have a little bit of a comparison of ML classifiers. The reason the NSL-KDD and KDD'99 datasets were not used for this thesis is covered in Section 8.5.

Mukherjee and Sharma (2012) look at the importance of feature reduction for classification accuracy in an IDS in their paper. They use a naive Bayes classifier to detect anomalies in the NSL-KDD dataset. They focus on how different feature selection algorithms will affect the classifiers ability to classify. Since this thesis uses feature reduction, why Mukherjee and Sharma's feature reduction algorithm was not used is considered in Section 8.6.

Farnaaz and Jabbar (2016) and Zhang, Zulkernine, and Haque (2008) discuss the application of random forests in intrusion detection systems. Zhang et al. (2008) use the KDD'99 dataset and Farnaaz and Jabbar (2016) use the NSL-KDD dataset for testing their results of random forest as a part of an IDS. For Farnaaz and Jabbar (2016) they use feature reduction to clean up their training data, as well as some other techniques that could be of interest in the future of this project. Zhang et al. (2008) designed their RF to work in real time based off of Java and WEKA, as mentioned they use the KDD'99 dataset which has been discovered since then by other researches to have problems, these researches include Tavallaee et al. (2009) and Revathi and Malathi (2013).

Yin, Zhu, Fei, and He (2017) tried to improve the results of a machine learning intrusion detection system by using a recurrent neural network (RNN) and they compare their results to a random forest classifier. The reason why an RNN was not used is explored in Section 8.7.

3.3 What is the Difference

This thesis focuses on machine learning on data captured by Bro, so the data and attributes are different (compare the table in the paper by Dhanabal and Shantharajah and the table in Table 4.2). Also in place of using labels made by an expert for a synthetic dataset we attempt to use an expert system that has had success in industry to label the data. While research has been done on a set of well documented data it was interesting to see how data from a new network capture approach (Bro) would be classified by a machine learner when that data was labeled by an expert system. It was also interesting to see if an expert system that uses a day of capture to detect anomalies would create new and interesting splits in the data for further investigations by data scientists.

Chapter 4

Data

Anytime a machine learning algorithm is used a researcher needs to look at data. There were three big datasets used in this project, the wine Dataset, an internal dataset from Black Hills Information Security, and the CTU-13 dataset from the Czech Technical University. It is also interesting to look at how the data was used for the project and the preprocessing done on the data, including mappings from one data type to another, the ways in which the attributes were removed, and how new attributes were created.

4.1 Wine Data

This is a common machine learning dataset that evaluates different quantities of 178 different wines (or samples) classified into three different classes of wine. Each sample has 13 attributes and a class. According to Lichman (2013) each attribute is continuous. The attributes are called alcohol, malic acid, ash, alcalinity [sic] of ash, magnesium, total phenols, flavanoids [sic], nonflavanoid [sic] phenols, proanthocyanins [sic], color intensity, hue, OD280/OD315 of diluted wines, and proline (Lichman). This is a good dataset to start with due to the limited number of attributes (just thirteen) and is a small set of data (178 samples). So training an ML algorithm on this data will allow quick development, testing, and debugging of base code to get it ready for using the algorithm with the network data. For further breakdown of the wine dataset and attribute types see Table 4.1.

Table 4.1: *The wine dataset attributes and the types of each attribute*

Data Name	Data Type
Alcohol	Double
Malic Acid	Double
Ash	Double
Alkalinity of Ash	Double
Magnesium	Int
Total Phenols	Double
Flavonoids	Double
Nonflavonoid Phenols	Double
Proanthocyanidins	Double
Color Intensity	Double
Hue	Double
OD280/OD315 of Diluted Wines	Double
Proline	Int

4.2 Network Data

The next few subsections cover the network data used for refining the ML algorithm and in Orange. When the algorithm had been developed to a robust state and could handle the wine dataset it was necessary to move to network data for testing the proposed method. Since Bro is being used the following fields are available to use for classification: time stamp, unique connection ID, source IP address, source port, destination IP address, destination port, service, duration, originator payload bytes, responder payload bytes, connection state (from a limited range), local origin, bytes missed in content gaps, connection history, origin packet count, originator IP bytes, responder packet count, responder IP bytes, and set of parent UIDs (if tunneled), an overview of the data can be seen in Table 4.2. However some of these attributes are not useful for ML purposes of classification, these attributes are timestamp, unique connection ID, source IP address, and destination IP address. Further detail on removing these attributes can be found at Section 4.2.3. While the data was originally

Table 4.2: *Bro conn attributes and the types of each attribute, for BHIS internal and CTU-13*

Data Name	Data Type
Time Stamp	time
Unique Connection ID	string
Source IP Address	string
Source Port	int
Destination IP Address	string
Destination Port	int
Transport Protocol	string
Service	string
Duration	double
Originator Payload Bytes	int
Responder Payload Bytes	int
Connection State	string
Local Origin	bool
Bytes Missed	int
Connection History	string
Origin Packet Count	int
Originator IP Bytes	int
Responder Packet Count	int
Responder IP Bytes	int
Set of Parent UIDs	string array

stored in MongoDB it was extracted to a comma separated values (CSV) file for multiple reasons, these reasons are further investigated in Section 4.2.3.

4.2.1 BHIS Internal Dataset

This was a dataset created at Black Hills Information Security (BHIS), it was used for initial testing and training of the ML classifier on network data. This data had no labels prior to the development of this project and required some manual overview of the data. The labels come from RITA analysis.

The primary advantage of this dataset is that it is relatively small. It only has 8,518 connections for a one-to-one match of anomalous data to normal data. The

advantage of using a smaller dataset is that an ML classifier can be quickly trained to ensure that all the attributes can be trained on without causing any issues. That classifier could then be used to test on network data with all applicable attributes.

However, this data was not used for final training and testing. The BHIS internal dataset was not used for final training and testing because of the limited number of attack vectors detected. Only three different threat actors were found on this dataset. These threat actors only ran a few port scans. Meaning a limited attack vector space and a very limited number of threat actors acting in a unique way. This is a huge limit to training a classifier as more variance on the data is preferred. High variance data can lead to a trickier problem space for a classifier to to work on, getting better results (or a more realistic view on what a network might look like).

4.2.1.1 Data Overview

The data was gathered using Bro network analysis running on a network switch in front of BHIS servers. This data contains both normal and anomalous traffic coming and going into and out of the BHIS network. This data was gathered over the course of a single day. Before analyzing the size of the dataset was 0.167 gigabytes (GB).

4.2.1.2 Using the Data

Once the classifier code has been developed in GoLang, some changes will need to be made as the classifier moves away from the wine dataset and onto network data. This was used for initial testing that would allow quick detection of any errors in reading network data, training on network data, or testing on network data.

4.2.2 CTU-13 Dataset

CTU-13 Dataset was generated by Garcia et al. (2014) for their paper on comparing the different detection methods for botnet traffic. It captures a wide range of malicious traffic including port scans, DDoS, and C2. It is a fantastic resource because Garcia

et al. (2014) analyzed and labeled the data and it uses Bro logs. The advantage of using this pre-labeled data is that it will make future work far easier. For instance the classifier could be trained to detect more advanced attack vectors, such as C2, and DDoS. For now the use of these data labels will not be used. Instead labels assigned to connections by RITA will be used. RITA has two broad categories of labels we used: beacons and scans. If a connection is labeled as a beacon or a scan it is labeled anomalous (marked in the CSV files with a 2). If the connection is not a beacon or a scan it is called normal data (marked in the CSV files with a 1).

4.2.2.1 Data Overview

The CTU-13 dataset has thirteen different scenarios, each scenario was initially captured as packet capture (PCAP) files, the scenarios are also available as Bro logs for all scenarios except scenario one. The scenarios with Bro logs were preferred since Rita uses Bro logs as input and no additional preprocessing was needed to get the data ready to label (analyzed with RITA).

After loading the CTU-13 dataset (but before analyzing with RITA) into MongoDB it was 7.641GB. The scenarios range from 0.007GB for CTU-Malware-Capture-42 to 2.168GB for CTU-Malware-Capture-51. After analyzing the dataset with RITA the total database size was 8.445GB. Post analysis of the scenarios range from 0.051GB for CTU-Malware-Capture-48 to 2.245GB for CTU-Malware-Capture-51. Each scenario is focused on different attack vectors. These vectors and notes for some scenarios can be found in Table 4.4. The scenarios were gathered in a range of time from 0.26 hours for CTU-Malware-Capture-52 to 66.85 hours for dataset CTU-Malware-Capture-44. The number of packets that were captured by each dataset range from 4,481,167 packets for dataset CTU-Malware-Capture-Botnet-46 up to 167,730,395 packets for dataset CTU-Malware-Capture-Botnet-44. The analysis of

Table 4.3: *Overview of CTU Data sets in MongoDB and Rita evaluation*

Dataset	Original Size	Import Size	Analyzed Size	Analyzed Growth Rate
Malware-Capture-42	0.013 GB	0.007 GB	0.008 GB	+14.29%
Malware-Capture-43	0.571 GB	0.640 GB	0.759 GB	+17.19%
Malware-Capture-44	1.352 GB	1.515 GB	1.635 GB	+10.42%
Malware-Capture-45	0.432 GB	0.447 GB	0.508 GB	+13.65%
Malware-Capture-46	0.043 GB	0.045 GB	0.058 GB	+28.89%
Malware-Capture-47	0.215 GB	0.219 GB	0.263 GB	+20.09%
Malware-Capture-48	0.036 GB	0.039 GB	0.051 GB	+30.77%
Malware-Capture-49	0.926 GB	1.014 GB	1.124 GB	+10.85%
Malware-Capture-50	0.685 GB	0.727 GB	0.837 GB	+15.13%
Malware-Capture-51	1.572 GB	2.168 GB	2.245 GB	+3.55%
Malware-Capture-52	0.056 GB	0.068 GB	0.078 GB	+14.71%
Malware-Capture-53	0.115 GB	0.119 GB	0.149 GB	+25.21%
Malware-Capture-54	0.549 GB	0.633 GB	0.716 GB	+13.11%

Malware-Capture-46 found no scans or beacons, but the growth is due to connecting to blacklisted sites.

Table 4.4: *The attack vectors recorded in each CTU Tableset*

ID	IRC	SPAM	CF	PS	DDoS	P2P	US	HTTP	Notes
Malware-Capture-42	X	X	X						
Malware-Capture-43	X	X	X						
Malware-Capture-44	X			X			X		
Malware-Capture-45	X				X		X		UDP and ICMP DDoS.
Malware-Capture-46		X		X				X	Scan web proxies.
Malware-Capture-47				X				X	Proprietary C&C. RDP.
Malware-Capture-48								X	Chinese hosts.
Malware-Capture-49				X					Proprietary C&C. Net-BIOS, STUN.
Malware-Capture-50	X	X	X	X					
Malware-Capture-51	X				X		X		UDP DDoS
Malware-Capture-52	X				X		X		ICMP DDoS
Malware-Capture-53						X			Synchronization
Malware-Capture-54		X		X				X	Captcha. Web mail.

4.2.2.2 Using the Data

The CTU-13 dataset was used to test how well a variety of classifiers would work in classifying versatile real data, and testing a lot of data so a fair comparison could be

made between the datasets (in Orange). In Subsection 4.2.3 a deeper look is taken into how the data is prepared before testing.

The CTU-13 dataset was also used for testing how an classifier in GoLang would classify a dataset that is both large and diverse. The amount of data is an important measure for determining how a classifier will work in a real world environment, while the diversity of the data is important to see how well a classifier can classify various attack vectors from various threat actors.

4.2.3 Data Preprocessing

After the network data had been imported and analyzed by RITA it was stored in a database. For ease of processing, portability, and compatibility with Orange it was decided to transfer the data to a comma separated values (CSV) file. The values were read from MongoDB and written to a CSV file. During this step some string values (e.g. service, protocol) were transformed into doubles so some of the classifiers (e.g. kNN, RFs) could use these numerical attributes to classify. While the connection history string values were transformed into a series of boolean values. For greater details on the preprocessing of the data, see Section 6.1.

The overview of the training sets used, are seen in Table 4.5 and Table 4.6, questions related to why these sets are not representative of the source data are addressed in Chapter 8.

A few traditional techniques for increasing classification accuracy were used to prepare the data for classification to boost results for a classifier. These techniques are one-way analysis of variation (one-way ANOVA) and association rules.

One-way ANOVA will test "if the difference in means is statistically significant" (for example means between the different attributes of the classes) using the f-distribution. (Lindquist 2011) If the difference of means is insignificant between the same attribute of the two classes then that attribute is highly correlated between the

Table 4.5: *Data used for early training, discovered to be missing data*

Dataset	Anomalous Samples	Normal Samples
Malware-Capture-43	1,370	456
Malware-Capture-44	3,553	1,184
Malware-Capture-45	842	281
Malware-Capture-47	422	141
Malware-Capture-48	86	29
Malware-Capture-49	2,219	740
Malware-Capture-50	2,200	733
Malware-Capture-51	541	79
Malware-Capture-52	N/A	N/A
Malware-Capture-53	N/A	N/A
Malware-Capture-54	N/A	N/A

Table 4.6: *Breakdown of final CTU-13 Training Data used*

Dataset	Anomalous Samples	Normal Samples
Malware-Capture-43	1,370	456
Malware-Capture-44	3,553	1,184
Malware-Capture-45	842	281
Malware-Capture-47	422	141
Malware-Capture-48	85	29
Malware-Capture-49	2,219	740
Malware-Capture-50	2,200	733
Malware-Capture-51	541	222
Malware-Capture-52	86	19
Malware-Capture-53	245	82
Malware-Capture-54	1,473	501

Table 4.7: *Breakdown of final CTU-13 Testing Data used*

Dataset	Anomalous Samples	Normal Samples
Malware-Capture-43	29,921	1,048,575
Malware-Capture-44	435,182	1,048,575
Malware-Capture-45	130,872	1,048,575
Malware-Capture-47	4,301	722,137
Malware-Capture-48	144	137,468
Malware-Capture-49	43,937	1,048,575
Malware-Capture-50	22,142	1,048,575
Malware-Capture-51	1,048,575	1,048,575
Malware-Capture-52	251,586	107,567
Malware-Capture-53	2,751	397,845
Malware-Capture-54	29,055	1,048,575

classes. Since the greater the variance is between the attributes of classes are then the better some classifiers (e.g. RF) can classify the data. One-way ANOVA in Python will return both the F and p values from the test. (Lindquist 2011) The F-value will range between [0, ∞) while the p-value will range between [0, 1]. Generally speaking the larger the F-value is the less correlated the data is. Conversely the smaller the p-value is the less correlated the data is. The two values are correlated, so as the F-value increase the p-value decreases. To adequately test the null-hypothesis (there is no significant difference between the attributes of the classes) a significance level is needed for the one-way ANOVA test, Lindquist (2011) says it is typical to choose a value of 0.05. A basic algorithm for attribute reduction can be seen in Figure 4.1, in this algorithm there are d attributes (the set of attributes will be called D), c classes (the set of classes will be called C), and n data points (the set of all data points is called N). Kursa and Rudnicki (2010) found that the use of their Boruta package in the R language can help "the fluctuating nature of a random forest importance measure and the interactions between attributes." So if a random classifier is used (as will be explored in Sections 7.1 and 7.2), it could be helpful to use some attribute

reduction techniques to balance some of the randomness. It is important to note at this point that this thesis uses the one-way ANOVA feature selection, and not the Boruta algorithm: the algorithm seen in Figure 4.1 is not the Boruta algorithm. Each attribute was compared to every other attribute, if the p-value between attributes is greater than 0.2 the value is automatically removed. It should also be noted that there are other feature selection algorithms, which were not investigated as it was outside the scope of this thesis.

Result: Attribute reduction using one-way ANOVA
foreach *attribute* $i \in D$ **do**
  foreach *example in* N **do**
    classes[example.Class].append(example)
  end
  $f, p = one\_way\_ANOVA(classes[1], classes[2], ...classes[c])$
  if $p \geq 0.05$ **then**
    remove *attribute* i;
  end
end

Figure 4.1: *Algorithm for attribute reduction using one-way ANOVA*

The next useful preprocessing technique is association rules, which seeks to add more attributes to the data. This is accomplished by generating the power set of all boolean attributes and then checking to see if the values appear together often in one class and not the other. If one class could use this new association, it is added to the attributes for every element. In Figure 4.2 let B be the set of all boolean attributes, P be the power set of the boolean attributes (where p a single set of attributes), N be the set of data, and A be the set of original attributes and new attributes. There are a few ways to check if a new attribute is helpful in classifying data, three popular measures are support, confidence and lift. These metrics are seen in equations 4.1, 4.2, and 4.3. Support measures the number of appearances of the new set ($\sigma(X \cup Y)$ over the number of elements in the total set (n), confidence is the

number of times the new set appears over the size of the first set, while lift combines the two and divides confidence by support for the later rule. Tan et al. (2006) say that "support is an important measure because a rule that has very low support may occur simply by chance." They contrast confidence which "measures the reliability of the inference made by a rule" and "the higher the confidence, the more likely it is for Y to be present in transactions that contain X." Later in their book Tan et al. (2006) show that "high-confidence rules can sometimes be misleading because the confidence measure ignores the support of the item set appearing in the rule consequent." They then proposed the lift measure because it balances the strength of both support and confidence, so when finding new associations lift is probably the best to use, and was used for our purposes.

Result: Association rules to create new attributes
$P = power\_set(D_b)$
foreach $p \in P$ **do**
  if $length(p) = 1$ **then**
    A.append(p)
  end
  else
    foreach $n \in N$ **do**
      if $p \in n$ **then**
        $classes\_support[n.class]++$
      end
      $classes\_count[n.class]++$
    end
    for $i = 1;\ i \leq C; i++$ **do**
      if $\left(\frac{classes\_support[i]}{classes\_count[i]}\right) \geq 0.9$ **then**
        A.append(p)
      end
    end
  end
end
return A

Figure 4.2: *Algorithm for association rules*

$$Support, s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{n} \tag{4.1}$$

$$Confidence, c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{4.2}$$

$$Lift = \frac{c(X \rightarrow Y)}{s(Y)} \tag{4.3}$$

These two techniques do not cancel each other out, as one will look for similarity between attributes of different classes and the other one looks at good correlation for boolean values inside one class, but not in others. By coupling these techniques it is possible to reduce unnecessary noise (by removing attributes that are highly correlated) and boost the strength of boolean values by using association rules.

Chapter 5

Classifiers

There are many different classifiers that could be used with varying levels of accuracy and areas of specialty. The two broad categories of a classifier (and one method for updating a supervised learner) should be investigated to talk about the different classifiers. For the scope of this thesis there were three classifiers selected to be tested: random forests (RFs), k-nearest neighbors (kNN), and support vector machines (SVMs). There should be an investigation on the basics of each classifier, how they are implemented, and the theory behind each one.

5.1 Types of Classifiers

The two big categories of classifiers are supervised machine learning (ML) and unsupervised ML, the difference between these categories are the presence of class labels. If the ML classifier needs to have the data labeled with classes to train, then that classifier is called a supervised learner, some examples include SVMs, RFs, and artificial neural networks (ANNs). (Coppin 2004) If the ML classifier does not need class labels on the data to learn or classify then that ML classifier is unsupervised, some examples include kNN, and Kohonen map.

The next method is called a semi-supervised learner, what distinguishes this method from the other two is the ability to update a supervised learner. This is accomplished by using a classifier that already exists to label new training data. Then use that newly labeled data to train an all new classifier. Why this approach is advantageous is that it allows the classifier to update without requiring new data to

be labeled by an expert in the field, reducing the time it takes to update the classifier. This approach could be further refined but that is discussed in Chapter 10.

Another term that is used is ensemble machine learning. Tan et al. (2006) say "an ensemble method constructs a set of base classifiers from training data and performs classification by taking a vote on the predictions made by each base classifier." They say that this will balance the error of a single base classifier, and normally achieves a better classification accuracy than the base classifier does.

5.2 Random Forests

RFs are an ensemble machine learning method that is composed of many random decision trees (Tan, Steinbach, and Kumar 2006). RF was selected because it is an ensemble method, and the base method of that ensemble is a random decision tree, which is easy to build, test and understand. A well engineered ensemble method will preform better than that ensemble method's base classifier, provided for each base classifier less than half make the wrong prediction (Tan et al. 2006). This means that if a forest is composed of many diverse decision trees a good ensemble method should be achieved. This is achieved using a bagging method, where a subset of the training data is selected to train each tree and since no tree is trained on the same data several different decision trees are built. One of the big advantages of the decision tree algorithm is that it is "quite robust to the presence of noise". (Tan, Steinbach, and Kumar 2006) Since the network data could have a high level of noise it makes sense to use the RF to natively eliminate the 'noise'. Coppin (2004) does argue that the base classifier (decision trees) are highly susceptible to noise. Both are possible because the base classifier (the tree) is susceptible to noise while the ensemble classifier is not. Another reason the RF was chosen is because a human user can evaluate the decisions used by each tree and develop new ideas to build better detection systems

in the future. Why this is listed as an advantage is that it can potentially help future researchers to investigate new ways to detect anomalous traffic.

```
Result: Build a decision Tree
if stopping_cond(N, D) = true then
    leaf = createNode();
    leaf.label = Classify(N);
    return leaf;
end
else
    root = createNode();
    root.test_cond=find_best_split(N,D);
    let V = {v|v is a possible outcome of root.test_cond}
    foreach v ∈ V do
        N_v = {n|root.test_cond(n) = v and n ∈ N}
        child = TreeGrowth(N_v, D)
        add child as descendant of root and label the edge (root → child as v
    end
end
return root;
```

Figure 5.1: *Tan's algorithm to build a decision tree*

Building an RF is the same as building multiple decision trees with a few modifications. Start with building a single tree, assume you have n to classify and each element has d attributes. At the start, every element is at the root node, and for each attribute s split points are generated. Then using these split points, compare how well each split point splits the elements at that node by using a purity measure. Purity is defined as $1 - impurity$, it is a measure of how much diversity exists after a split. Tan et al. (2006) give three different ways to calculate impurity; entropy, Gini index, and classification error. In Equations 5.1, 5.2, and 5.3 remember the following symbols: c is the number of classes, and $p(i|t)$ denotes the number of elements of class i at node t. The best split point is found by counting how many elements of each class will go each way after the split is taken and then using one of the equations from Equations 5.1, 5.2, and 5.3 is used then you minimize impurity, which is the

same as maximizing purity. This step is wrapped up in the find_best_split function in Figure 5.1. After you decide on the best split you send a subset of the elements to each of the children nodes (based on the split) and repeat the process until a stopping condition is reached (e.g. a specific purity is reached). To transform a decision tree into a decision forest, the algorithm is run until you have t trees. This is the training step, the testing step simply requires each tree to classify an element and after each tree has labeled that data, a vote between the trees is taken (the trees are all kept equal since each has a voting power of one), whichever class has the most vote is how that piece of data is classified. A simple example would be to imagine a forest of eleven trees, and two classes. An element is run through each tree and six trees say that element is 'normal' and five trees say that the element is 'anomalous', that element is labeled normal and the classification continues.

$$Entropy(t) = \sum_{i=0}^{c-1} p(i|t) log_2 p(i|t) \tag{5.1}$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{5.2}$$

$$Classification\ error(t) = 1 - max_i[p(i|t)] \tag{5.3}$$

A final consideration of an RF is that the more diverse the trees are the better the forest will classify. (Tan et al. 2006) To achieve a high diversity in the trees a method called bagging is used. Bagging takes all n elements in a set and randomly selects a subset of those elements, with replacement (a single element might appear multiple times in each subset). Using this new subset of elements, a single tree is trained. It may also be desirable to limit which elements a tree may use, so in addition to using bagging to train a tree, a subset of features is used. This randomness in the tree

building process "may help reduce the bias present in the resulting tree." (Tan et al. 2006)

Notice that due to the need to measure the purity after each split the growing of a decision tree requires a knowledge of the class. Since the decision tree needs the classes to train, it is a supervised classifier, because the base classifier is supervised it follows that a random forest is also a supervised classifier.

5.3 K-Nearest Neighbors

The kNN algorithm takes each instance of the training data as a d-dimensional vector (d being the number of features each piece of data has). (Coppin 2004) Then it finds the distance between a new piece of data and the training data. A popular distance measurement is the Euclidean distance defined as the distance ($d(p, q)$) between two points (p and q) where $d(p, q) = \sqrt{(q_1 - p_1)^2 + ... + (q_n - p_n)^2}$ and each subscript is the feature number up to d features. (Coppin 2004) The new piece of data is classified as the majority class of its k-nearest neighbor (where k is an integer value chosen by the user), and the newly labeled data can be added to the training set. (Coppin 2004) It may help to look at Figure 5.2 to see this algorithm in an algorithm format.

It is important to note kNN is an unsupervised learning algorithm where the number of classes are specified by the user and the training data does not need to be labeled. Coppin (2004) claims that kNN "performs very well with noisy input data." This built-in noise elimination could be helpful classifying noisy network data, and should theoretically help increase the classification accuracy. One of the fundamental assumptions of the kNN algorithm is that elements that are close physically (in the d-dimensional space) are from the same class. (Coppin 2004) Another way of looking at this is that the normal data will be close together using a distance metric and the anomalous data will be close together using that same distance metric. As with

all assumptions that are made this may or may not apply to the particular case of network data. Tan et al. (2006) warn one of the limiting factors of the kNN algorithm is the variability of data. For instance the data from the CTU-13 Dataset can vary from 0-64535 for the port ranges, while the distance between the boolean variables is either a false (zero) or true (one), this may impact classification (for more information see Section 8.4). They also say that "classifying a test example can be quite expensive because [of the] need to compute the proximity values individually between the test and training examples." One final consideration is that kNN will sometimes set into a local problem, "whereas ... rule-based classifiers attempt to find a global model the fits the entire input space." Therefore the kNN data may not fit all of the data that is tested with it well, and that might be a hard problem to detect and prevent.

Result: Classify using k-nearest neighbor
Let k be the number of nearest neighbors and N be the set of training examples
foreach *test example* $z = (x', y')$ **do**
  compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every example $(\mathbf{x}, y) \in N$;
  Select $N_z \subset N$, the set of k closest training examples to z;
  $y' = argmax_z \sum_{(\mathbf{x}_i, y_i) \in N_z} I(v = y_i)$;
end

Figure 5.2: *Tan's algorithm for k-nearest neighbor classifier*

5.4 Support Vector Machine

Tan, Steinbach, and Kumar (2006) inform that SVMs have their roots in statistical learning theory. SVMs are particularly good at "handwritten digit recognition and text categorization". They are also good at handling high-dimensionality data. (Tan, Steinbach, and Kumar 2006) Before explaining how SVMs work, it is first needed to understand what a maximal margin hyperplane is.

Imagine a d-dimensional space, for a dataset of n elements each element is plotted into that space (each element has d attributes), a hyperplane is found to separate the classes graphically. In most cases using the simple d-dimensional space will not have a linear separator, so it will be necessary to map all the data to another space where a linear separator can be found. A simple example of a two-dimensional SVM (from Tan et al.) to separate circles from squares can be seen in Figure 5.3. In this example the hyperplane is the line. It is possible (and likely) that many different hyperplanes work to separate the classes, so Tan et al. (2006) inform that SVMs work best when the distance between the hyperplane and the nearest element of each class is maximized. The method to find the hyperplane for a nonlinear SVM is shown in Equations 5.4, 5.5, and 5.6, provided by Tan, Steinbach, and Kumar (2006). In these equations $\Phi(\mathbf{x})$ is a transformation to a space where the decision boundary will be linear, while the linear decision boundary (needed for SVMs) takes the form $\mathbf{w} \cdot \Phi(\mathbf{x}) + b = 0$. Finally n is representative of the number of elements present in the training set.

$$\min_{\mathbf{w}} \parallel \mathbf{w} \parallel^2 \; subject\ to: \; y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1, i = 1, 2, ..., N \tag{5.4}$$

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{5.5}$$

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign\left(\sum_{i=1}^{n} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b \right) \tag{5.6}$$

5.5 Orange

Orange Canvas is an open source data mining toolbox that implements several different machine learning algorithms that include the three mentioned above, as well

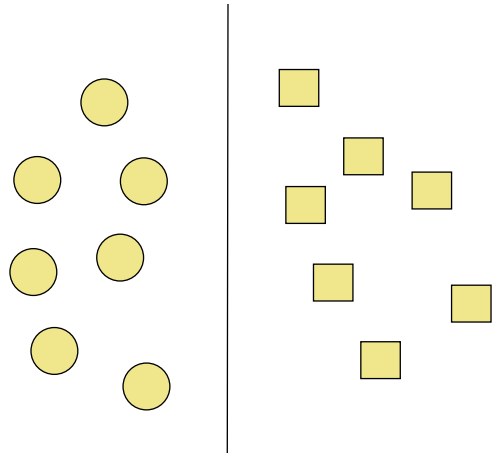

Figure 5.3: *Example of an SVM in two dimensions*

as neural networks, naive Bayes, CN2 and others. It is written in the Python programming language, and is a 'visual programming' interface, so the user requires no knowledge to run the tool set. A screenshot of an Orange work flow can be seen in Figure 5.4. As the figure shows, it is super easy to select the files to open for training and testing, selecting the attributes the classifier can use, and then testing the classifier to get the relevant results from testing the classifier.

5.6 Measuring Classifier Accuracy

To decide which classifier is the best one to use for classifying anomalous network traffic there needs to be some metric or value to assign to the accuracy of each of the classifiers. Tan, Steinbach, and Kumar (2006) indicate that the "evaluation of the performance of a classification model is based on the counts of test records correctly and incorrectly predicted by the model." This measure of correctly labeled records is called accuracy (seen in Equation 5.7), and the measure of incorrectly labeled records is called error (seen in Equation 5.8).

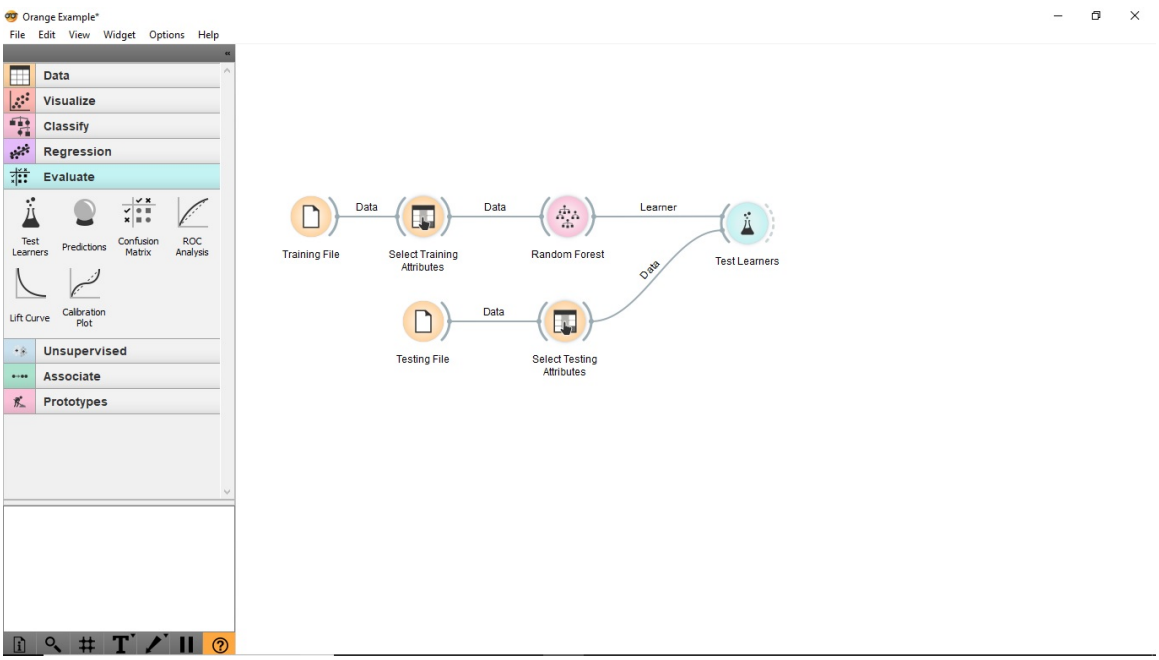

Figure 5.4: *An example of an Orange Workspace that trains and tests a RF*

$$Accuracy = \frac{elements\ correctly\ classified}{total\ element\ count} \tag{5.7}$$

$$Error = \frac{elements\ incorrectly\ classified}{total\ element\ count} \tag{5.8}$$

It is important to note that the number of elements correctly classified is a subset of the total elements, this means that accuracy will be $0 \leq Accuracy \leq 1$. Another important take away is that if an element is not correctly labeled it must be incorrectly labeled, therefore error can also be defined as $Error = 1 - Accuracy$. For this reason including both the accuracy and the error is unnecessary since one can be easily found using the other.

Tan, Steinbach, and Kumar (2006) go on to point out that if low anomaly data is used (such as network traffic), that every element could be classified as normal data and a high classification accuracy (CA) could be achieved. They say this is due to

the class imbalance problem, which states that every class is as important as every other class in problem domains where some classes are far more important than other classes, such as an anomaly class in large piles of normal data. A classifier has got to walk a fine line, and keep its 'self-control. It cannot just classify everything as good or bad, but be judged by its accuracies and inaccuracies made in classifying. Tan et al. (2006) go on to say to get a better view of how well a classifier performs other measurement scores such as sensitivity and specificity, which are dependent on true positive (TP), false positive (FP), true negative (TN) and false negative (FN), should be used. Sensitivity (or True Positive Rate, TPR) is defined in Equation 5.9, it measures how well a classifier will classify a target class. Specificity (or True Negative Rate, TNR) is defined in Equation 5.10, specificity is a measure of a classifier's ability to define the non-target class. The next import thing is to define what positive and negative mean, in the context of this thesis a positive is something that is anomalous, and a negative as something that is normal. Another way to say this is; the target class is anomalous traffic, and the non-target class is normal traffic. Tan, Steinbach, and Kumar (2006) also define false positive rate (FPR), and false negative rate (FNR), but more interesting than either of these are precision and recall. The odd thing is; recall is the same as sensitivity. Precision can be seen in Equation 5.11. The precision of a model is a measure of how few false positive alarms are triggered while recall has the same benefits as specificity (how well the classifier detects anomalous traffic). Tan et al. (2006) do state that either precision or recall is maximized but the other is not, since it is important to look at the recall (fewer false negatives are important for classifying network data) precision was decided to be the less important value. Later work may decide that false positive and false negative should be balanced and use the F1-Measure for that, which is a metric that is "a harmonic mean between recall and precision". Which Tan et al. (2006) say will better represent the lower of the

two values.

Another interesting tool to use in comparing classifiers is the receiver operator characteristic (ROC) curve, which is "a graphical approach for displaying the trade-off between true positive rate and false positive rate of a classifier." (Tan, Steinbach, and Kumar 2006) A good classifier will be as close to the top left of the graph as possible (maximize area under curve and above the line $x = y$). (Tan, Steinbach, and Kumar 2006)

$$TPR = \frac{TP}{TP + FN} \tag{5.9}$$

$$TNR = \frac{TN}{TN + FP} \tag{5.10}$$

$$Precision,\ p = \frac{TP}{TP + FP} \tag{5.11}$$

Chapter 6

Proposed Methods

To test the hypothesis the following methods were used, first take the Bro Logs and import them to RITA (this will store the data in MongoDB). Second analyze the data in MongoDB with RITA (this will store anomalies in new collections), followed by using these anomaly pairs to remove them from the normal data (exporting the RITA results into CSV files). While removing data from MongoDB string values were transformed into integer or boolean values as discussed in Section 6.1. After the data was exported some attributes were removed as they would not help a forest, and may hinder the classification of data. Also at this step labels were added to the data. After the data had labels, representative data was selected (discussed in detail in Section 6.1), to make a training set. This training set can be used to train supervised machine learning algorithms in GoLang.

6.1 Network Data Processing Details

Once the data was exported from RITA (in a CSV file), the time stamp field was removed, since the start time of the connection was inconsequential to if the data was anomalous or not. It is tempting to say if it was after a specified clock time then maybe a connection is anomalous: however the time is not stored as a clock time, instead it is stored as the system time and could impact portability of the classifier. The source IP address and the destination IP address were also removed, this was primarily because training a classifier on either a source or destination IP address would only serve to create a highly specialized blacklist, which is easily bypassed by a threat actor by selecting another machine to attack or attacking from another computer (e.g. using

fast flux from Section 2.2.5). For the purpose of this project in the real world, it would mean that a long period of time is spent training a classifier only to have it quickly become outdated. Also since this is a highly specialized blacklist the classifier would likely be less portable, which means it would likely only work for the training dataset, on a specific machine. For these reasons it was necessary to remove these attributes before building and training a classifier. Of the thirteen datasets two were discarded, the first was Malware-Capture-42 because it did not have the Bro logs needed for RITA to classify (it is possible to transfer PCAPs to Bro logs but the additional processing was not done), also discarded was Malware-Capture-46 because RITA did not detect any scans or beacons, and labeling data is not the focus of this thesis. Notice however that there is a growth size in Table 4.3 for Malware-Capture-42, this is because it was transfered but there were no scans or beacons found (believed to be because there were no Bro logs, but this was not verified). Approaches to solving this issue are discussed in Chapter 10.

During the extraction of data from MongoDB to a CSV file, some potentially useful string attributes were transferred to an integer value or a series of boolean values. This was done to get a numerical value that classifiers could use. The integer values included the application layer (or service), the transport layer (or transport protocol) attribute, and the connection state. The connection history string was transformed into a series of boolean attributes (indicating if a flag was present in the history or not). The transformation table for the Application layer (service) can be seen in Table 6.1, the transformation for Transport layer (transport protocol) can be seen in Figure 6.2, and the transformation for connection state can be seen in Table 6.3. Notice that in Table 6.1 some of the integer values are repeated, this is because Bro will log the same combination of layers in different ways, so ssl,http is logged different from http,ssl. Values like this should act similarly so they share a

Table 6.1: *Mapping of Application Layer string values to integer values*

Application Layer	Integer type
ssl	1
dns	2
dhcp	3
sip	4
snmp	5
ssh	6
teredo	7
dnp3_udp	8
smtp	9
krb	10
smtp,ssl	11
ssl,smtp	11
http	12
pop3	13
ftp	14
ftp-data	15
ssl,http	16
http,ssl	16
rdp	17
irc	18
irc,http	19
http,irc	19
socks	20
All others	0

value (since they use the same application layer). Giving these similar layers different values would be something to explore in the future.

Once all the strings had been transformed to a numerical representation the data was run through a Python script that labeled the data and then combined the normal and anomalous data for a single scenario into a single file. Some of the data lost data points after running the labeling script, so the new data sizes are shown in Table 4.7. This should have no, or little impact on training and classifying. It is assumed that the data is being trimmed by the labeling script because the data is smaller after

Table 6.2: *Mapping of Transport Layer string values to integer values*

Transport Layer	Integer type
tcp	1
udp	2
icmp	3
All others	0

Table 6.3: *Mapping of Connection state string values to integer values*

Connection Name	Meaning	Integer Value
S0	Connection Attempt no reply	1
S1	Connection established not terminated	2
SF	Normal establish & termination	3
REJ	Connection attempt rejected	4
S2	Established, originator attempts close, no reply from responder	5
S3	Established, responder attempts close, no reply from originator	6
RSTO	Established, originator aborted	7
RSTR	Established, responder aborted	8
RSTOS0	Originator sent SYN then RST; no responder SYN-ACK	9
RSTRH	Responder sent SYN-ACK then RST; no originator SYN	10
SH	Originator sent SYN then FIN; no responder SYN-ACK	11
SHR	Responder sent SYN-ACK then FIN; no originator SYN	12
All Others	Any other traffic	0

labeling than what is exported by GoLang, because the value 1,048,575 being the most common value that appears, which is odd to see so often on such different data. Upon further investigation this appears to be the limit of what could be loaded into memory, since opening the data in a spreadsheet also limited the data to 1,048,575. Another script would randomly gather a percentage of each scenario's labeled data into a training file. After a couple iterations of testing some scenarios were discovered to have problems being classified, after investigating what was discovered is that some scenarios were not represented in the training set, an overview of this is seen in Table 4.5, it was corrected and an updated training set overview is seen in Table 4.6.

6.2 Experiment Overview

There are two big portions to look at, the first is to determine which classifier works best for the network data and should be built in GoLang. The second is to look at the accuracy of that best classifier after it is written in GoLang and check if that classifier would work to classify data.

6.2.1 Classifiers Experiment

To compare different classifiers we will use a single program suite to ensure a good machine learning (ML) package was not compared to a bad ML package and using the same ML suite normalizes the packages quality and ensures each classifier preforms as well as the others. Orange Canvas will be used because it is easy to use and the author has previous experience with it. Orange offers a wide selection of well engineered and maintained machine learning classifiers that will test and compare the classifiers out of the box. Orange does support k-nearest neighbors (kNN), random forests (RF), and support vector machines (SVM). As mentioned in Chapter 4, Orange does not have support for MongoDB, but it does support CSV files. So our CTU-13 dataset was moved to CSV files, with the changes that were made earlier (covered in Chapter 4).

For this experiment Orange will take a single datafile and send that data set to all three classifiers. The classifiers will use random sampling on the input data (randomly choose a percentage of the data), it will select 70% of the data for training and repeat the training/testing process 10 times. Then the CA, sensitivity, and specificity. Also the ROC may be taken for an easily viewable graph to report on.

Also during this stage it will likely be necessary to evaluate the runtime of any close classifiers. Since Orange does not have a timing suite we will instead look at the Big O runtime growth. This will show how fast an algorithm will grow given a set of data and any necessary variables to that particular algorithm.

Notice that this is not training on the training data and then testing, this will present something similar to a best case scenario. The training data will be very close to the testing data and the values for CA, TPR and TNR will be very high.

6.2.2 GoLang Classifier

Initially one of the testing sets will be used for training the random forest. Then cross testing will be attempted using the rest of the testing sets. However it is unlikely this will have a desirable accuracy, so the training data will be used in later testing.

The late testing stages will use the training data generate a random forest, and test the random forest on all the testing data, evaluate the results of the forest. The forest was removed from the testing folder, changes were made to the forest, and a new test is started (generate new forest, test, evaluate). The changes may include adding association rules, modifying how the forest works (use features with or without replacement), and feature elimination. This approach is depicted in Figure 6.1.

The quality of the classifier will be determined based on how well it classifies (primarily) and it will also be based on how well it correctly detects normal data correctly and anomalous data correctly.

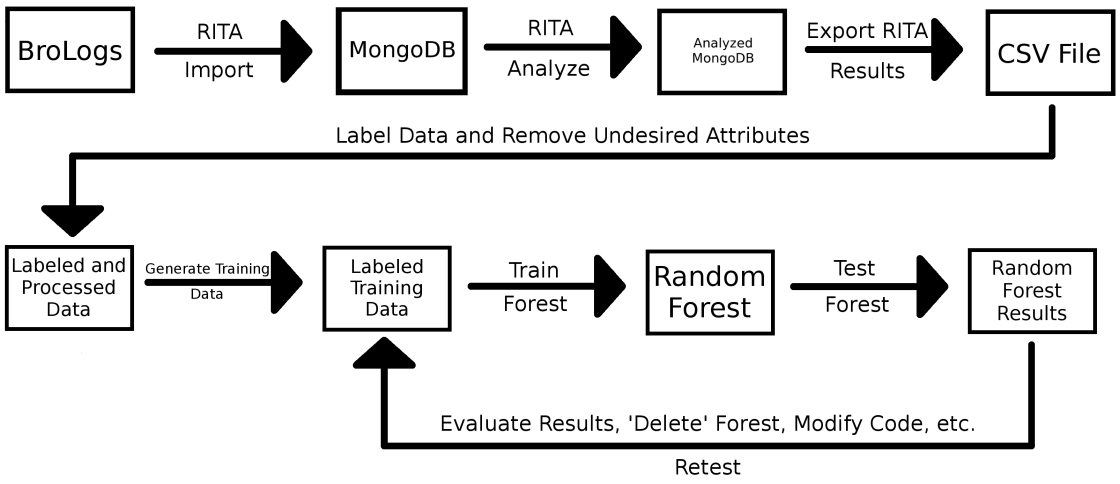

Figure 6.1: *Workflow used for testing the hypothesis*

6.3 Evaluating the Forests

It was desired to look at which attributes a random forest use. Believing that it may be possible to further refine future ML algorithms, or even expert systems if it is known which attributes were favored by the different forests. Since each forest is composed on multiple trees, each of indeterminate height, it could take a data scientist years to wade through all of the forest data. Using a Python script to report on the occurrence of each attribute for a tree, averaged over the forest. Since some attributes can appear multiple times, and the height of each tree is uncertain, it was decided to use a converging infinite series (a geometric series seen in Equation 6.1) to represent the importance of each attribute. This value is called the importance factor for this thesis. The value of two was decided for n in the geometric series, since it converges to a simple integer, two. This means the importance factor would be between zero and two for each tree, then find the running total over all the trees, and finally these values are averaged to give a final importance factor between zero and two. The script will also show the number of times the attribute appeared in the

forest and if the attribute is binary the number of times the forest split to false and the number of times it split to true (remember binary attributes are preprocessed to be zero or one so values greater than or equal to 0.5 are true and everything else if false). This script will also show the max tree height, average tree height and minimum tree height.

$$\frac{1}{1} + \frac{1}{n} + \frac{1}{n^2} + \frac{1}{n^3} + ... = \frac{n}{n-1} \tag{6.1}$$

Chapter 7

Experiments and Results

One issue that was encountered using the CTU-13 dataset in Orange was that the full CSV datasets would cause Orange Canvas to freeze before crashing, so a subset of each scenario was created (limited to about 60,000 elements each). This reduction was done by a Python script that would take a collection of data and try to get half normal and half anomalous data, if there was less than half anomalous data it would fill the difference with normal data. The data was selected using Python's random function, then output to the reduced scenario file. Once the data was reduced, Orange could run and test the CTU-13 data.

7.1 Accuracy Analysis

Table 7.1 shows our results from running the reduced datasets through Orange. The values were found in the Test Learners module. This module will take any number of classifier modules as input (in this case, RF, SVM and kNN) and a dataset. Then using some specified learn/test technique it will compare the different classifiers. The technique used was random sampling repeating the train/test phase ten times. The training set size was 70% of the input data, where training data was selected from the input data randomly, the data that was not selected became the testing data. Once the training data was selected the classifiers were built (RF and SVM), and then the testing data was classified by each of the classifiers. Then this process is repeated, nine more times. Each subtable shows the classification accuracy, sensitivity (TPR), and specificity (TNR) for each dataset, and each row represents another classifier for that dataset. It was important to run many different tests to see how each classifier

would preform over multiple attack vectors, and see if the accuracy would hold. It is helpful to see all of the different results to quickly and efficiently see how each classifier compares to the others for multiple datasets.

What is seen from the Table 7.1 results is that kNN is the most accurate. The most sensitive (in most cases) is the RF. It is important to notice the rare case in Subtable 7.1e where the RF almost completely fails to detect anomalous traffic. It is not likely that that is a common problem, and could be caused by many different issues. In that particular case it is interesting to look at the classification accuracy, which is quite high, and a low sensitivity rate. From looking at the dataset itself it contains a low amount of anomalous traffic (144 instances), remember Orange used random sampling. Meaning that it would likely return a high amount of normal data to train on and low amount of anomalous data to train with. So the data for training needs to be carefully selected to get a fair representation of both classes when a classifier is built in GoLang (but that is discussed in Chapter 8). In most cases it appears that the kNN and the RF are as good as the other. The SVM performs worst of all in almost all conditions. So it would be better to use either the kNN or RF for better accuracy in the final product. The ROC curve of the classifiers can be seen in Figure 7.1 and Figure 7.2. Figure 7.1 shows the full graph of all three, the bottom curve is for SVM, the middle line is RF, and the top (blue) line is kNN. Since it is a little hard to see the difference between the kNN line and the RF line, Figure 7.2 zooms into the top left corner of the ROC curve to better show the space between the two classifiers. Notice that all three lie above the $x = y$ line so all three classifiers should be considered good classifiers, the more interesting is, which classifier is best.

Table 7.1: *Results for CTU-Datasets in Orange*

(a) *Malware-Capture-43 Results*

Classifier	CA	TPR	TNR
KNN	0.9874	0.9915	0.9834
RF	0.9732	0.9978	0.9486
SVM	0.9208	0.9636	0.8780

(b) *Malware-Capture-44 Results*

Classifier	CA	TPR	Spec
KNN	0.9929	0.9935	0.9922
RF	0.9787	0.9989	0.9584
SVM	0.9492	0.9573	0.9411

(c) *Malware-Capture-45 Results*

Classifier	CA	TPR	TNR
KNN	0.9957	0.9966	0.9948
RF	0.9951	0.9924	0.9979
SVM	0.9712	0.9604	0.9819

(d) *Malware-Capture-47 Results*

Classifier	CA	TPR	TNR
KNN	0.9987	0.9934	0.9991
RF	0.9967	0.9638	0.9992
SVM	0.9632	0.6341	0.9886

(e) *Malware-Capture-48 Results*

Classifier	CA	TPR	TNR
KNN	0.9996	0.9326	0.9998
RF	0.9982	0.2372	1.0000
SVM	0.9976	0.0000	0.9999

(f) *Malware-Capture-49 Results*

Classifier	CA	TPR	TNR
KNN	0.9909	0.9931	0.9887
RF	0.9509	0.9951	0.9066
SVM	0.8734	0.9509	0.7958

(g) *Malware-Capture-50 Results*

Classifier	CA	TPR	TNR
KNN	0.9956	0.9963	0.9951
RF	0.9719	0.9970	0.9572
SVM	0.9051	0.9278	0.8919

(h) *Malware-Capture-51 Results*

Classifier	CA	TPR	TNR
KNN	0.9998	0.9998	0.9998
RF	0.9998	0.9997	0.9999
SVM	0.9928	0.9994	0.9863

(i) *Malware-Capture-52 Results*

Classifier	CA	TPR	TNR
KNN	0.9998	0.9997	0.9998
RF	0.9982	0.9999	0.9964
SVM	0.9916	1.0000	0.9832

(j) *Malware-Capture-53 Results*

Classifier	CA	TPR	TNR
KNN	0.9995	0.9959	0.9996
RF	0.9984	0.9968	1.0000
SVM	0.9879	0.8855	0.9928

(k) *Malware-Capture-54 Results*

Classifier	CA	TPR	TNR
KNN	0.9975	0.9980	0.9971
RF	0.9794	0.9980	0.9620
SVM	0.9216	0.9613	0.8843

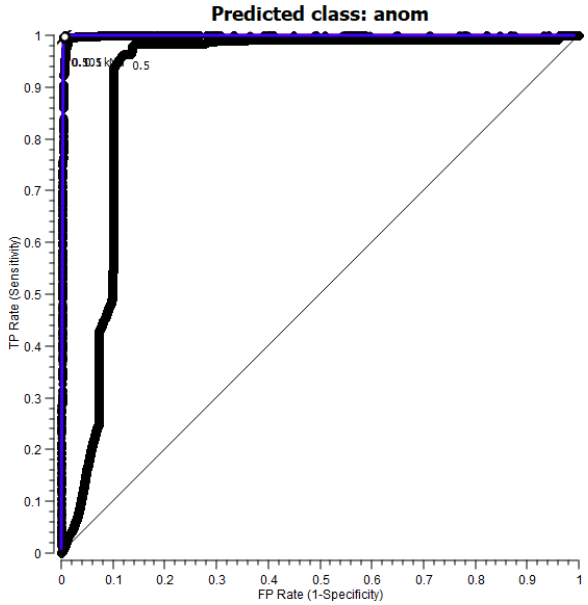

Figure 7.1: *A snapshot of the ROC for all three classifiers in Orange on dataset Malware-Capture-43 Results*

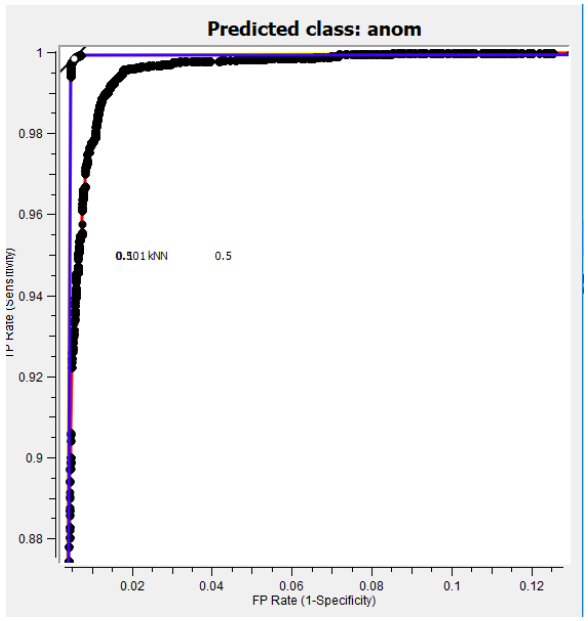

Figure 7.2: *Close view of kNN and RF results on dataset Malware-Capture-43 Results*

7.2 Runtime Analysis

The next factor to consider is the time it takes each classifier to classify the data. If the best classifier takes too long to detect a breach what good does it do? If a classifier is a little less accurate but significantly faster at classifying data then it will be preferred to a slower, but more accurate classifier. It would be great if time would stand still and a system administrator could look around and see more of what surrounds them, but it was stressed in Chapter 2 that time is critical. While Orange is a great tool for classifying and comparing the different classifiers it lacks the ability to give a user runtime (normally time is not a factor). When running the tests it took about a day and a half to run all the data for the different scenarios for both SVMs and RFs (each test taking a matter of minutes), and about five days to run all data through for the KNN, each test taking about almost four hours to run. In place of researching the runtime it would be helpful to look at the Big O growth rate of the different classifiers, to see how each algorithm will scale for multiple factors including, number of attributes, number of elements, and any other that might be relevant to a particular classifier. Since the SVM did not perform as well as the other two classifiers in Section 7.1, only the runtime of kNN and RF will be evaluated.

7.2.1 kNN Analysis

To analyze the kNN algorithm run time, consider each of the parts of the algorithm like Veksler (2015) does in her lecture slides. The first part of the algorithm requires the finding of the distance to a given point, this is $\mathcal{O}(d)$ or linear time (the time to find the distance to a point is dependent on the dimensionality of the data). The next step is to find the distance to that point from every other data point in our set, if there are n data points that will take, $\mathcal{O}(n * d)$. The next step is to find the k nearest points to every point. To look through a list is $\mathcal{O}(n)$ and it needs to look at

$$\mathcal{O}(n * d + n * k)$$

Figure 7.3: *Veksler's k-nearest neighbors runtime*

$$\mathcal{O}(n * d * k)$$

Figure 7.4: *Osadchy's k-nearest neighbors runtime*

the 'found' distance to every other point, and repeat that k times. This adds a time complexity of $\mathcal{O}(n * k)$. The total time complexity for kNN is seen in Figure 7.3. Furthermore one of the key components of the kNN algorithm is deciding the best value for k. (Veksler 2015) However Osadchy (2013) argues that the runtime of kNN is a little different, she agrees that finding the distance between one point and every other point is $\mathcal{O}(n*d)$ but argues that the overall time for finding the k nearest points requires multiplying by k, this is reflected in Figure 7.4. Veksler (2015) says that the test stage is computationally expensive, and there is no training stage. She also goes on to say that for her problems it is better to have a slow training time and a fast testing time (her problem domain is in computer vision where fast reaction is also important).

7.2.2 RF Analysis

Remember that random forests are an ensemble machine learning algorithm. Therefore to evaluate the runtime of a random forest, the runtime of a decision tree needs to be evaluated. For training, a single tree will require a comparison of all the attributes for all the data points at a node level (boolean values can be removed once used). This means it has to run over d attributes for n data points and is repeated over the height of the tree, which on average is $\log(n)$ node levels tall. Combined these parts

$$\mathcal{O}(d * n * log(n))$$

Figure 7.5: *Unpruned Tree Training Runtime*

$$\mathcal{O}(t * d * n * log(n))$$

Figure 7.6: *Random Forest Training Runtime*

give the equation seen in Figure 7.5. The only addition that a forest adds to that equation is that the decision tree part is repeated t times, once for each tree in the forest. So when added in, the equation for a decision forest is seen in Figure 7.6. Tan et al. (2006) say that "once a decision tree has been built, classifying a test record is extremely fast, with a worst-case complexity $\mathcal{O}(w)$, where w is the maximum depth of the tree." Remember that a forest is composed of t trees so it can be found that the worst run time of a forest to classify n element (after being built) is seen in Figure 7.7, in this case w is the maximum height of all the trees.

7.2.3 Evaluating Runtimes

One of the advantages a random forest has, is its training stage (once the forest is grown it can be used more quickly than it took to grow, kinda like real forests). (Veksler 2015) Therefore a forest can be grown, and saved, to be used later at a much quicker rate. Also the forest can be later be used to make an updated forest as discussed in Section 5.1. What is seen from the Big O runtime is that both algorithms

$$\mathcal{O}(n * t * w)$$

Figure 7.7: *Random Forest Testing Runtime*

should have a similar growth rate. However running in Orange it seems like kNN runs at a much slower rate, one possible reason for this is that the RF runtime is closer to a worst case. This is because Big O is generalizing something that is 'random' and Big O evaluation tries to find a curve which when multiplied by some constant that equation will be greater than *every* runtime of that algorithm, while the kNN algorithm *must* run as described in Section 5.3. Put simply, the RF can preform better than the kNN in time complexity, but at its worst the RF will always preform *as well* as the kNN. From the experience of testing in Orange, and what Veksler (2015) warned about the test stage for kNN, the random forest was built in GoLang.

7.3 Random Forest in GoLang

Once it was decided to use random forests for classifying network data, it was important to see how it worked in cross data testing. The wine dataset from Section 4.1 was used to build a basic RF framework. This was necessary since GoLang does not yet have a general machine learning library implemented at this point. After the basic RF was built, the BHIS internal dataset from Section 4.2.1 was used. However due to the limited dataset and attack vectors in this dataset we found a forest of any size, while highly accurate, was composed of very similar trees. In fact all the trees were identical. What this accomplished was a very specific forest to classify a very specific dataset (what was warned about in Section 4.2.3). However the internal BHIS was useful for further developing a forest framework to debug problems that appeared by transferring from a simple dataset (wine) to a more complicated network dataset.

The bulk of the work in creating and refining the RF was accomplished using the CTU-13 dataset (seen in detail in Section 4.2.2). When the code was ready to run, promising results were generated when an RF was trained using data from Malware-Capture-45, seen in Table 7.2. This was a really exciting result to see, however

when the forest was cross-tested on the data from Malware-Capture-51 as seen in Figure 7.3. This result was repeated multiple times with many different datasets, each with similar cross- testing results. When trained on one set of data the forest was rarely portable to other datasets (rarely is used because some datasets with similar attacks could detect each other). The next step was to take a small random sampling from each datasets and use this random sample approach to generate the training dataset (the dataset seen in Tables 4.5). Between this step and the next a couple issues were discovered, some with the forest and some with the training data. First the forest was modified to reuse non-boolean attributes if a tree decides that attribute should be reused. Once these changes were made the previous tests were repeated without much improvement. At this time some of the tests started to throw memory error problems, and it was discovered that the training data had too many attributes (some of the scenarios had been missed during data creation) and some of the attack vectors had no representative examples.

So the data seen in Table 4.6 was created, and training on this data the results from Table 7.4 were achieved. While the results were getting better and there is a fantastic sensitivity more work was desired to get a lower false alarm rate (remember most network traffic should be normal and if almost 40% of normal packets are triggering a 'red alert' a network administrator would probably start ignoring this 'red alert'). In an attempt to increase the specificity of the forest one-way ANOVA was implemented before training each tree, with this new training technique the results for the forest can be seen in Figure 7.5. For every bagged dataset one-way ANOVA found a high correlation between Local Origin, Local Response, Dst Syn Set, and Src Inconsistent Packet, this is because these flags are rarely set so they all have a zero value. Occasionally one-way ANOVA also eliminated Response Bytes, Missed Bytes, Dst Inconsistent Packet, Origin Packets, Response Packets and Response IP

Table 7.2: *Results of forest trained and tested on Malware-Capture-45*

CA	TPR	TNR
96.5768%	95.0485%	98.1050%

Table 7.3: *Malware-Capture-45 trained RF and Malware-Capture-51 tested*

CA	TPR	TNR
49.6468%	0.0000%	99.2934%

Bytes. These values were more rare to see eliminated but depending on the bagged samples could be. With an increase of 12% in classification accuracy by adding one-way ANOVA feature reduction it was decided to attempt a further increase in the accuracy by adding association rules to the preprocessing stage for the forest and the results from Figure 7.6 were achieved. The results from this final test will be discussed in Chapter 9.

One of the most important takeaways from the data seen in Tables 7.2 and 7.3 is that it is vitally important to ensure that cross testing takes place. Even the similarities between scans and beacons (and inside each attack vector), a large sampling from many different attacks is needed to see how well a classifier will work on a live network. Without cross-testing it would be easy to say a high-level of classification as well as a high-level of sensitivity has been achieved, when in reality many attacks are getting through. From this cross-testing need, it can also be inferred that updating the classifier would be needed to catch evolving attack vectors.

The results between Tables 7.3 and 7.4, show that by combining even a small num-

Table 7.4: *Improved tree training and data sampling, all data tested*

CA	TPR	TNR
68.2763%	99.9525%	60.0404%

Table 7.5: *Training with One-Way ANOVA preprocessing*

CA	TPR	TNR
81.4167%	92.2904%	78.5895%

Table 7.6: *Training with ANOVA & Association Rules*

CA	TPR	TNR
81.2054%	92.6871%	78.2202%

ber of samples from each of the network captures a much higher level of classification accuracy, sensitivity and specificity is achieved. This is a much better result and the combination data is better for building a classifier that is portable, between datasets and in the future better for porting the forest between systems (since a wider range of training attack should transfer to a higher range of detecting attacks in the testing phase). There may be some questions resulting from the implications which will be looked at in Chapter 8.

Tables 7.4 and 7.5 depict the biggest jump in accuracy, specificity and sensitivity, and underline how important removing unnecessary attributes is for this particular dataset. An increase of 13% is the biggest improvement in accuracy that is seen, while some might look at the low impact that association rules have in Table 7.6 and decide that they are not important. In fact it appears that association rules hurt the classifier, however due to the marginal change in the data this is not a significant change to the classifier, and while it might add a little bit of time. They do help the sensitivity and it is important to catch more of the anomalous traffic than normal traffic this is probably more desirable than leaving the association rules out. An argument could be made that this small of a change can be caused by slight variation in the random training samples selected, or a series of slightly different trees being generated.

7.4 Forest Analytics

As mention in Section 6.3, it was possible to generate a rough overview of a forest using a python script. What is seen by comparing Table 7.7 and Table 7.8 is when one-way ANOVA is used the preferred split attribute is shifted from Protocol to Src Syn Set. Another interesting thing to note, the addition of the association rule in Table 7.9, which combines the Src Data and Dst Data attributes, makes a fairly significant contribution. This should not surprise anyone, since association rules are designed to find a rule that would help one class be classified better. While it is not certain which class this association rule helps it could actually be a factor in the lower accuracy and specificity, while still increasing the sensitivity of the forest. This is supposed because there are fewer instances of anomalous traffic overall, and the new attribute seems to be contributing to high levels of the tree on average. One final thing to point out is that in all three tables, the Local Origin, Local Response, Dst Syn Set, Src Inconsistent Packet, and Dst Inconsistent Packet attributes are not used. An eagle eyed observer might ask why more attributes are not taken out between Table 7.7 and Table 7.8, this is simply because the values that have a high level of covariance were almost always the ones that one-way ANOVA ignored anyway. However by relying on removing those variables instead of random variable removal, each tree gained a better classification accuracy which boosted the accuracy of the entire forest.

Table 7.7: *Non-boolean replace forest attribute importance*

Attribute	Importance Factor	Appears	True Appear	False Appear
Origin Port	0.00449	39975	N/A	N/A
Response Port	0.00339	64572	N/A	N/A
Protocol	0.78156	8949	N/A	N/A
Service	0.13369	12148	N/A	N/A
Duration	0.04777	306085	N/A	N/A
Origin Bytes	0.00061	26220	N/A	N/A
Response Bytes	0.00134	17140	N/A	N/A
Connection State	0.00544	785	N/A	N/A
Local Origin	0.00000	0	0	0
Local Response	0.00000	0	0	0
Missed Bytes	0.00008	422	N/A	N/A
Src Syn Set	0.04197	2048	1627	421
Dst Syn Set	0.00000	0	0	0
Src Syn-Ack Set	0.00240	357	0	357
Dst Syn-Ack Set	0.00803	392	309	83
Src Ack Set	0.06084	1936	177	1759
Dst Ack Set	0.01721	2272	2229	43
Src Data Set	0.57355	10340	10293	47
Dst Data Set	0.30595	10627	10602	25
Src Fin Set	0.01462	2379	2246	133
Dst Fin Set	0.01193	2338	2330	8
Src Rst Set	0.00680	2796	527	2269
Dst Rst Set	0.00109	1986	5	1981
Src Bad Checksum	0.00019	7	0	7
Dst Bad Checksum	0.00018	7	0	7
Src Inconsistent Packet	0.00000	0	0	0
Dst Inconsistent Packet	0.00000	0	0	0
Origin Packets	0.01436	19810	N/A	N/A
Origin IP Bytes	0.00000	1	N/A	N/A
Response Packets	0.12181	37350	N/A	N/A
Response IP Bytes	0.00020	31	N/A	N/A

Table 7.8: *One-way ANOVA forest attribute importance*

Attribute	Importance Factor	Appears	True Appear	False Appear
Origin Port	0.04340	190499	N/A	N/A
Response Port	0.00932	6047	N/A	N/A
Protocol	0.37248	17342	N/A	N/A
Service	0.11343	24793	N/A	N/A
Duration	0.13290	595844	N/A	N/A
Origin Bytes	0.02664	107445	N/A	N/A
Response Bytes	0.00102	59932	N/A	N/A
Connection State	0.00947	4556	N/A	N/A
Local Origin	0.00000	0	0	0
Local Response	0.00000	0	0	0
Missed Bytes	0.00060	1014	N/A	N/A
Src Syn Set	0.78235	10181	2454	7727
Dst Syn Set	0.00000	0	0	0
Src Syn-Ack Set	0.00046	149	3	146
Dst Syn-Ack Set	0.09089	8074	1663	6411
Src Ack Set	0.05768	6108	3202	2906
Dst Ack Set	0.01144	2369	2290	79
Src Data Set	0.53191	11477	11472	5
Dst Data Set	0.25856	10497	10413	84
Src Fin Set	0.01585	6806	4775	2031
Dst Fin Set	0.01824	6279	4877	1402
Src Rst Set	0.01213	8037	4845	3192
Dst Rst Set	0.00559	4584	2185	2399
Src Bad Checksum	0.02021	5076	2	5074
Dst Bad Checksum	0.01589	4815	1	4814
Src Inconsistent Packet	0.00000	0	0	0
Dst Inconsistent Packet	0.00000	0	0	0
Origin Packets	0.26672	40344	N/A	N/A
Origin IP Bytes	0.05427	7152	N/A	N/A
Response Packets	0.02519	7507	N/A	N/A
Response IP Bytes	0.00032	162	N/A	N/A

Table 7.9: *ANOVA & association rules forest attribute importance*

Attribute	Importance Factor	Appears	True Appear	False Appear
Origin Port	0.04555	192838	N/A	N/A
Response Port	0.00847	6442	N/A	N/A
Protocol	0.34834	18927	N/A	N/A
Service	0.10414	26132	N/A	N/A
Duration	0.13420	600553	N/A	N/A
Origin Bytes	0.02730	108780	N/A	N/A
Response Bytes	0.00110	60268	N/A	N/A
Connection State	0.00764	4981	N/A	N/A
Local Origin	0.00000	0	0	0
Local Response	0.00000	0	0	0
Missed Bytes	0.00066	1302	N/A	N/A
Src Syn Set	0.76857	11082	3536	7546
Dst Syn Set	0.00000	0	0	0
Src Syn-Ack Set	0.00048	150	2	148
Dst Syn-Ack Set	0.08889	8297	1907	6390
Src Ack Set	0.04953	6007	3235	2772
Dst Ack Set	0.01158	2870	2810	60
Src Data Set	0.51401	10171	10171	0
Dst Data Set	0.24553	9195	9174	21
Src Fin Set	0.01776	8238	5943	2295
Dst Fin Set	0.01720	6988	5633	1355
Src Rst Set	0.01222	9647	6065	3582
Dst Rst Set	0.00450	5155	2381	2774
Src Bad Checksum	0.02045	5135	1	5134
Dst Bad Checksum	0.01574	4819	0	4819
Src Inconsistent Packet	0.00000	0	0	0
Dst Inconsistent Packet	0.00000	0	0	0
Origin Packets	0.25772	42342	N/A	N/A
Origin IP Bytes	0.05260	7268	N/A	N/A
Response Packets	0.02425	8364	N/A	N/A
Response IP Bytes	0.00084	202	N/A	N/A
Src & Dst Data Set[1]	0.11830	2974	N/A	N/A

[1] A new attribute created by the association rules

Chapter 8

Discussion

Some tactics we used that might be argued include, the training data not representing the source data, the early tests of forests trained on data from Malware-Capture-45 and Malware-Capture-51 (not all the data that is seen in later tests), and that too many changes took place in between some tests, that the data should have been normalized, use a more standard test dataset like the KDD'99 dataset, use different feature reduction algorithms, use a recurrent neural net (RNN) or other classifier, or that what this accomplishes is a more advanced IDS.

8.1 Training Data Misrepresentation

It might be said that the training data over-represents anomalous data. While it is true that there is a lot of anomalous data it was found that the bagging method might not select enough anomalous data to train a forest that adequately catches all of the attack vectors. It was found that over-representing the target class data meant that the classification of that class would be boosted. It is believed that because the specificity is still pretty high (around 80%), and overall classification is also high (also around 80%) this is an acceptable method to train the forest on. This can be justified due to the shear amount of normal data in the testing step. If this truly had an deep impact it was in reducing the number of normal data that was correctly classified. Because the specificity remains high, it can be assumed that data is being classified quite well, and fairly.

8.2 Limited testing

An argument might arise because the first couple tests we trained on one dataset and tested on itself or *one* other dataset, those tests are not representative of all the data. These two datasets were actually selected because it was believed that they used similar attacks (seen in Table 4.4) and these similar attacks would act similarly. In finer breakdown (in later tests) it can be seen that these datasets have similar sensitivity, and classification accuracy. Further, as the forest code was developed it became necessary to not only see how the classifier worked on datasets with similar attacks but also seeing how it would classify data that was very different. So if a classifier did a bad job of classifying similar data, it probably would not work well on different datasets. It was also thought that for quick testing a classifier, and refining the techniques, similar datasets should be used.

8.3 Too many changes between test

In the early stages, when it became necessary to to switch from using training on one data set and one testing set to using a composite training data set and testing on all data, there were other changes (such as tree changes and data changes). How can it be assured that a better classifier was built and the data changes did not affect it? There are too many independent variables. While the classifier did have better performance it was not significant enough to report on. Changing the data may have had some impact (better representatives of the data to classify), this is a hard argument to make because every time the forest is grown it uses bagging. Since a random forest is fundamentally 'random' it is always possible that some data is selected in the bagging stage will represent the data better, there is no way to account for that randomness which is why testing on many datasets was preferred in later classification stages, to get a fair view of how everything was done.

8.4 Data should be normalized

In Section 3.2 it was mentioned that some studies have argued for the normalization of data. However the CTU-13 data was not normalized for this thesis, why not? The reason is that this thesis supposes the data will be developed to take real-time data. While the normalization values could be saved, to get a better scenario of real world classification it was decided to keep preprocessing to a minimum (generally if it could not be preformed by the classifier in the training step, it should not be done), for example if certain attributes are eliminated in the training the classifier does not need to remove the attributes in the testing stage.

8.5 Use a standard dataset, such as NSL-KDD

In Section 3.2 it could seem the NSL-KDD is standard, why use the CTU-13 dataset and not the NSL-KDD dataset which seems to be more common? This can be answered in similar fashion to Section 8.4, the data might be live in the future. To integrate the data with RITA requires the data to align (features, collection, etcetera). While the data in the NSL-KDD has fantastic labels it does not have the same features as RITA uses. The CTU-13 dataset does have the same features as RITA, and Bro Logs have and use, as discussed in Section 4.2.2.1.

8.6 Use a different feature reduction algorithm

As mentioned in Section 3.2 there are many different feature reduction techniques that could be used, but one-way ANOVA was used. Also Section 3.2 discusses a paper dedicated to finding a good classifier, why not use one of their feature reduction algorithms. The reason one-way ANOVA was selected was because it was a simple algorithm. Also while it was desired to find a good method to classify anomalous data it was not necessary to find the *best* feature reduction algorithm, the purpose of

this thesis was to find if a classifier could be used in a real time environment to catch anomalies more quickly than requiring a large capture of network data an expert system might use.

8.7 Look at more classifiers

As mentioned in Section 3.2 why not compare more classifiers in Section 7.1? This is because of the limited classifiers in Orange. Orange does not implement deep learning ML approaches but it does implement the base classifiers (and some popular ensemble methods). So for a quick overview it was decided to use kNN, an RF, and an SVM, due to their different approaches to ML. In the future it may be desirable to add some form of a neural network to give users another view, but again the focus was to see if the Bro data labeled by RITA could be used to accurately be used with a ML classifier to classify data in a real-time environment. The need for the best classifier with the particular data used is a part of a larger question that can be further investigated in the future. The RF can offer insight into why it classified a piece of data as normal or anomalous while that is not the case for a neural network, and being able to give reasons for classification of a particular piece of data.

8.8 This is just a new IDS

Arguably, this is correct. This thesis is focused on being able to detect advanced intrusions, and find potential pre-attack vectors before they become a problem (though some of the attacks our classifier finds are happening such as the beacon). A typical IDS will find attacks as they are happening. Also an IDS will find attacks but as mentioned in Chapter 2, IDSs simply are not working as they currently exist. Yes, this does detect intrusions to a network, but it can be viewed as the next advancement of an IDS. This 'next step' is actively hunting for threat actors in a network, it does not require a maintained list but seeks to inform a human hunter. Equipping

the hunter with more information enables them to better defend any network they are charged with guarding.

Chapter 9

Conclusion

In a sentence, it is possible to use an ML classifier, trained on data logged with Bro to detect expert system labeled anomalous traffic [RITA] to classify data with a high level of accuracy. This is accomplished through some typical preprocessing techniques (such as one-way ANOVA for attribute reduction, and association rules), and a well built classifier (the code is well designed, and the classifier builds well). It was also found that the training data for the classifier is very important. One of the best ways to accomplish good training data is to use samples from many different attack sets, in order to get many different attack vectors a threat actor might use.

It could be advised that the network administrator should make a decision about using one-way ANOVA with association rules or just one-way ANOVA. This might be a little premature because this paper did not look too much into using association rules, but with some of the planned changes in Chapter 10 this should be done later when more research has been accomplished. We have simply drawn the chart, but the system administrator must make 'the captain's call'.

It is believed that due to the high classification accuracy the classifier currently has, the classifier could be used in a live network for real-time, or near real-time classification. It should be accomplished after a little more work is accomplished, such as moving to streaming the data instead of reading from a CSV file, as discussed in Chapter 10.

Chapter 10

Future Work

In the future, the forest may move away from using Bro log data to streaming live data, this would allow a network administrator to see attacks happening live and take action as a threat actor is attacking their network. This further reduces the time of detection from a matter of days to a matter of hours or even seconds. It might also be necessary for the random forest to be expanded to detect more attack vectors, these might include DoS, DDoS and other more advanced attacks, as opposed to a simple normal and anomalous label. This would also allow a network administrator or hunt team to know exactly what the attack vector is and better know how to counter the attack. One consideration for this is the attack vectors would need to be labeled by an expert, this is a time intensive problem which was overcome by using an expert system (RITA) to classify. Another possible step is to find a way to combine a former forest's labels and the label RITA creates to make a more refined semi-supervised classifier, and ensure that the margin of error doesn't grow too wide, and to keep the false alarms from firing too often, and balance the learner from adapting to false alerts.

There is some challenge associated with moving from Bro logs to streaming data since the data that RITA uses from Bro is dependent on a information gather time, specifically a day. Bro keeps the logs in a plain text format, meaning it might be easy to open and read the data from the current Bro log, and make it possible to evaluate in real time. However it is likely Bro will use a mutual exclusion lock to keep the file from being read as it is being written to. Since this log rotates every hour by default, opening the most recent hourly capture from Bro would allow a highly active hunt

time. Since a time reduction from months to an hour would be an amazing time gain, even dropping from twenty-four hours to one hour can save thousands of dollars over the course of a year, not to mention the life of a product.

Before any of that work is accomplished the RF should be fully integrated into RITA, and a little bit more work should be done to finding the split points for the forest. A small number of split points are being tested so the RF is likely not finding the best split point. As the number of split points increases the computation time to build the RF increases, but so should the accuracy of the classifier.

As mentioned in Chapter 1 new malware is being developed constantly, an RF coupled with a semi-supervised update could equip a threat hunter to keep up with threat actors, with minimal effort on the hunters part. So the semi-supervised method would need to be refined, and researched. A novel way of accomplishing this would be to find a way to combine the labels from an RF with the certainty measure from the RITA output. If this is indeed the preferred method for classifying the data a simple way to do this is to assume each tree to be independent of every other and change the forest classification from using a majority vote option to a Bayesian classifier for the forest. Then using the certainty measure from RITA in the scans and beacon data to determine whether to classify each data point as anomalous or normal before training the replacement forest; this has an advantage of balancing errors that might propagate from too many generations, while still leveraging the knowledge of an expert system with the learning ability of a classifier.

In the future it might also be desired to add more ML classifiers to see how they compare and contrast with each other. For the main portion of this thesis it was more important to offer a hunt team the ability to see why a particular piece of data was classified, than it was for it to classify correctly (while correct classification was certainly a big concern, it was not the only one). The RF can offer its steps to

classification (since it can be evaluated and shown graphically). While some of the other classifiers offer more vague reasons (such as an SVM) which would not make sense. What information would a system administrator gain from a hyperplane? More ML classifiers (especially coupled with a random forest) would offer more views, the user would just have to be alright without seeing how that classification came about.

References

Albayati, M. and B. Issac (2015). Analysis of intelligent classifiers and enhancing the detection accuracy for intrusion detection system. *International Journal of Computational Intelligence Systems 8*(5), 841–853.

Coppin, B. (2004). *Artificial Intelligence Illuminated.* 40 Tall Pine Drive, Sudbury MA: Jones and Bartlet Publishers.

Dhanabal, L. and S. Shantharajah (2015, June). A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer and Communication Enineering 4*.

Farnaaz, N. and M. A. Jabbar (2016, January). Random forest modeling for network intrusion detection system. *Procedia Computer Science 89*, 213–217.

Fielding, E. R. and E. J. Reschke (2014, June). Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, Internet Engineering Task Force.

Garcia, S., M. Grill, H. Stiborek, and A. Zunino (2014, May). An empirical comparison of botnet detection methods. *Computers and Security Journal 45*, 100–123.

Goldstein, M. and S. Uchida (2016, April). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS One 11*.

Kursa, M. B. and W. R. Rudnicki (2010, September). Feature selection with the boruta package. *Journal of Statistical Software 36*.

Lee, R. M. and R. Lee (2016, February). White paper: The who, what, where, when, why and how of effective threat hunting. Technical report, SANS.

Lichman, M. (2013). UCI machine learning repository.

Lindquist, M. A. (2011). One-way anova.

Minihane, N., F. Moreno, E. Peterson, R. Samani, C. Schmugar, D. Sommer, and B. Sun (2017, December). Mcafee labs threat report december 2017. Technical report, McAfee Labs.

Mukherjee, S. and N. Sharma (2012, February). Intrusion detection using naive bayes classifier with feature reduction. *Procedia Technology 4*.

Osadchy, R. (2013, April). Nonparametric density estimation nearest neighbors, knn. .

Ponemon Institute (2017, June). 2017 cost of data breach study: Global overview. Technical report, Ponemon Institute, 2308 U.S. 31 North, Traverse City, MI. This study was performed by Ponemon Institute and sponsored by IBM Security.

Reddy, E. K., V. N. Reddy, and P. G. Rajulu (2011). A study of intrusion detection in data mining. In *Proceedings of The World Congress on Engineering 2011*, pp. 1889–1894. IAEng.

Revathi, S. and A. Malathi (2013, 01). A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. *2*, 1848–1853.

Tan, P.-N., M. Steinbach, and V. Kumar (2006). *Introduction to Data Mining*. Hoboken, New Jersey: Pearson Education Inc.

Tavallaee, M., E. Bagheri, W. Lu, and A. A. Ghorbani (2009, July). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6.

Veksler, O. (2015, September). Lecture slides in machine learning in computer vision. `http://www.csd.uwo.ca/courses/CS9840a/Lecture2_knn.pdf`.

Yin, C., Y. Zhu, J. Fei, and X. He (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access 5*, 21954–21961.

Zhang, J., M. Zulkernine, and A. Haque (2008, Sept). Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 38*(5), 649–659.

Glossary of Terms

Attack Vector - A specific attack or approach to compromising a system

Beacon - An event where a computer or computer system sends a connection request to another computer or computer system at a regular or semi-regular interval. This can be either benign or malicious. In the case of a benign beacon it is connecting for updates or for ads. In a malicious case it is probably an indication of connecting to a C2 server.

C2 - Command and Control - an attack vector where a threat actor uses a compromised system to connect back to a system owned by the threat actor to issue commands on the compromised system. Often indicated by beacons.

Click Fraud - Also called pay-per-click fraud is the practice of generating false clicks for higher traffic statistics than actually exist.

Compromised System - A system that a threat actor has access to. Used in attacks or precursors to attacks.

DDoS - Distributed Denial of Service - an attack vector where a threat actor attempts to overload a system by launching many requests to that system from multiple compromised systems.

DoS - Denial of Service - an attack vector where a threat actor attempts to overload a system by launching many request to that system from a single threat system.

FastFlux - A technique used to mask an IP Address that is serving malicious code or phishing servers.

ICMP - Internet Control Message Protocol - an Internet layer protocol for use as a base in Internet communication.

IOC - Indicators of Compromise - indicators of a breach of a network or system by a threat actor.

IP address - Internet Protocol address - an address for a computer on a network. The address can be either private or public, most computers have both a public and private IP address.

IRC - Internet Relay Chat - an application layer protocol for passing text messages.

Net − BIOS - an API similar to a session layer, that allows systems to communicate over a local area network. Normally runs over TCP/IP.

Noise - a generic term for any addition to a typical attack to throw off anomaly detection. In the case of scan detection, using a variable list of ports. In relation to beacon detection modifying the beacon pulse to an slightly arrhythmic.

P2P - Peer-to-Peer - a network where all users (peers) are equally privileged participants and resources are shared between users.

Phishing - An attack vector where a threat actor will attempt to gain sensitive information by disguising themselves as a trusted or valid individual or organization.

Port Scan - An attack vector where a threat actor scans many ports on a computer or computer system by sending connection requests to those computer ports. Often a precursor to a future attack vector.

RDP - Remote Desktop Protocol - a protocol developed by Microsoft that provides a user with graphical control over a remote system.

Semi − Supervised Learning - A technique used by supervised learning algorithms to allow them to learn on data that another classifier has labeled.

Supervised Learning - One of two categories of machine learning methods it indicates that the training data is labeled.

Threat Actor - An individual or organization that attempts to gain unauthorized access to a computer or computer system through various attack vectors.

UDP - User Datagram Protocol - a transport layer protocol for passing data between systems.

Unsupervised Learning - One of two categories of machine learning methods, it indicates that the training data is unlabeled.

List of Abbreviations

Active Countermeasures is abbreviated ACM

Actual Intelligence Hunter is abbreviated AI-Hunter

Analysis of Variance is abbreviated ANOVA

Black Hills Information Security is abbreviated BHIS

Command and Control is abbreviated C2, and C&C in Table 4.3

ClickFraud is abbreviated CF in Table 4.3

Comma Separated Values is abbreviated CSV

Czech Technical University is abbreviated CTU

Distributed Denial of Service is abbreviated DDoS

Denial of Service is abbreviated DoS FastFlux is abbreviated FF in Table 4.3

False Positive is abbreviated FP

False Positive Rate is abbreviated FPR

False Negative is abbreviated FN

False Negative Rate is abbreviated FNR

Hypertext Transfer Protocol is abbreviated HTTP

Hypertext Transfer Protocol Secure is abbreviated HTTPS

Internet Control Message Protocol is abbreviated ICMP

Intrusion Detection System is abbreviated IDS

Indicators of Compromise is abbreviated IOC

Internet Relay Chat is abbreviated IRC in Table 4.3

k-Nearest Neighbors is abbreviated kNN

Machine Learning is abbreviated ML

Peer to Peer is abbreviated P2P in Table 4.3

Port Scan is abbreviated PS in Table 4.3

Remote Desktop Protocol is abbreviated RDP in Table 4.3

Random Forest is abbreviated RF

Real Intelligence Threat Analytics is abbreviated RITA

Receiver Operator Characteristic is abbreviated ROC

Support Vector Machine is abbreviated SVM

True Negative is abbreviated TN

True Negative Rate is abbreviated TNR

True Positive is abbreviated TP

True Positive Rate is abbreviated TPR

User Datagram Protocol is abbreviated UDP in Table 4.3

Code compiled and controlled by CTU Group is abbreviated US in Table 4.3

Index

Vita

Samuel Jacob Carroll

Born: February 9, 1993, Rapid City, South Dakota.

Education:

High school: Central High School, Rapid City, May 2011.

College: Bachelor of Science in Computer Science, South Dakota School of Mines and Technology, Rapid City, South Dakota, December 2015.

Work experience:

Black Hills Information Security: Software Development Intern May 2016-Present:

Tracking Research: May 2016-November 2016

C2 Research: November 2016-May 2017

Rita and AI-Hunter development: May 2017-Present