2006-09

Sample entropy and random forests a methodology for anomaly-based intrusion detection and classification of low-bandwidth malware attacks

Hyla, Bret M.

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/2633

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

SAMPLE ENTROPY AND RANDOM FORESTS: A METHODOLOGY FOR ANOMALY-BASED INTRUSION DETECTION AND CLASSIFICATION OF LOW-BANDWIDTH MALWARE ATTACKS.

by

Bret M. Hyla

September 2006

Thesis Co-Advisors:	Craig Martell
	Kevin Squire

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE Sample Entropy and Random Forests: A Methodology for Anomaly-based Intrusion Detection and Classification of Low-bandwidth Malware Attacks.		5. FUNDING NUMBERS
6. AUTHOR(S) Bret M. Hyla		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES   The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words) Sample Entropy examines changes in the normal distribution of network traffic to identify anomalies. Normalized Information examines the overall probability distribution in a data set. Random Forests is a supervised learning algorithm which is efficient at classifying highly-imbalanced data. Anomalies are exceedingly rare compared to the overall volume of network traffic. The combination of these methods enables low-bandwidth anomalies to easily be identified in high-bandwidth network traffic. Using only low-dimensional network information allows for near real-time identification of anomalies. The data set was collected from 1999 DARPA intrusion detection evaluation data set. The experiments compare a baseline f-score to the observed entropy and normalized information of the network.   Anomalies that are disguised in network flow analysis were detected. Random Forests prove to be capable of classifying anomalies using the sample entropy and normalized information. Our experiment divided the data set into five-minute time slices and found that sample entropy and normalized information metrics were successful in classifying bad traffic with a recall of .99 and a f-score .50 which was 185% better than our baseline.

14. SUBJECT TERMS Anomaly Detection, Data Mining, Intrusion Detection, Malicious Anomalies, Random Forests, Machine Learning, DARPA IDE			15. NUMBER OF PAGES 81
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500   Standard Form 298 (Rev. 2-89)
                                             Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

SAMPLE ENTROPY AND RANDOM FORESTS: A METHODOLOGY FOR ANOMALY-BASED INTRUSION DETECTION AND CLASSIFICATION OF LOW-BANDWIDTH MALWARE ATTACKS.

Bret M Hyla
Captain, United States Marine Corps
B.B.S., Texas A & M University, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2006**

Author: Bret M. Hyla

Approved by: Craig Martell
                Thesis Co-Advisor

                Kevin Squire
                Thesis Co-Advisor

                Peter Denning
                Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Sample Entropy examines changes in the normal distribution of network traffic to identify anomalies. Normalized Information examines the overall probability distribution in a data set. Random Forests is a supervised learning algorithm which is efficient at classifying highly-imbalanced data. Anomalies are exceedingly rare compared to the overall volume of network traffic. The combination of these methods enables low-bandwidth anomalies to easily be identified in high-bandwidth network traffic. Using only low-dimensional network information allows for near real-time identification of anomalies. The data set was collected from 1999 DARPA intrusion detection evaluation data set. The experiments compare a baseline f-score to the observed entropy and normalized information of the network. Anomalies that are disguised in network flow analysis were detected. Random Forests prove to be capable of classifying anomalies using the sample entropy and normalized information. Our experiment divided the data set into five-minute time slices and found that sample entropy and normalized information metrics were successful in classifying bad traffic with a recall of .99 and a f-score .50 which was 185% better than our baseline.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROLOGUE

In this chapter we will introduce the reader of a new approach in classifying low volume anomalies in network traffic. In addition, we will continue with discussion of Sample Entropy and Normalized Information. We will then provide a brief description of our hypothesis. We will continue with discussion of how we tested our hypothesis using the Random Forest Algorithm. We will conclude with a synopsis of the remaining chapters of this thesis.

B. BACKGROUND

Sample Entropy examines changes in the normal distribution of network traffic to identify anomalies. Normalized Information examines the overall probability distribution in a data set. Random Forests is a supervised learning algorithm which is efficient at classifying highly-imbalanced data. Anomalies are exceedingly rare compared to the overall volume of network traffic.

C. HYPOTHESIS

Our hypothesis is that the combination of Sample Entropy and Normalized Information will enable low-bandwidth anomalies to be identified in high-bandwidth network traffic. We anticipate that by only using low-dimensional network information it may in the future be able to allow for near real-time identification of anomalies. The data set the hypothesis was tested against was the 1999 DARPA intrusion detection evaluation data set. The experiments compared a baseline f-score to the observed entropy and normalized information of the network.

The Random-Forest algorithm is an unsupervised machine-learning algorithm which has proven capable at classifying highly-imbalanced data sets, but not in the field of intrusion detection. Our experiments will address whether the combinations of sample entropy and/or normalized information processed by a Random-Forest algorithm are capable, and the degree of capability in identifying low-bandwidth anomalies. These anomalies often avoid detection by standard anomaly-based intrusion detection systems.

D. ORGANIZATION OF THIS DOCUMENT

The remainder of this thesis is organized as follows. Chapter II will discuss intrusion detection systems, a few common anomalies, different machine-learning algorithms, and Stealth Watch an anomaly-based intrusion detection system. Chapter III will describe the design of the experiment and gathering of a data set needed to test our hypothesis. Chapter IV will analyze and discuss the results of the experiments. Chapter V offers conclusions and recommendations for future work and the Appendix contains the code used to transform the original data into a suitable format for the experiments.

II. BACKGROUND

A.   INTRUSION DETECTION

An intrusion to a computer or computer network is defined by [Canavan. 2001] as "an unauthorized attempt or achievement to access, alter, render unavailable, or destroy information on a system or the system itself." Network administrators were previously able to review various logs on a daily basis to check for intrusion attempts. However, given the growth of the Internet and the volume of traffic now being generated on a networK-means waiting for daily checks is too late. Another approach is required.

Intrusion Detection Systems (IDSs) began as research projects for the US government in the early 1980's. In, 1980 James Anderson published the first paper in which he describes an effort to improve the computer security auditing and surveillance of a network. In his paper,[Anderson. 1980] the threat was broken into four categories:

1.   External Penetration

An individual from outside the organization attempting to gain access to computer network resources; also an employee who has physical access but is not an authorized computer user.

2.   Internal Penetration

Anderson breaks this type of penetration into three subgroups. He claims that this variant of threat is more prevalent than an external threat.

a.   *Masquerader*

This is a user who has gained a proper user identification and a corresponding password.   Locating

this type of user can be attempted by looking at audit records for deviations from normal activity for a given user.

b. *Legitimate User*

Misuse of authorized access to the computer network. This might be reveled in an audit log if the user is accessing data for which they do not have authorization. Once again, a normal profile of user activity on the computer system is required to locate anomalies.

c. *Clandestine User*

This is a user who can obtain administrator control of a computer and delete or alter the audit trail. Here having a reference model of the operating system with which to compare the current state of the machine is key to detection. Storing audit records in central location not on the local machine is another approach which makes hiding the activity of a clandestine user much more difficult.[Anderson. 1980]

B. TYPES OF INTRUSION DETECTION SYSTEMS

Unlike a firewall, intrusion detection systems do not block unauthorized packets based on a rule set. An IDS instead analyzes the packet header and packet content and makes a determination of legitimacy. If a packet is deemed malicious an alert is generated, allowing a system administrator to examine the packet.

Intrusion Detection Systems come in two basic types: host-based and network-based. The following two sections describe these two types in general terms.

1. Network-based Intrusion Detection

Network-based intrusion detection systems (NIDS) analyze network packets, compare packet structure to know

malware patterns, search an internal rule set, then make a determination of misuse, and if necessary generate an alarm which is reported to a centralized location. A NIDS has several advantages. It is effective at detecting outsiders attempting to penetrate the network, one or at most a few sensors are all that is required to provide coverage for the entire network, and since the NIDS is listening for all network traffic, it is positioned to detect attacks directed at any host on the network. A NIDS, if configured appropriately also has the potential to stop an attack prior to reaching the hosts. A NIDS generally runs on a specially built machine so it does not degrade the computing resources of individual systems. NIDS have two approaches they use to classify an intrusion.

a. Signature-based IDS

This system matches a know signature or pattern that was generated by the IDS vendor. The rule set is stored locally in every instance of the IDS. It is also commonly referred to as rule-based intrusion detection (RBID). When a new attack pattern has been observed it is analyzed and a new rule is generated by the vendor. The vendor notifies customers that an "update" is available so their instance can have the most current rule set. Most vendors sell systems which can be configured to automatically check with the vendor for updates and automatically install them. This method ensures the IDS always has the vendors most current rule set and a network administrator does not have to spend the time to check on a daily basis.

b. Anomaly-based IDS

In 1987, Denning [Denning. 1987] described a model for a real-time intrusion detection system which

built on Anderson's work. Her hypothesis was that security violations in the network could be discovered by searching for abnormal patterns of use in the system. The model created a profile based on statistical metrics for every subject in the system and compares the baseline profile to current activity searching for deviations from the profile. Anomaly-detection defined by Bace is "using statistical techniques to find patterns of activity that appear to be abnormal."[Bace. 2001] These patterns of activity are evaluated for possible signs of malicious activity. While these systems have great potential to defend against new or unknown attacks, determining what traffic is abnormal is still a great challenge. In 2000, Lancope Corporation released StealthWatch, one of the first anomaly-based IDS. [Lancope. 2006]

2. Host-based Intrusion Detection systems (HIDS)

A host-based IDS is designed to monitor and analyze data that originates on the individual system on which if installed. HIDS are particularly effective at detecting misuse of the system by an authorized user.[Proctor. 2001] HIDS have several different sources of data available on the host, system logs, audit logs, listing of active processes, keystroke monitoring, and packet throughput.

There are several advantages to HIDS including:

- Actual results of an attack or user misuse of system are available.
- Less reliance on a set of rules.
- Higher likelihood of detecting an unknown attack.
- Insiders knowing they are being monitored are less likely to misuse the system.
- No additional hardware requirement.
- Encrypted network traffic is accessible for analyzing.

HIDS by themselves have numerous disadvantages. The most critical is that a HIDS can have a significant performance impact on the host. If the host is compromised the system logs, if stored locally, are subject to manipulation. Some attacks, like a buffer overflow, are not likely to be logged. Finally, if the system is compromised the monitoring can be terminated or nullified. [Crothers. 2003] Therefore, good-quality audit sources are critical. These logs should be created by a trusted source, be of sufficient detail to recreate every event, and stored off-host to protect their integrity.[Proctor. 2001]

Both network-based IDS and HIDS have advantages and drawbacks to specific attack methods but together they create a much more effective network defense than either alone. Some examples of malicious activity likely to be found in a current network that NIDS and HIDS can help discover and prevent is presented in the next section.

C.   MALICIOUS ACTIVITY

Malicious activity can be defined as an intentional attempt to bypass computer security measures in some fashion. [Crothers. 2003] Users may attempt to download music files from a common peer to peer files sharing system like KaZaA in violation of company policy. They may install an internet shareware game on their computer which has a network scanner embedded inside of it. A user could open an attachment from an unknown user asking the user to "click", which, while displaying the funny video, enables a worm to be loaded into the local system. In the following section worms network scanners, peer-to-peer software and network scanners will be described.

1. Worm

A worm is a self replicating computer program that is self-contained and does not need any other software program to replicate. The name "worm" comes from *The Shockware Rider*, a science fiction novel written by John Brunner in 1975. The first worm on a worldwide network was the Christmas Tree Worm released in December 1987, which spread across IBM's network and BITNET. [Erbschloe. 2005] The power of the worm was such both networks were severely affected. A worm has four primary qualities: a propagation mechanism, transport an executable piece of code, identify additional machines vulnerable to the worm and various means to attempt to avoid detection. This combination of attributes makes a worm appealing to malicious users.

2. Peer-to-Peer

These are programs that allow you to connect to other users to share files, instant message other users text, voice messages or files, and conduct distributed processing, which utilizes the unused computing power on your local computer to create huge computing power capacity. They also allow you to create a network to upload and download material; this is often music, video and games. This ability to upload and download material is of great concern to network security personnel. Files that are downloaded can contain additional content; this content can be spyware, viruses, Trojan horses, or worms. Once the file is downloaded the system can then be exploited and serve as a zombie or malware server to spread malicious code inside the local area network. These applications also allow others to receive access and place files on your

local machine without your knowledge. Continuously scanning the network for any signs of peer to peer activity can help eliminate a common attack vector for malware. One of the more dangerous types of malware that is now exploiting peer to peer systems are worms, I will give an example of one recently identified worm in the next section.

3. Service Discovery

This is an attempt by an unauthorized user or piece of software to discover what applications or computers exist in your network. This informs the attacker which computers are turned on and what ports they are listening for network traffic on. Service discovery is utilized by all levels of hackers. SuperScan by FoundStone and Nmap by Insecure are two popular tools for service discovery. Network security personnel should be concerned if this type of activity is detected on the network. Scanning is an indicator that service discovery is or has taken place and the attacker can now craft an exploit specifically designed to exploit vulnerabilities found in your network.

D. MACHINE-LEARNING ALGORITHMS

There are several machine-learning algorithms that have been created to attempt to find patterns and anomalies in data sets. The following sections will describe a few of them in detail.

1. CART

The classification and regression tree (CART) is a general framework in which for a given set of data can be broken into smaller subsets determined by category labels. Each split is designed to select the best label to split upon with the goal of creating a subset of data with the exact same categorical values. Data subsets that are not

pure are called nodes, and splitting continues until a data set ideally contains only the same categorical values. This subset is deemed "pure" and splitting is halted on the given subset of data. The subset is also classified as a leaf of the tree. In highly variable data, a floor can be set on the splitting function based on the observations in a node, this prevents an expanse of leafs with only one observation. [Duda. 2001]

2.   K-MEANS

The K-means algorithm was introduced by 1982. [Lloyd. 1982] It remains quite popular due to its simplicity and speed. The K-means procedure works as follows. Given a set of n size data points, partition the data points in k clusters based on local search. A random set of initial k cluster centers is chosen. Each point is assigned to the closest cluster center determined by minimizing the sum of Euclidean distance of its features. The centers of the clusters are recomputed based on the new set of data points in the given cluster. The procedure is repeated until all points are assigned to the cluster that minimizes is Euclidean distance. The clusters with their data points are then returned from the procedure.[Arthur, et al. 2006]

3.   Hierarchical Agglomerative Algorithm

The hierarchical agglomerative procedure clusters data points as follows. Given n data points, assign each to its own cluster. The procedure then searches the space for the two clusters having minimum Euclidean distance between the vectors.   The procedure continues until all clusters have been joined into one cluster containing all data points. Alternatively, you can force a floor on the number of clusters. [Lakhina, et al. 2005]

4. Random Forests

Classification accuracy has seen large improvements by growing a number of trees and having them vote for the most popular class. Random vectors are used to govern the growth of individual trees. Breiman demonstrated an early form of this method in 1996 with his introduction of bagging.[Breiman. 1996] In bagging, trees are grown from the training set by taking random examples from the set. Dietterich and Breiman continued to refine the randomness in [Dietterich. 1998]and [Breiman. 1998]. Ho proposed using "the random subspace" method to take a random subset of features to grow individual trees, [Ho. 1998] because Random Forests are used extensively in our experiments, we will describe them more below.

a. Formal Definition

Random Forests were formally defined in 2001 as:

A classifier consisting of a collection of tree-structured classifiers {h(x, Θ_k), k=1,…} where the {Θ_k} are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x. [Breiman. 2001] The random vector is defined as Θ. The nature and dimensionality of Θ depends on its use in tree construction.

b. Overfitting

Breiman proves with Theorem 1.2 in [Breiman. 2001] that if you have a large number of trees, the Strong Law of Large Numbers and the tree structure will ensure that Random Forests will not overfit as additional trees are added, rather the additional trees limit the value of the generalization error.

5. Random Forests vs. Adaboost

Research into Random Forests explored various methods to lower the generalization error.[Dietterich. 1998][Breiman. 1998, Freund, et al. 1996] [Bauer, et al. 1999]

Adaboost was the benchmark to compare any implementation of a Random Forest. Breiman worked to improve accuracy by injecting randomness to minimize the correlation p while maintaining strength. Breiman's class of random trees had five promising characteristics:

- Accuracy equal or better to Adaboost

- Robust handling of outliers and noise

- Faster than bagging or boosting

- Provides internal estimates of error, strength, correlation and variable importance

- Simple and easily parallelizable

a. Empirical Experiments

Breiman conducted several experiments using 16 data sets from the University of California Irvine repository. Breiman compared two means of growing Random Forests, in both a random 10% of the data was set aside. A Random Forest was grown to a size of 100 trees, where F is the number of inputs to split on. The experiments were run twice, once with F=1 and the second time with F equal to the result of equation (1.1), where M is the number of inputs.

$$F = \text{int}(\log_2 M + 1) \tag{1.1}$$

Each method was run 100 times and the test-set errors were averaged. For a fair comparison the same procedure

was used to separate the data and 50 trees were combined for the Adaboost runs. Breiman's results showed that Random Forest using a random input(Forest-RI) selection were comparable to Adaboost with the added advantage of being much faster. A Forest-RI took four minutes to execute where Adaboost took approximately three hours.

Breiman modified the random input concept by defining more features by taking random linear combinations of a subset of the input variables. This version of the Random Forest was called Forest-RC. Forest-RC did better compared to Adaboost than Forest-RI.

b. *Noise*

Additional experiments to determine how sensitive Random Forests were to mislabeled data, aka "noise", when compared to Adaboost. Adaboost had a sharp decrease in classification with 5% noise, while for both Random Forests procedures noise had only minor changes.

c. *Conclusions*

Breiman demonstrated in 2001 that Random Forests are an effective tool in predication. Overfitting is not an issue. Breiman's results demonstrated Random Forests are at least as accurate as other machine-learning algorithms. Another advantage of Random Forests is that the training set is not altered throughout the procedure as is the case with bagging and boosting.[Breiman. 2001]

E. CHOOSING A MACHINE-LEARNING ALGORITHM

Caruana and Niculescu-Mizil in 2006, completed a comprehensive empirical study on learning algorithms. This was the first large scale comparison since King conducted the STATLOG study in 1995. They examined 10 supervised

13

learning methods compared them with 8 different performance metrics. The results are detailed in Table 1.

MODEL	CAL	COVT	ADULT	LTR.P1	LTR.P2	MEDIS	SLAC	HS	MG	CALHOUS	COD	BACT	MEAN
BST-DT	PLT	.938	.857	.959	.976	.700	.869	.933	.855	.974	.915	.878*	.896*
RF	PLT	.876	.930	.897	.941	.810	.907*	.884	.883	.937	.903*	.847	.892
BAG-DT	−	.878	.944*	.883	.911	.762	.898*	.856	.898	.948	.856	.926	.887*
BST-DT	ISO	.922*	.865	.901*	.969	.692*	.878	.927	.845	.965	.912*	.861	.885*
RF	−	.876	.946*	.883	.922	.785	.912*	.871	.891*	.941	.874	.824	.884
BAG-DT	PLT	.873	.931	.877	.920	.752	.885	.863	.884	.944	.865	.912*	.882
RF	ISO	.865	.934	.851	.935	.767*	.920	.877	.876	.933	.897*	.821	.880
BAG-DT	ISO	.867	.933	.840	.915	.749	.897	.856	.884	.940	.859	.907*	.877
SVM	PLT	.765	.886	.936	.962	.733	.866	.913*	.816	.897	.900*	.807	.862
ANN	−	.764	.884	.913	.901	.791*	.881	.932*	.859	.923	.667	.882	.854
SVM	ISO	.758	.882	.899	.954	.693*	.878	.907	.827	.897	.900*	.778	.852
ANN	PLT	.766	.872	.898	.894	.775	.871	.929*	.846	.919	.665	.871	.846
ANN	ISO	.767	.882	.821	.891	.785*	.895	.926*	.841	.915	.672	.862	.842
BST-DT	−	.874	.842	.875	.913	.523	.807	.860	.785	.933	.835	.858	.828
KNN	PLT	.819	.785	.920	.937	.626	.777	.803	.844	.827	.774	.855	.815
KNN	−	.807	.780	.912	.936	.598	.800	.801	.853	.827	.748	.852	.810
KNN	ISO	.814	.784	.879	.935	.633	.791	.794	.832	.824	.777	.833	.809
BST-STMP	PLT	.644	.949	.767	.688	.723	.806	.800	.862	.923	.622	.915*	.791
SVM	−	.696	.819	.731	.860	.600	.859	.788	.776	.833	.864	.763	.781
BST-STMP	ISO	.639	.941	.700	.681	.711	.807	.793	.862	.912	.632	.902*	.780
BST-STMP	−	.605	.865	.540	.615	.624	.779	.683	.799	.817	.581	.906*	.710
DT	ISO	.671	.869	.729	.760	.424	.777	.622	.815	.832	.415	.884	.709
DT	−	.652	.872	.723	.763	.449	.769	.609	.829	.831	.389	.899*	.708
DT	PLT	.661	.863	.734	.756	.416	.779	.607	.822	.826	.407	.890*	.706
LR	−	.625	.886	.195	.448	.777*	.852	.675	.849	.838	.647	.905*	.700
LR	ISO	.616	.881	.229	.440	.763*	.834	.659	.827	.833	.636	.889*	.692
LR	PLT	.610	.870	.185	.446	.738	.835	.667	.823	.832	.633	.895	.685
NB	ISO	.574	.904	.674	.557	.709	.724	.205	.687	.758	.633	.770	.654
NB	PLT	.572	.892	.648	.561	.694	.732	.213	.690	.755	.632	.756	.650
NB	−	.552	.843	.534	.556	.011	.714	-.654	.655	.759	.636	.688	.481

Table 1   Normalized scores of each learning algorithm by problem(averaged over eight metrics) (From Ref. [Caruana, et al. 2006])

Uncalibrated Random Forests performed best at the precision/recall break even point and accuracy metrics and across three of the data sets. Calibration of a Random Forest only provided a small improvement. [Caruana, et al. 2006] In 2004, Random Forests were used in classifying data sets with highly-imbalanced classes. Often the interest is in ensuring correct classification of the "rare" class. The way the Random Forest classifier works is to assign a weight to each class, with the rare class given the larger weight.   Weighting occurs twice, once for weighting where to split and then in the terminal node.

14

The classification for each node is voted upon in a "weighted majority vote." The number of cases in the node is multiplied by the weight given to the class of the case and the node is classified by taking the higher weight class. Random Forests proved to be more robust than CART 4.5, Neural Nets or SHRINK in classifying highly-imbalanced data sets. [Chen, et al. 2004] Therefore, Random Forest algorithm as implemented by Breiman and Cutler in the Salford Systems Random Forest v1.0 package will be used to classify anomalies in our experiment.

F.   RELATED WORK

The following section will discuss related work that has been done in the problem area of anomaly-detection and classification with intrusion detection.

The majority of recent approaches to classify anomalies from network traffic information have focused on the changes in volume of network traffic as a key metric.[Duan, et al. 2005, Hong Han, et al. 2002, Jaroszewicz, et al. 2005, Jian Yin, et al. 2004, Julisch, et al. 2002, Khanna, et al. 2006] However, as seen in Table 2, anomalies also impact the traffic-feature distributions in differing ways.

Anomaly Label	Definition	Traffic Feature Distributions Affected
Alpha Flows	Unusually large volume point to point flow	Source and destination address (possibly ports)
DOS	Denial of Service Attack (distributed or single-source)	Destination address, source address
Flash Crowd	Unusual burst of traffic to single destination, from a "typical" distribution of sources	Destination address, destination port
Port Scan	Probes to many destination ports on a small set of destination addresses	Destination address, destination port
Network Scan	Probes to many destination addresses on a small set of destination ports	Destination address, destination port
Outage Events	Traffic shifts due to equipment failures or maintenance	Mainly source and destination address
Point to Multipoint	Traffic from single source to many destinations, *e.g.*, content distribution	Source address, destination address
Worms	Scanning by worms for vulnerable hosts (special case of Network Scan)	Destination address and port

Table 2   Qualitative effects on traffic feature distributions by differing anomaly types. (From Ref. [Lakhina, et al. 2005])

There are several types of anomalies that have very little impact on the volume of network traffic and thus can escape a volume approach to anomaly-detection. Therefore a different approach must be undertaken to locate low volume anomalies in network traffic. [Lakhina, et al. 2005] Lakhina's hypothesis was that anomalies induce a change in the OD flow. A worm will skew distribution for the destination addresses, and a skewed distribution for the target port the worm is scanning.

Several machine-learning algorithm approaches have been utilized in classifying anomalies in network traffic, but Random Forests have not been thoroughly studied for their effectiveness in classifying anomalies. [Yang, et al. 2004, Zhao Junzhong, et al. 2002, Ren, et al. 2004, Chavan, et al. 2004, Colombe, et al. 2004] Random Forests have been very successful in other domains in classifying

16

highly-imbalanced data.   [Chen, et al. 2004, Ham, et al. 2005, Jian Xue, et al. 2006]

This paper will use the definition from [Lakhina, et al. 2004] to describe traffic features. A traffic feature is a field in the header of a packet. Four fields from the header will be used to identify anomalies: source address denoted (sIP), destination address denoted (dIP), source port denoted (sPort), and destination port denoted (pPort).

A method to measure the uncertainty of a given discrete event occurring based on a set of observed distributions was first described in 1949.[Shannon, et al. 1949]   This metric as described is known as sample entropy. Starting with a discrete set of symbols {s_1, s_2 … s_n} with associated probabilities P_i, the entropy of the discrete distribution is a measure of randomness in the sequence of symbols drawn from it is shown in equation (1.2).

$$Sample\ Entropy = -\sum_{i=1}^{n} P_i \ \log_2 P_i \qquad\qquad (1.2)$$

The value of sample entropy lies in the range (0,$\log_2$ n).   Note, entropy does not depend on the symbols themselves, just on their probabilities. With a given number of symbols s, the uniform distribution in which every symbol is equally likely to appear, is the maximum entropy distribution   and   H= $\log_2 m$. Minimum entropy distribution   occurs   when   distribution   is   totally concentrated, here the metric takes on a value of H = 0. [Duda. 2001]

Sample entropy can be used to estimate the actual entropy of the random behavior of 1999 DARPA data set. This paper will not assume to capture the actual randomness behavior of all five weeks of the 1999 DARPA data traffic. Rather we will use sample entropy as a metric to capture the frequency tendency of the distribution of only the observed data set.

In this thesis, sample entropy is computed from feature distributions gathered from probe counts. Sample entropy's range of values depends on the number of distinct values seen in the observed data set.

We also calculate another metric which we call Normalized Information, from equations (1.3) and (1.4). Information is calculated by finding the data set frequency distribution for a feature. Let P_i be the probability of a feature occurring in the overall data set. The value of information in a five-minute time slice is normalized by the average number of bytes per packet in a given time slice as see in equation (1.4).

$$Infomation \ = \ -\sum_{i=1}^{n} \log_2 P_i \qquad\qquad (1.3)$$

$$Normalized \ Information \ = \ \frac{Information}{Avg \ Num \ Bytes \ Per \ Packet} \qquad (1.4)$$

G. CHAPTER SUMMARY

In summary, this chapter described, at a high level, differing ways to intrude into a computer network and systems designed to detect that behavior. In addition, three types of malicious activity were described. Four

machine-learning algorithms were introduced with the Random Forest algorithm covered in greater detail. The reasons for choosing the Random Forest algorithm as our classifier was also discussed. Finally, sample entropy and normalized information were defined.

THIS PAGE INTENTIONALLY LEFT BLANK

III.   DESIGN OF EXPERIMENT

In the last chapter we described the several malicious attacks and various machine-learning algorithms. In this chapter, we will describe how we generate a matrix of vectors for each data set. We then describe how we used this labeled data to run a series of Random Forest experiments with the goal of predicting classification labels from the predictor vectors.

A.   EXPERIMENTAL OVERVIEW

Machine-learning algorithms have been utilized in anomaly-detection experiments. However, prior to May of 2006, there had not been any published results using Random Forests. Since the ratio of attack traffic to normal traffic is highly-imbalanced, we selected the Random Forest algorithm.

The remainder of this chapter details the data set, code used transform the data set, software packages used in conducting the experiments, and problems encountered in conduct of the experiment.

B.   DATA SET

To run our experiments we used traffic generated by the MIT Lincoln Labs as part of the DARPA 1999 IDS evaluation. [Zissman. 2002] The evaluation has five weeks of traffic, divided into two sections. The first portion of the evaluation is three weeks of training data. Only the second week of this data contained attacks. The second portion of the evaluation is two weeks of test data. Each week of data had five days of traffic. Traffic was collected at two points in the network, inside and outside the boundary router. Data collection began approximately

8am each day and stopped at 6am the following day this means each block of traffic contained approximately 22 hours of traffic.   The simulation was then shut down for maintenance and restarted at 8am for the next day's data block.

Collected Data

We took the inside and outside tcpdump data from second week of training data and the first week of test data for our experimental data.   There was an error on the second day of test week when the network traffic sniffer did not collect inside traffic. The first day of the second week of test data was used to ensure that there was five days of traffic from both the training and test data for evaluation.

C.   TRANSFORMING THE DATA

The DARPA data contained full Ethernet packets.   To run the experiments, we needed to extract six features from each packet: timestamp, source IP address, destination IP address, source port number, destination port number, and number of bytes in the packet. All code used to transform the data can be found in Appendix A.

1.   Sampling

To simulate sampling from live traffic only every tenth packet was chosen for the experimental data set. Sampling was conducted separately on the inside and outside tcpdump files. This sampling was done with Sampler.java.

2.   Extracting Features from Full Packet Data

We explored various means for extracting the six features from a packet.   Network traffic viewers like Wireshark and Tcpdump were tested for ability to extract the features and were found to be excellent on filtering on the information match of a particular feature.   This

products could not just extract information from an identified feature in the packet, therefore another tool was required. Additional research located SiLK: The Analyst's Suite. [SourceForge. 2006] This suite was created by Carnegie Mellon University's Computer Emergency Response Team (CERT) to examine traffic throughout a network, observe malicious activity, trace server behavior and other analytical tools. [SourceForge. 2006]The SiLK software package currently only works on LINUX type operating systems. We utilized three analytical tools to extract the features from the tcpdump data files.

a. Converting TCPDUMP to SiLK files

We had to first convert the data from Tcpdump file format into a SiLK flow record. SiLK flow records collapse fragmented packets into one flow record. This allows for addition of OSI layer four information to the flows. We used the rwptoflow utility to make the conversion. Figure 1 shows the command line used to transform a file into a SiLK flow record.


```
C:\>
C:\>C:\DARPA data\test data>rwptoflow inside8Mar.tcpdump > inside8Mar.rw
```

Figure 1. Rwptoflow file conversion

b. Extracting Features from a SiLK File

Once we had the data in a SiLK file, we were able to employ the rwcut to print selected fields to a new file. We used the –delimited option to utilize a comma to separate output files and the -fields option to select the fields to be sent to the output file. Figure 2 provides an example of the command.

```
C:\>
C:\>rwcut inside8Mar.rw -delimited=, -fields=sIP,dIP,sPort,dPort,bytes, eTime
 > inside8Mar.txt
```

Figure 2. Rwcut command to extract selected fields to a
designated output file

The data was now in a flat file, with each field separated by commas, and had only one packet per line. The last field also has a comma after it to allow for ease of reordering fields if necessary. Figure 3 shows one line of output ordered per the rwcut utility: Source IP, Destination address, Source Port, Destination Port, number of bytes in the packet, and the start time of the packet. Time is formatted with 24 hour numbering and the time zone is Greenwich Time.

```
196.37.75.158,172.16.112.194,25,1024,15872,1999/03/29T13:00:04.106,
```

Figure 3. Extracted data fields

3. **Calculating Aggregate Data for a Given Five-minute Time Slice**

a. *Sorting the Data by Time*

We now had the data in separate flat files and needed to combine them into one large file ordered by their timestamp. We used the DOS copy command to append the files into one large file. We created two small Perl programs called Reorder.pl and Sorter.pl to reorder the fields with the timestamp first, this allows the Sort.pl to sort the data on that field and output the data back into the same file in ascending order.

b. *Placing Packets in Five-minute Time Slices*

We utilized a Perl package called Date::Calc which allows for comparison of two dates. The package contains a Delta function to determine the difference between two times. We decided to split our data into five-minute time slices to allow for comparison to related work. We created an array of arrays, each array contained five-minutes worth of packets. We used Entropy_Info.pl to do this comparison. Figure 4 illustrates the average number of packets calculated per time slice.

Week 2 Average Number of Packets per 5 Minute Time Slice
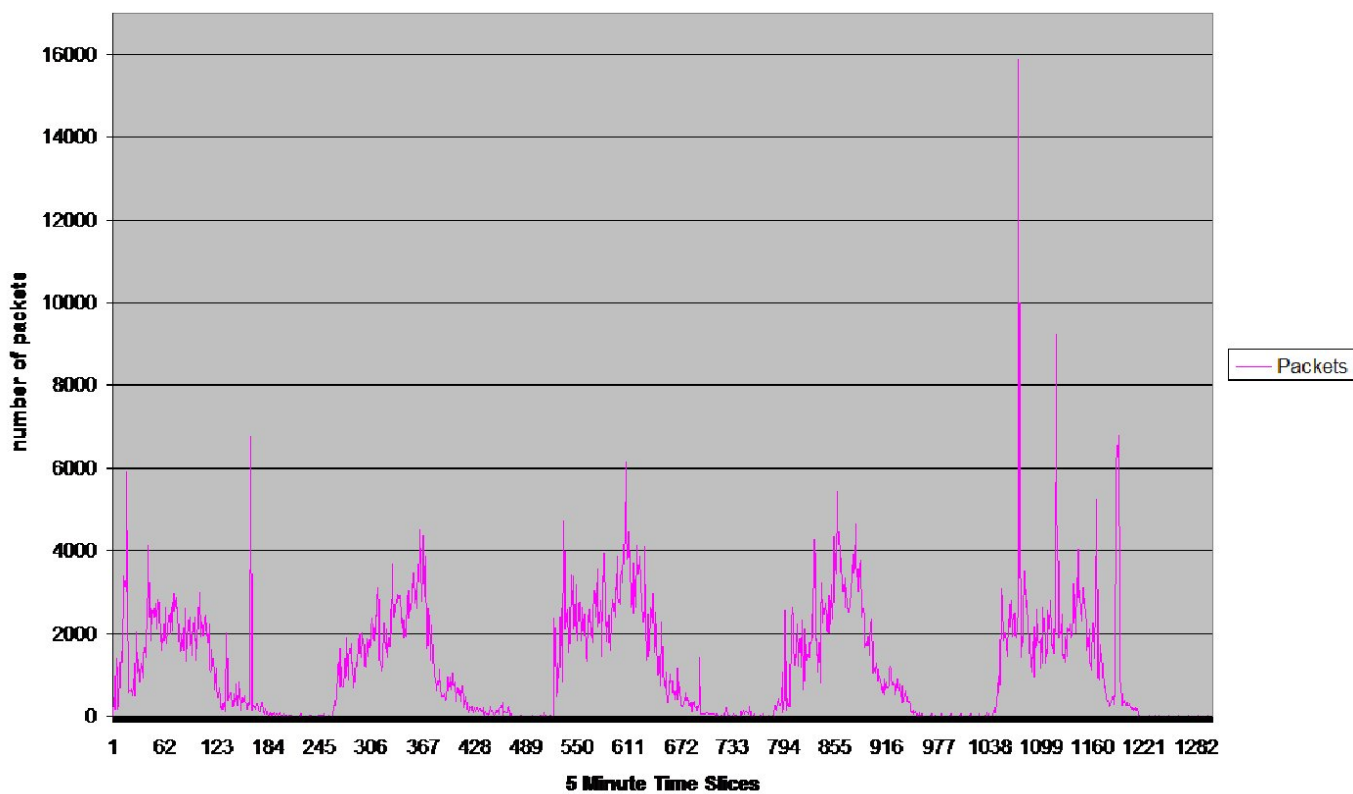
Figure 4. Average Number of Packets per Time Slice

c.   *Overall Probability Distribution*

We created a hash table for Source IP, Destination address, Source Port, Destination Port. We also counted the total number of packets in the file. We are able to determine for each unique key in the hash table, the number of counts for that key and the overall probability for the key in the distribution. The key is associated with the overall probability and they are written to a new hash table. We used Entropy_Info.pl to do this comparison.

d.   *Five-minute Time Slice Information*

We examine each five-minute slice of traffic for each of the four features. For each unique instance of a feature, we find the $\log_2$ of that instance's probability from the overall distribution calculated earlier and sum them for the overall information contained in the five-minute time slice. We used Entropy_Info.pl to do this comparison.

e.   *Normalizing Information*

Since the number of packets in each time slice varied greatly we needed to normalize the information by dividing each raw feature value by the number of packets in a time slice. This calculation was done using Excel.

f.   *Five-minute Time Slice Entropy*

We examine each five-minute slide of traffic for each of the four features. For each unique instance of a feature, we calculate the instance's probability from the five-minute probability distribution. This probability is multiplied by the $\log_2$ of the probability and summed for the entropy of the five-minute time slice. We used Entropy_Info.pl to perform the calculations.

g. *Average Number of Bytes per Packet in a Five-minute Time Slice*

We calculated the average number of bytes in the time slice by determining the total number of bytes in a time slice and simply dividing by total number of packets in a time slice. We used Bytes.pl to extract total number of bytes per time slice and imported that data into the Excel spreadsheet containing the other data

D. RANDOM FORESTS

This section will describe the basic setup of Salford Systems implementation of Random Forests (RF). A trial version of this software package is available for 30 days.

1. Variable Selection

We loaded our data in a CSV format. Figure 5 shows the initial menu after the data is loaded. We would select the four predictor variables and a target variable. The target variable will be what the RF attempts to classify based off of the predicator variables.
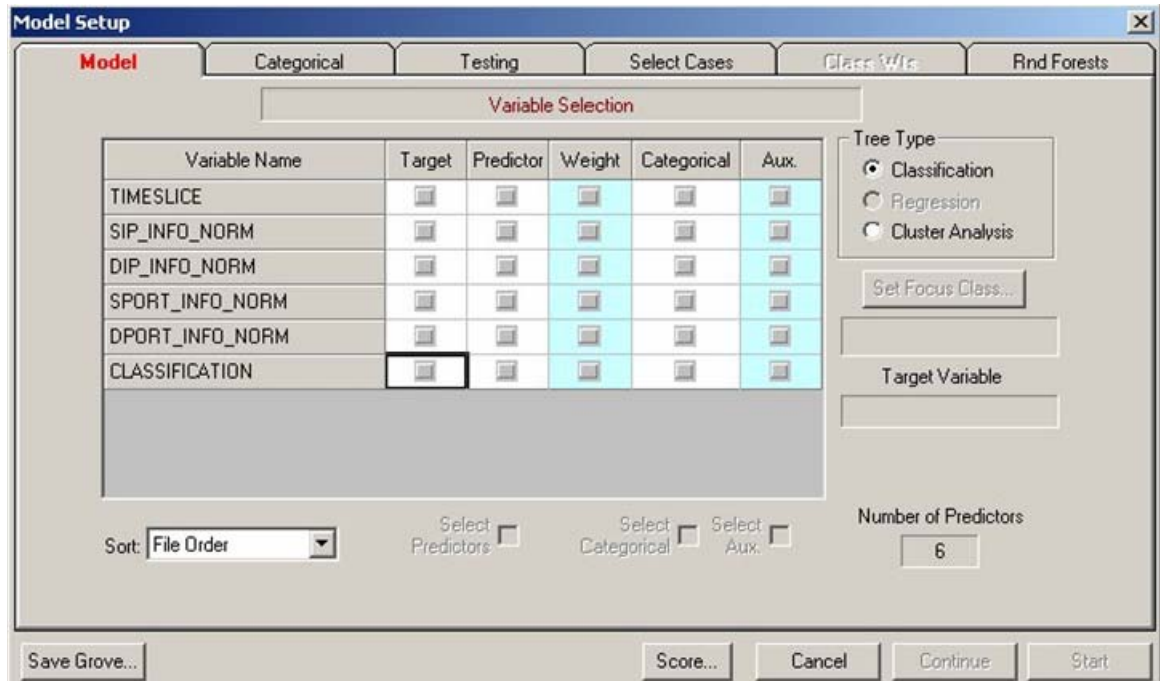
Figure 5. Random Forest Variable Selection

2. Testing

One of the major advantages of RF is it does not modify the original data set. RF by default uses

Out-of-Bag data for testing. It does this by pulling out approximately one-third of the data for self-testing. This is an extension of cross-validation which is repeated several hundred times. This ensures a high reliability. Figure 6 shows how the weights of each class can be modified. Figure 7 shows how you can modify the testing process if desired.

If classifying one class is important, weighting for that class can be specified orders of magnitude higher than other classes.
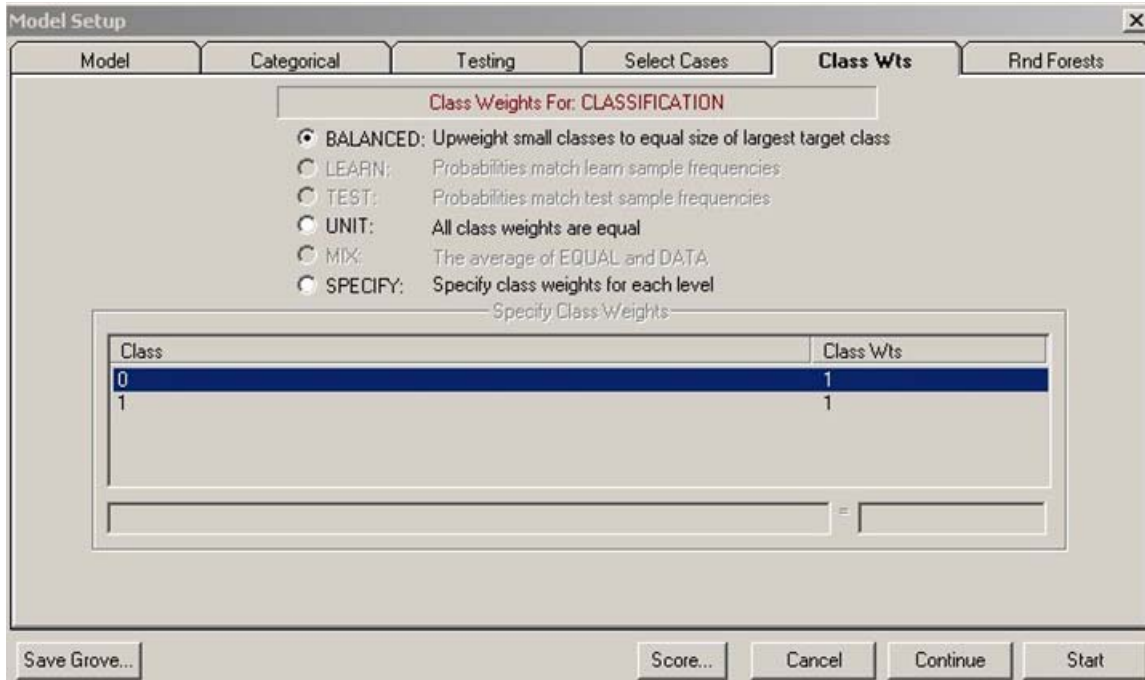
Figure 6. Weighting Choices for training and testing
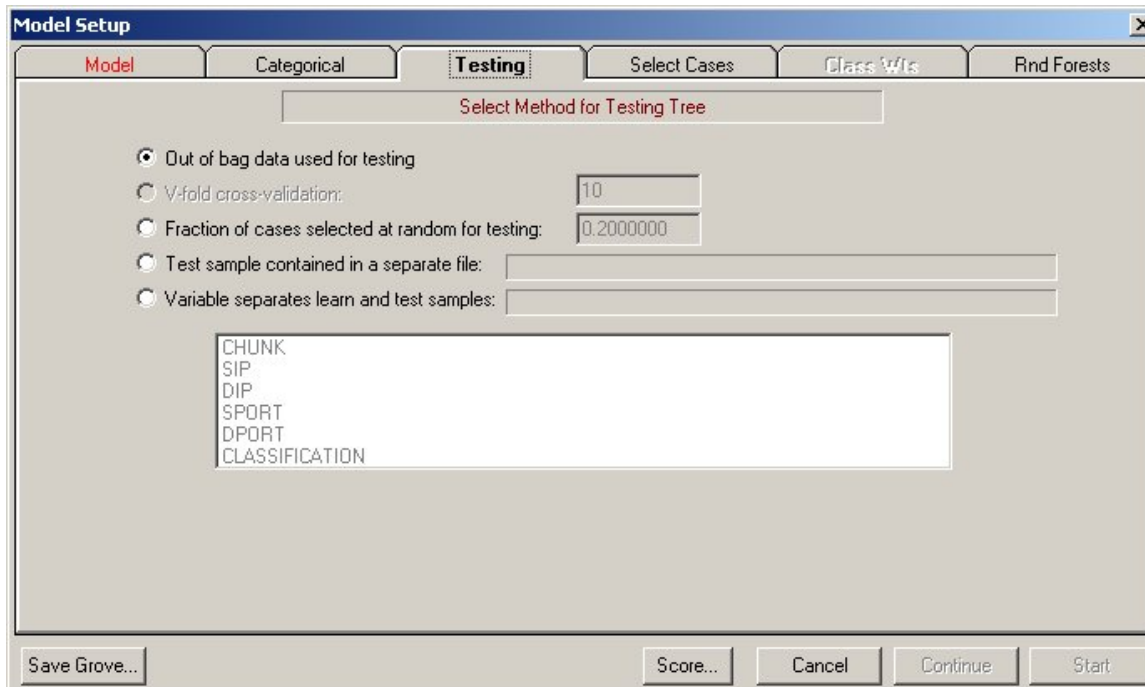
Figure 7. Options for Testing the Forest

3.   Random Forest Tree Choices

The next screen allows the tester to choose the number of trees to be grown, number of predictors for a node, the size of the bootstrap sample. Figure 8 shows this clearly. The manual recommends that the number of predicators for each node should be the square root of the total number of predictors.
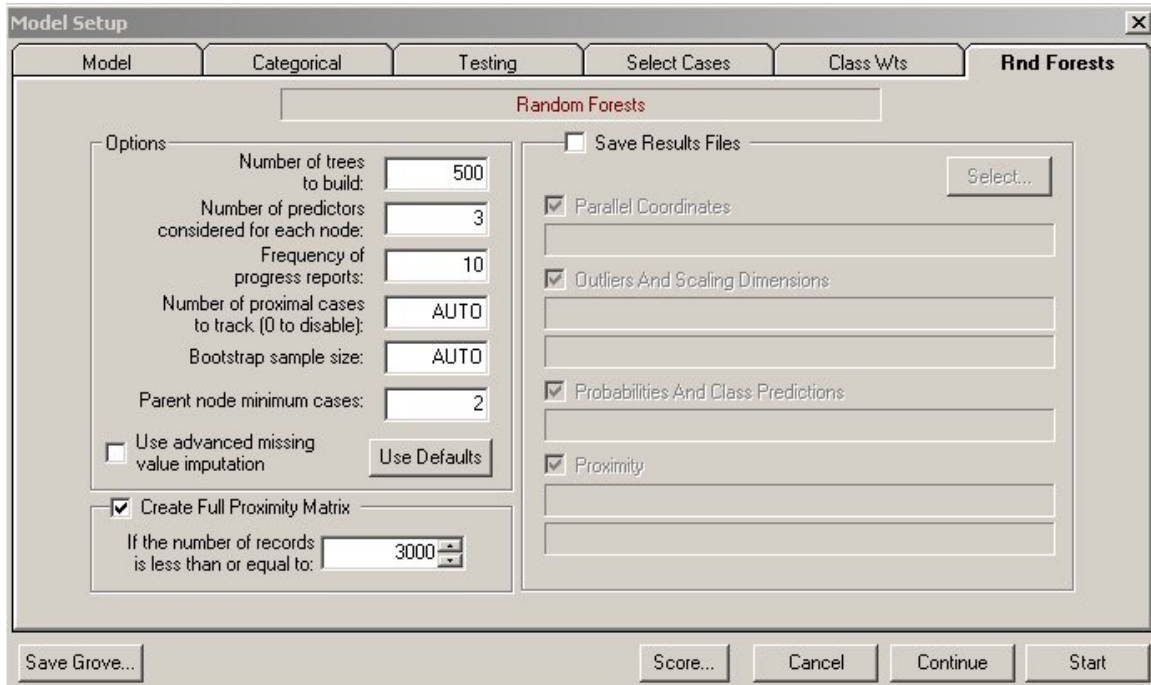
Figure 8. Options for Testing the Forest

3.   Experiment Parameters

Several experiments were run on four combinations of the data sets. The four data sets are as follows: Normalized Information, Sample Entropy, Normalized Information and Sample Entropy, and Normalized Information with Sample Entropy and average number of bytes per packet which we defined in Chapter II.

a. *Variables*

All available predicators in each data were utilized. Classification was always the target variable.

b. *Model Adjustments*

Each of the following parameters in the model were adjusted independently: The number of trees to be built was varied from 100, 250, 500 and 1000. For each size of the forest, class weights were adjusted between balanced classes and weighting attack traffic to 10, 100, and 1000. The weight of normal traffic was kept constant at 1.

E. PROBLEMS DISCOVERED IN CONDUCT OF THE EXPERIMENTS

There were a few unexpected problems that occurred while we conducted these experiments that should be noted.

1. DARPA 1999 Training Data

The week two training data which contains the attack sequences only lists the starting times for the attacks. In the test data the duration of attacks were also recorded. This proved to be significant as several attacks spanned greater than 10 minutes. This meant that one of these attacks must result in multiple five-minute time slices being treated as containing attack traffic. Therefore the second week of training data was not utilized in obtaining our experimental results.

2. Stealth Watch

Data was initially collected from the Naval Postgraduate School's network. Stealth Watch stores probes for 30 days, which would allow for a robust data set. A careful examination of the probe data set showed that only highly suspicious probes were present in the data set. Using this data would not provide the correct balance of normal to attack packets

3. Wire Shark

A second attempt was made to collect raw packets from the Naval Postgraduate School's network utilizing Wire Shark, the packet sniffer formally known as Ethereal, in this attempt seven days of traffic would be collected and attack traffic would be identified after the fact utilizing Snort and Stealth Watch. One second of network traffic sampled at 2:30 pm on a weekday generated a file two Mega Bytes in size. There was sufficient space on the campus storage area network to store the files until they could be reduced using the SiLK suite. However, Wire Shark was generating temporary files on the collection server and within minutes would consume all free disk space available and crash the service. Limiting packet captures to 68 bytes increased the time to service failure but not enough to make it a viable approach for large amounts of traffic. Using multiple files was also attempted without success. The first file would write correctly and then the service would crash when attempting to write to the second file.

F. CHAPTER SUMMARY

In summary, this chapter described the experiment's data set, how the data set was transformed, the parameters for the experiment runs and ended with a discussion of problems encountered in conducting the experiment.

IV. DATA RESULTS AND EVALUATION

In the last chapter, we described the experiment design. In this chapter, we will describe the results of the series of Random Forest experiments.

A. BASELINE: CALL-ALL-BAD

In all the experiments present in this chapter, the baseline used is Call-all-bad. We use the f-score equation (1.5) rather than harmonic mean equation (1.6) to evaluate our results. We assumed an algorithm that labels all observations as a *bad*. Precision is calculated simply as the proportion of actual *bads* in the dataset. Recall will always be 1.

$$F - Score = \frac{2}{\frac{1}{P} + \frac{1}{R}} \tag{1.5}$$

$$Harmonic\ Mean = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \tag{1.6}$$

The baseline results are conservative since the recall score is 1, which will increase the baseline f-score. The key point is that the recall is not at the expense of precision. If it were, then the f-score being, a special case of the harmonic mean would be lower. To show this, first compute the f-score with p=.6 and r=.6. You can see the answer is .6, identical to the arithmetic mean. However if you adjust your algorithm such that recall is increased to 1 while precision is lowered to .1, the harmonic mean is lower than the arithmetic mean. That is, the arithmetic mean would be .55, while the f-score would be .181.[Martell. 2005]

33

1. 1999 DARPA Week Two Test Baseline

There are 1099 *good* observations and 202 *bad* observations in the data set. This generates a precision of .155 and a recall of 1 resulting in a f-score baseline of .27.

A. NORMALIZED INFORMATION EXPERIMENTS

In this section we present the results of our experiments using normalized information. This section is divided into balanced and unbalanced experiments. The balanced experiments attempt to maximize precision and recall for both *good* and *bad*, while unbalanced experiments try to maximize recall at the expense of precision. The unbalanced experiments were done because for defense purposes, we are far more concerned that all *bad* observations are captured. A result of this weighting is that some *good* observations will be included in the *bad* classification observations.

1. Balanced Experiments

The results of the balanced experiments are given in Table 3. All the experiments are versions of Random Forests with the differences being in the number of trees used.

	Normalized Information Balanced			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.68	0.70	0.69	257%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.66	0.74	0.70	260%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.67	0.71	0.69	257%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.66	0.72	0.69	256%

Table 3   Precision, Recall, and F-Score Results for
Balanced Weighting on the Normalized Information
Data Set

2.   Unbalanced Experiments

The results of the unbalanced experiments are given in
Tables 4-6.   All the experiments are versions of Random
Forests with the differences in the number of trees used.

	Normalized Information *Bad* Weight 10			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.58	0.80	0.67	251%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.58	0.82	0.68	252%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.58	0.82	0.68	252%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.58	0.83	0.68	253%

Table 4 Precision, Recall, and F-Score Results for *Bad*
Weighted 10 on the Normalized Information Data Set

	Normalized Information *Bad* Weight 100			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.42	0.94	0.58	216%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.38	0.96	0.54	201%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.38	0.96	0.55	203%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.38	0.96	0.55	204%

Table 5 Precision, Recall, and F-Score Results for *Bad* Weighted 100 on the Normalized Information Data Set

	Normalized Information *Bad* Weight 1000			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.41	0.96	0.57	213%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.36	0.97	0.52	195%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.34	0.99	0.51	189%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.32	0.99	0.49	181%

Table 6 Precision, Recall, and F-Score Results for *Bad* Weighted 1000 on the Normalized Information Data Set

3. Results

It is interesting that the 1000 tree unbalanced data with *bad* weighted at 10 experiment run was able to increase the recall by .07 while only reducing precision by .08 as compared to the 250 trees balanced data run resulting in a

f-score decrease of only .02. It also was interesting in that there was a clear recall increase by continually weighting *bad* heavier, so that at 500 trees with a *bad* weight of 1000 experiment run, a recall of .99 was achieved, at a of cost of precision dropping to .34 for a f-score of .51. We also note that growing the Random Forest to 1000 trees could hurt the precision in certain runs of the experiment.

B.   SAMPLE ENTROPY

In this section we present the results of our experiments using sample entropy. As before, this section is divided into balanced and unbalanced experiments

1.   Balanced Experiments

The results of the balanced experiments are given in Table 7. All the experiments are versions of Random Forests with the differences being in the number of trees used.

	Sample Entropy Balanced			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.63	0.65	0.64	239%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.65	0.67	0.66	246%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.65	0.67	0.66	247%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.65	0.67	0.66	246%

Table 7   Precision, Recall, and F-Score Results for Balanced Weighting on the Sample Entropy Data Set

2. Unbalanced Experiments

The results of the unbalanced experiments are given in Tables 8-10. All the experiments are versions of Random Forests with the differences being in the number of trees used.

	Sample Entropy *Bad* Weight 10			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.54	0.78	0.64	239%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.54	0.80	0.64	239%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.53	0.82	0.64	239%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.53	0.81	0.64	238%

Table 8 Precision, Recall, and F-Score Results for *Bad*
Weight 10 on the Sample Entropy Data Set

	Sample Entropy *Bad* Weight 100			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.37	0.93	0.53	195%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.34	0.94	0.50	186%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.35	0.96	0.52	192%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.35	0.96	0.51	190%

Table 9 Precision, Recall, and F-Score Results for *Bad*
Weight 100 on the Sample Entropy Data Set

	Sample Entropy *Bad* Weight 1000			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.38	0.94	0.54	202%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.34	0.97	0.50	187%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.32	0.98	0.48	179%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.32	0.98	0.49	181%

Table 10   Precision, Recall, and F-Score Results for
Bad Weight 1000 on the Sample Entropy Data Set

3.   Results

These results were interesting in that growing forests
from a size of 100 to 1000 only increased the recall from
.03 to .05 for a given weight factor.   In this series of
runs, the best achieved was a .98 by weighting *bad* to 1000,
this resulted in a precision of .32 for a f-score of .49.

C.   NORMALIZED INFORMATION AND SAMPLE ENTROPY

In this section we present the results of our
experiments using normalized information and sample
entropy. As before, this section is divided into balanced
and unbalanced experiments.

1.   Balanced Experiments

The results of the balanced experiments are given in
Table 11.   All the experiments are versions of Random
Forests with the differences being in the number of trees
used.

	Normalized Information + Sample Entropy Balanced			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.64	0.73	0.68	255%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.64	0.77	0.70	261%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.64	0.76	0.69	259%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.64	0.78	0.70	260%

Table 11   Precision, Recall, and F-Score Results for
Balanced Weighting on the Normalized Information and
Sample Entropy Data Set

2.   Unbalanced Experiments

The results of the unbalanced experiments are given in Tables 12-14.   All the experiments are versions of Random Forests with the differences being in the number of trees used.

	Normalized Information + Sample Entropy *Bad* Weight 10			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.56	0.89	0.68	254%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.53	0.89	0.67	248%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.54	0.91	0.67	251%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.53	0.91	0.67	249%

Table 12   Precision, Recall, and F-Score Results for
Bad Weight 10 on the Normalized Information and Sample
Entropy Data Set

	Normalized Information + Sample Entropy *Bad* Weight 100			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.42	0.94	0.58	216%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.38	0.98	0.55	205%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.39	0.98	0.56	207%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.39	0.97	0.55	205%

Table 13   Precision, Recall, and F-Score Results for
Bad Weight 100 on the Normalized Information and Sample
Entropy Data Set

	Normalized Information + Sample Entropy *Bad* Weight 1000			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
100 Trees	0.42	0.95	0.59	218%
Call-all-Bad	0.16	1.00	0.27	
250 Trees	0.37	0.97	0.54	199%
Call-all-Bad	0.16	1.00	0.27	
500 Trees	0.33	0.98	0.50	185%
Call-all-Bad	0.16	1.00	0.27	
1000 Trees	0.30	0.99	0.47	173%

Table 14   Precision, Recall, and F-Score Results for
Bad Weight 1000 on the Normalized Information and
Sample Entropy Data Set

41

3. Results

It was interesting that for the balanced weighting there was no statistical gain in growing the forest larger than 250 trees. It was also interesting that with a *bad* weight of 100 a recall of .98 and a precision of .39 resulting in a f-score of .56 was achievable at 500 trees. Recall was able to see a gain of .04 at 500 trees while only causing a reduction in the by .02 compared to the 100 tree f-score.

D. NORMALIZED INFORMATION, SAMPLE ENTROPY AND AVERAGE BYTES PER PACKET

In this section we present the results of our experiments as seen in Table 15 using normalized information, sample entropy and average bytes per packet.

1. 500 Tree Experiments

	Normalized Information + Sample Entropy + Avg Bytes 500 Trees			
	Precision	**Recall**	**F-Score**	**Increase over Baseline**
Call-all-Bad	0.16	1.00	0.27	
Balanced	0.62	0.79	0.69	257%
Call-all-Bad	0.16	1.00	0.27	
Bad Wgt 10	0.52	0.92	0.66	247%
Call-all-Bad	0.16	1.00	0.27	
Bad Wgt 100	0.38	0.97	0.55	204%
Call-all-Bad	0.16	1.00	0.27	
Bad Wgt 1000	0.33	0.99	0.50	185%

Table 15 Precision, Recall, and F-Score Results for 500 Trees with all Weightings on the Normalized Information, Sample Entropy and Average Bytes per Packet Data Set

2. Results

Since the previous three sets of experiments showed very minor gains by going beyond 500 trees, we decided to run this series of experiments at 500 trees. This run of experiments was not expected to show better results than the earlier data sets and was run only because the data was available. It was very interesting that it was possible to achieve a recall of .99 with a *bad* weighting of 1000, precision however, was only .33 resulting in a f-score of .50.

E. EVALUATION OF OVERALL RESULTS

Overall we found it very interesting that the worst f-score result was a .47 from the Normalized Information and Sample Entropy 1000 trees *bad* weight of 1000 run. This result still beat the baseline f-score by 173%. However, more importantly with this f-score, recall was .99, this metric is the focus for intrusion detection.

We also found it puzzling that the Normalized Information metric independently could achieve a higher recall than Sample Entropy. Further work is needed to analyze this result.

A extremely good result was the ability to obtain a recall of .96 with only a Random Forest of 100 trees. This result came from the Normalized Information with a *bad* weight of 1000. This result can be run on a laptop running a 2GHz Pentium4 processor with 384MB of RAM in under 30 seconds. It shows the possibility of conducting near real-time analysis of traffic and locating attack traffic that is getting past a rule-based intrusion detection system.

F.   CHAPTER SUMMARY

In summary, this chapter detailed the results from utilizing the four different combinations of variables varying the number of trees grown and the weight of the bad data. In the next chapter, we will discuss our conclusions and layout possible future work.

V. CONCLUSION AND FUTURE WORK

A.   CONCLUSION

Random Forests were able to classify anomalies in the 1999 DARPA data set. Using only six features from the TCP/IP header data, Random Forests could identify over 99% of five-minute time slices containing attack traffic. This recall comes at a significant reduction in the overall f-score dropping it from .65 to .34. However, this is a worthwhile trade off since in intrusion detection the goal is to ensure all attack traffic is captured.

We conclude that the distribution of packet features (IP addresses and ports) reveals the presence of a wide range of attack traffic. Sample entropy and normalized information are capable automatic classification of anomalies via unsupervised learning.

B.   FUTURE WORK

A goal of this thesis was to determine the effectiveness of Random Forests in classifying anomalies in network traffic. Therefore, future work should include testing Random Forests on additional intrusion detection data sets. There is only the DARPA data set from 1998 and 1999 currently available for scientific researchers to run experiments. It would be interesting to test on the Abilene and Geant Data sets to determine the effectiveness of Random Forests on that data set.[1]

A huge boon to the intrusion detection scientific community would be to develop and make available a labeled data set from the Naval Postgraduate School Network.

---

[1] Computer Science, Boston University 111 Cummington Street, Boston, MA 02215 | Telephone 617 353 8919 | E-mail cs@bu.edu

We also believe it would be interesting to transform the 1999 DARPA data set using the subspace method and Multi-way method combining all vectors into one to allow for a truer comparison to the results of [Lakhina, et al. 2005].

Another idea would be compare the results of Random Forests to the K-means algorithm. Unfortunately in our experiments, the K-means results did not cluster the data set sufficiently to allow for a scientific comparison. The data sets should be run on other implementations of the K-means algorithm to confirm these results.

Finally, we identify issues that could aid in the advancement of intrusion detection research:

1. Develop a more efficient way of automatically consolidating, transforming, and analyzing extracted data. One possible approach would be to combine the various programs written for this thesis into one program which would automatically generate files for Random Forest to classify. Random Forest algorithm as implemented by Salford Systems is capable of running batch jobs. The automatically generated files could be a run by a batch job and labeled as good or bad. This would allow a network analyst to focus on labeled bad traffic.

2. Explore the importance of the predictor variables, and discover if the predictors are constant across the four data sets. Our experiments indicated that the source port was the key predictor of bad traffic and that the destination port was relatively unimportant. Evaluating these variables with principle component analysis could provide further insight into these findings.

3.   Develop a prototype classifier to take sample entropy from near real-time traffic.   This could be designed to work with a hard or sliding five minute window. The time-slices would then be automatically processed and run as a batch job by the Random Forest algorithm, which would label the time-slice as good or bad. The results of the Random Forest would generate alerts for bad traffic.

4.   In parallel to the prototype classifier run a standard rule-based IDS like Snort. The snort alerts could be correlated with alerts from the Random Forest classifier and items with a low correlation would be flagged for human examination.   The low correlation might indicate bad traffic that evaded Snort.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. GENERATED CODE

A.   SAMPLER.JAVA

```java
import java.io.*;

import java.math.*;
import javax.swing.*;
import java.lang.Integer.*;
/**
* Written by Bret Hyla
* extracts 1  in 10 lines from a text file.
* Sept 2006 NPS
 */
public class Sampler {

    public int counter =0;


    public Sampler () {


    }
public static void main(String args[])

  {  Sampler run= new Sampler();
   int i=0;
   int count=0;
            try {
BufferedReader    reader    =    new    BufferedReader(new
FileReader("FullDataSet.txt"));
    try {


BufferedWriter    writer    =    new    BufferedWriter(new
FileWriter("SampledDataSet.txt"));
    while (true){


    String test =reader.readLine();


    if (test==null){


  writer.close();
  break;
                    }
    while( i<10){
```

```
        test =reader.readLine();


    i=i+1;



                }
  if (test==null){
  writer.close();

    break;


                       }
    writer.write(test);


    writer.newLine();


  count++;
  i=0;
          }
    writer.close();


    }  catch(IOException ex) {  //2


        ex.printStackTrace();    //2
        } //2
        reader.close();
    } catch (IOException ex) {  //2


        ex.printStackTrace();    //2
        } //2
    }
}
```

B.   SORTER.PL

```perl
# works with following format on CL sorter.pl followed by
filename
# file is sorted on first element, if tie then second etc
#           base         file          came          from
http://www.developertutorials.com/tutorials/cgi-perl/perl-
sorting-050423/page1.html
# Bret Hyla
# NPS

open(SORT,">bytes_sorted.txt");
my @words=<>;

foreach(sort @words) {
print SORT ;
}
close SORT;
```

C.   REORDER.PL

```perl
# The print SOURCESORTED line can contain any

# of the variables listed in the while loop.
# The order can be modified simply by choosing
# the variable to be put first.
# Bret Hyla September 2006

open(FILE, "allsampleddata.txt");
open(SOURCESORTED, ">bytes.txt");
while (<FILE>) {
     $line=$_;


($sIP, $dIP, $sPort, $dPort, $bytes, $time, $comma)=
(split /,/,$line);
print SOURCESORTED "$time,$bytes \n";

     }

close FILE;
close SOURCESORTED;
```

D.   BYTES.PL

```perl
# This program finds the average number of bytes in

# a 5 minute time slice referred to here as chunk.
# This code is not part of the overall main program as
# we decided after the first data sets were created to
# see if adding the average size of bytes per packet
# would increase the classification rate
# Bret Hyla September 2006

open(FILE, "bytes_sorted.txt");
open(BYTES, ">bytes_in_chunk.txt");

use Date::Calc qw(:all);

# Set a base date to do the timestamp comparison, it #should
be the first timestamp in sorted file from oldest #to
newest.

$yr="1999"; $mon="03";$day= "08"; $hr="13"; $min="00";
$sec="00";

$chunk=0;
$good=0;
$bytes_inchunk=0;

while (<FILE>) {

   $line=$_;
   ($date,$bytes)=(split /,/,$line);
$bytes_inchunk=$bytes_inchunk+$bytes;

$newtime=$date;
$newtime=~ (s/T/:/);
($yr2,$mon2,$day2,$hr2,$min2,$sec2) = (split
/[\/:]/,$newtime);

($D_y,$D_m,$D_d,$Dh,$Dm,$Ds) =
Delta_YMDHMS($yr,$mon,$day,$hr,$min,$sec,
     $yr2,$mon2,$day2, $hr2,$min2,$sec2);


push  @AoA, [ ($chunk, $sIP, $dIP, $sPort, $dPort,$good) ];


#looks at the delta in the two packet time stamps and if
#condition is met creates a new chunk
     if ($Dm>4 or $D_m>0 or $D_d>0 or $Dh>0) {


          $yr=$yr2;            $mon=$mon2;
```

```
        $day=$day2;              $hr=$hr2;
        $min=$min2;              $sec=$sec2;

        $chunk++;
        print BYTES"$chunk, $bytes_inchunk \n";
        $bytes_inchunk=0;
    }

    } #while


close FILE;
```

E.   TEST_TRAIN.PL

```
# This program splits the first file into two files
# These files contain 80% and 20% of the original file.
# Warning, if you have already sorted the data be sure you
# have your classification groups as the primary key.
open(FILE, "Info_normalized_entropy_2ndweekv2_sorted.csv");
open(TEST, ">test.txt");
open(TRAIN,">train.txt");

$chunk=0;
print"$chunk is 0";

while (<FILE>) {
    $line=$_;
    #print" chunk is $chunk\n";
print TRAIN"$line";
$chunk++;

    if ($chunk>4){


  print TEST"$line";
  $chunk=0;
  print" chunk is $chunk\n";
    }
 }
close FILE;

close TEST;
close TRAIN;
```

F.   ENTROPY_INFORMATION.PL

```perl
open(FILE, "sorted.txt");
# several sections of code were based on tutorials found at
#http://perldoc.perl.org/perllol.html

use Date::Calc qw(:all);

#initial a base date to do the timestamp comparison, it
#should be the first timestamp in sorted file from oldest
#to newest.

# Bret Hyla September 2006 NPS

$yr="1999"; $mon="03"; $day = "08"; $hr="13"; $min="00";
$sec="00";

$chunk=0;
$good=0;
while (<FILE>) {
     $line=$_;


($date,$sIP, $dIP, $sPort, $dPort,$comma)=(split /,/,$line);

     $g++;


$sIPhashinfo{$sIP}++;   $dIPhashinfo{$dIP}++;
$sPorthashinfo{$sPort}++; $dPorthashinfo{$dPort}++;

$newtime=$date;
$newtime=~ (s/T/:/);
($yr2,$mon2,$day2,$hr2,$min2,$sec2) = (split
/[\/:]/,$newtime);
($D_y,$D_m,$D_d,$Dh,$Dm,$Ds) =
Delta_YMDHMS($yr,$mon,$day,$hr,$min,$sec,
     $yr2,$mon2,$day2, $hr2,$min2,$sec2);


push  @AoA, [ ($chunk, $sIP, $dIP, $sPort, $dPort,$good) ];


#looks at the delta in the two packet time stamps and if
#condition is met creates new chunk


     if ($Dm>4 or $D_m>0 or $D_d>0 or $Dh>0) {


          $yr=$yr2;              $mon=$mon2;
          $day=$day2;            $hr=$hr2;
```

54

```perl
            $min=$min2;              $sec=$sec2;

            $chunk++;
        }

        } #while


close FILE;


# function to find unique source ips and their prob across
#all data
      foreach $keyinfo (keys %sIPhashinfo) {
   $p++;
   $valueinfo =$sIPhashinfo{$keyinfo};
   $probinfo=$valueinfo/$g;
   $probinfo{$keyinfo}=$valueinfo/$g;


}
      print "num source unique ip keys $p \n";


# function to find unique dest ips and their prob across
#all data
foreach $key2info (keys %dIPhashinfo) {
                          $q++;
                          $value2info
                          =$dIPhashinfo{$key2info};
                          $prob2info{$key2info}=$value2
                          info/$g;
}

      print "num dest ip unique keys $p \n";


# function to find unique source ports and their prob
#across all data
      foreach $key3info (keys %sPorthashinfo) {
$r++;
$value3info =$sPorthashinfo{$key3info};
$prob3info=$value3info/$g;
$prob3info{$key3info}=$value3info/$g;
}
      print "num source port unique keys $r \n";


# function to find unique dest ports and their prob across
```

```perl
# all data
foreach $key4info (keys %dPorthashinfo) {
                               $s++;
                               $value4info
                               =$dPorthashinfo{$key4info};
                               $prob4info=$value4info/$g;
                               $prob4info{$key4info}=$value4
                               info/$g;

}
print "num dest port unique keys $s \n";
print" total lines read is $g";

open (INFO,">info.txt");
open (ENTROPY, ">entropy.txt");
      $prior=0;


      for  $i (0.. $#AoA) {


      if ($prior != $AoA[$i][0]){


#         print "\nNew Prior: $prior\n\n";

         foreach $key (keys %sIPhash) {
            $value =$sIPhash{$key};
            $prob=$value/$t;
#           $sum = $sum +$prob;
            $entropyeach=-1*( $prob* log($prob) );
            $infoeach = -1* log($probinfo{$key});
#           print "    info is $infoeach\n";
            $totalinfo = $totalinfo + $infoeach;
            $totalentropy= $entropyeach +$totalentropy;
         }
   foreach $key2 (keys%dIPhash) {
   $value2 =$dIPhash{$key2};
   $prob2=$value2/$t;
#           $sum2 = $sum2 +$prob2;
            $entropyeach2=-1*( $prob2* log($prob2) );
            $infoeach2 = -1* log($prob2info{$key2} );
            $totalinfo2 = $totalinfo2 + $infoeach2;
            $totalentropy2=$entropyeach2 +$totalentropy2;
         }
      foreach $key3 (keys%sPorthash) {
```

```perl
    $value3 =$sPorthash{$key3};
    $prob3=$value3/$t;
#         $sum3 = $sum3 +$prob3;
          $entropyeach3=-1*($prob3* log($prob3) );
          $infoeach3 = -1* log($prob3info{$key3});
          $totalinfo3 = $totalinfo3 + $infoeach3;
          $totalentropy3=$entropyeach3 +$totalentropy3;
       }
       foreach $key4 (keys%dPorthash) {


    $value4 =$dPorthash{$key4};
    $prob4=$value4/$t;
#         $sum4 = $sum4 +$prob4;
          $entropyeach4=-1*( $prob* log($prob4) );
          $infoeach4 = -1* log($prob4info{$key4});
          $totalinfo4 = $totalinfo + $infoeach4;
          $totalentropy4=$entropyeach4 +$totalentropy4;
       }

print ENTROPY "$prior,$t,
$totalentropy,$totalentropy2,$totalentropy3,$totalentropy4\n
";
print INFO "$prior,$t, $totalinfo,$totalinfo2,$totalinfo3,
$totalinfo4\n";
      $prior= $AoA[$i][0];


#     clearing all variables for the next time slice
$t=0;
undef $sum; undef    $key; undef $prob    ;
undef $entropyeach; undef $infoeach;
undef $totalinfo;    undef $totalentropy;
undef $sum2; undef $key2; undef $prob2;
undef $entropyeach2; undef $infoeach2; undef $totalinfo2;
undef $totalentropy2; undef $sum3; undef $key3;

undef $prob3; undef $entropyeach3; undef  $infoeach3;
undef  $totalinfo3; undef $totalentropy3;
undef $sum4; undef $key4; undef $prob4;
undef $entropyeach4; undef $infoeach4; undef $totalinfo4;
undef $totalentropy4; undef %sIPhash; undef %dIPhash;

undef  %sPorthash; undef %dPorthash;
}    #    if ($prior != $AoA[$i][0])

       $t++;

          $srIP= $AoA[$i][1];
```

```perl
        $sIPhash{$srIP}++;

    #     print " count after t++ $t";
    #     print" $t";

        $dtIP= $AoA[$i][2];
        $dIPhash{$dtIP}++;

        $srPort= $AoA[$i][3];
        $sPorthash{$srPort}++;

        $dtPort= $AoA[$i][4];
        $dPorthash{$dtPort}++;


    }                        #    for  $i (0.. $#AoA) brace


close ENTROPY;
close INFO;
```

LIST OF REFERENCES

ANDERSON, J. 1980. Computer Security Threat Monitoring and Surveillance. 1-56.

ARTHUR, D. AND VASSILVITSKII, S. 2006. How slow is the k-means method? In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry,* Sedona, Arizona, USA, ACM Press, New York, NY, USA, 144-153.

BACE, R.G. 2001. Intrusion detection systems. 43.

BAUER, E. AND KOHAVI, R. 1999. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning* 36, 105-139.

BREIMAN, L. 1996. Bagging Predictors. *Machine Learning* 26, 123-140.

BREIMAN, L. 1998. Using adaptive bagging to debias regressions. 547, Statistics Department, University of California, Berkeley

BREIMAN, L. 2001. Random Forests. Statistics Department, University of California, Berkeley

CANAVAN, J.E. 2001. Fundamentals of network security. 319.

CARUANA, R. AND NICULESCU-MIZIL, A. 2006. An empirical comparison of supervised learning algorithms. In *ICML '06: Proceedings of the 23rd international conference on Machine learning,* Pittsburgh, Pennsylvania, ACM Press, 161-168.

CHAVAN, S., SHAH, K., DAVE, N., MUKHERJEE, S., ABRAHAM, A. AND SANYAL, S. 2004. Adaptive neuro-fuzzy intrusion detection systems*. Proceedings International Conference on Information Technology: Coding and Computing, 2004..ITCC 2004.,* 70-74 Vol.1.

CHEN, C., LIAW, A. AND BREIMAN, L. 2004. Using Random Forest to Learn Imbalanced Data. 666, Statistics Department, University of California, Berkeley

COLOMBE, J.B. AND STEPHENS, G. 2004. Statistical profiling and visualization for detection of malicious insider attacks on computer networks. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security,* Washington DC, USA, ACM Press, , 138-142.

CROTHERS, T., MCSE. 2003. Implementing intrusion detection systems : a hands-on guide for securing the network. 316.

DENNING, D.E. 1987. An intrusion-detection model. 13, 222-232.

DIETTERICH, T. 1998. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. *Machine Learning* 28, 22.

DUAN, Q., HU, C. AND WEI, H. 2005. Enhancing network intrusion detection systems with interval methods. *Proceedings of the 2005 ACM symposium on Applied computing* 1444-1448.

DUDA, R.O. 2001. Pattern classification. 654.

ERBSCHLOE, M. 2005. Trojans, worms, and spyware : a computer security professional's guide to malicious code. 212.

FREUND, Y. AND SCHAPIRE, R. 1996. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 148-156.

HAM, J., YANGCHI CHEN, CRAWFORD, M.M. AND GHOSH, J. 2005. Investigation of the random forest framework for classification of hyperspectral data. *Geoscience and Remote Sensing, IEEE Transactions on* 43, 492-501.

HO, T.K. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 832-844.

HONG HAN, XIN-LIANG LU AND LI-YONG REN. 2002. Using data mining to discover signatures in network-based intrusion detection. *Proceedings, 2002 International Conference on Machine Learning and Cybernetics* , 13-17 vol.1.

JAROSZEWICZ, S. AND SCHEFFER, T. 2005. Fast discovery of unexpected patterns in data, relative to a Bayesian network. *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* 118-127.

JIAN XUE AND YUNXIN ZHAO. 2006. Random Forests-Based Confidence Annotation Using Novel Features from Confusion Network. *Proceedings.2006 IEEE International Conference on Acoustics, Speech and Signal Processing, 2006.ICASSP 2006*, I-1149; I-1152.

JIAN YIN, GANG ZHANG, YI-QUN CHEN AND XIAN-LI FAN. 2004. Multi-events analysis for anomaly intrusion detection. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 1298-1303 vol.2.

JULISCH, K. AND DACIER, M. 2002. Mining intrusion detection alarms for actionable knowledge. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* 366-375.

KHANNA, R. AND LIU, H. 2006. System approach to intrusion detection using hidden Markov model. In *IWCMC '06: Proceeding of the 2006 international conference on Communications and mobile computing,* Vancouver, British Columbia, Canada, ACM Press, , 349-354.

LAKHINA, A., CROVELLA, M. AND DIOT, C. 2004. Characterization of network-wide anomalies in traffic flows. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement,* Taormina, Sicily, Italy, ACM Press, New York, NY, USA, 201-206.

LAKHINA, A., CROVELLA, M. AND DIOT, C. 2005. Mining anomalies using traffic feature distributions. *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications* 217-228.

LANCOPE. 2006. Lancope's StealthWatch. 2006, 12.

LLOYD, S.P. 1982. Least Squares Quantization in PCM. *IEEE Trans. Information Theory* 28, 129-137.

MARTELL, C. 2005. Form: An Experiment in the Annotation of the Kinematics of Gesture. 1-245.

PROCTOR, P.E. 2001. Practical intrusion detection handbook. 359.

REN, D., RAHAL, I., PERRIZO, W. AND SCOTT, K. 2004. A vertical distance-based outlier detection method with local pruning. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management,* Washington, D.C., USA, ACM Press, , 279-284.

SHANNON, C. AND WEAVER, W. 1949. Comprehensive statement. *The Mathematical Theory of Communication*

SOURCEFORGE. 2006. SiLK Analysis Suite. 2006,

YANG, D., HU, C. AND CHEN, Y. 2004. A framework of cooperating intrusion detection based on clustering analysis and expert system. *Proceedings of the 3rd international conference on Information security* 150-154.

ZHAO JUNZHONG AND HUANG HOUKUAN. 2002. An evolving intrusion detection system based on natural immune system. *TENCON '02.Proceedings.2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering* , 129-132 vol.1.

ZISSMAN, M. 2002. 1999 DARPA IDE Week 4 Test Data. 2006,

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
    Fort Belvoir, Virginia

2. Dudley Knox Library
    Naval Postgraduate School
    Monterey, California

3. Marine Corps Representative
    Naval Postgraduate School
    Monterey, California

4. Director, Training and Education, MCCDC, Code C46
    Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC, Code
    C40RC
    Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn:
    Operations Officer)
    Camp Pendleton, California