2015

Use of Entropy for Feature Selection with Intrusion Detection System Parameters

Frank Acker
Nova Southeastern University, afrank@nova.edu

Use of Entropy for Feature Selection with Intrusion Detection System Parameters

by
Frank L. Acker

A dissertation in partial fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in
Computer Information Systems

Graduate School of Computer and Information Sciences
Nova Southeastern University

2015

We hereby certify that this dissertation, submitted by Frank Acker, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

_____ _____

James D. Cannady, Ph.D. Date
Chairperson of Dissertation Committee

_____ _____

Rita M.Barrios, Ph.D. Date
Dissertation Committee Member

_____ _____

Paul Cerkez, Ph.D. Date
Dissertation Committee Member

Approved:

_____ _____

Amon Seagull, Ph.D. Date
Interim Dean, Graduate School of Computer and Information Sciences

Graduate School of Computer and Information Sciences
Nova Southeastern University

2015

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Use of Entropy for Feature Selection with Intrusion Detection System Parameters

By
Frank L. Acker
November 22, 2015

The metric of entropy provides a measure about the randomness of data and a measure of information gained by comparing different attributes. Intrusion detection systems can collect very large amounts of data, which are not necessarily manageable by manual means. Collected intrusion detection data often contains redundant, duplicate, and irrelevant entries, which makes analysis computationally intensive likely leading to unreliable results. Reducing the data to what is relevant and pertinent to the analysis requires the use of data mining techniques and statistics. Identifying patterns in the data is part of analysis for intrusion detections in which the patterns are categorized as normal or anomalous. Anomalous data needs to be further characterized to determine if representative attacks to the network are in progress. Often time subtleties in the data may be too muted to identify certain types of attacks. Many statistics including entropy are used in a number of analysis techniques for identifying attacks, but these analyzes can be improved upon. This research expands the use of Approximate entropy and Sample entropy for feature selection and attack analysis to identify specific types of subtle attacks to network systems. Through enhanced analysis techniques using entropy, the granularity of feature selection and attack identification is improved.

Acknowledgements

This has been quite an adventure. It involved learning a lot about the subject matter and of what I can actually accomplish. It is something I did not think was realistically attainable, but it is here. This would not have been possible without the support of my family. They provided me the quiet time needed to conduct the work. They also worked with me to keep me moving forward. Their review and comments were invaluable.

Dr. Cannady provided the guidance and feedback of the needed expectations required to complete the degree. The committee members, Dr. Barrios and Dr. Cerkez gave invaluable feedback to improve the research presentation and contents.

A special acknowledgement goes to Phillip Porras of SRI International for his assistance in providing access to quality data for analysis and sharing his vast knowledge on the subject of intrusion detection systems. Professional colleagues also constantly encouraged my efforts to complete.

Table of Contents

List of Tables

Tables

Chapter 1

Introduction

Background

Intrusion detection systems help identify malicious and dangerous attacks sent to networks and computers while allowing normal traffic to arrive at its intended destination. In order for intrusion detection systems to identify harmful traffic to computers and networks, packets of data are classified to determine if the contents contain malicious actions or not. Fields of data representing the traffic flow must be collected and analyzed to determine which traffic may pass and which traffic is blocked. The two primary methods used for intrusion detection are signature-based systems and anomaly based systems. A signature based system attempts to match specific patterns in the packets traversing the network for byte strings which are known to be malicious. Anomaly based systems analyze the statistics of the traffic to determine if the packet is malicious.

Data for intrusion detection systems may be collected from multiple sources such as system access logs and activity logs. As these disparate sources merge into a single corpus of data with many records that may provide insight into the collected activity. Each record contains fields that provide information about the activity that the record represents. Some of the fields may contain similar, irrelevant, or missing data, which could potentially cloud the analysis and the overall quality of data. The amount of data collected may also be quite large and impractical to analyze.

For anomaly intrusion detection, fields within the data files are referred to as features. These features describe a particular aspect of information in the record. Since there may be duplicate and irrelevant features contained within the data, using only those features directed at the analysis reduces the computing resources and may improve the accuracy of the resulting analysis. The process of selecting the data, to include only needed features, is termed feature selection. The goal of feature selection is to use only the fields that represent the packet activity while maintaining the integrity of the record and the integrity of entire data set.

There are different methods available to select these pertinent features based on statistics by using one or more algorithms such as used in artificial intelligence, clustering, classification, statistics, and specialized applications targeting specific problems. There are no generic solutions to detect each different type of intrusion or anomalous activity.

Problem Statement

This research addresses the problem of reducing the number of features and correctly identifying relevant features from a set of collected data for an anomaly-based intrusion detection system while maintaining integrity of the data. Data acquired for an intrusion detection system frequently originates from multiple sources such as system activity logs, content of data packets and headers, system calls, memory and disk access activities, and other information. Intrusion detection systems may also share these logs among other network devices for collaboration in a distributed manner. Reducing the amount of data to that which is relevant requires categorizing the information from the logs into parameters, also referred to as dimensions. In a data set of network traffic, attacks are identified by the selection of features that represent particular activities. This implies that not all attacks are found by the same selection of features in all cases. Research conducted by Lima, de Assis, and de Souza (2012) using the KDD CUP 99 (KDD Cup 99 Data, 1999) data resulted in a different set of attributes for each of the four major attack types. Without reducing the number of features, detecting attack patterns within the data is more difficult for rule generation, forecasting, or classification (Gheyas & Smith, 2010). One of the problems is that not all of the features are important (Velayutham & Thangavel, 2012). Identifying and eliminating redundant and irrelevant features within the data, while maintaining the integrity of the corpus, results in features which succinctly describe the activity recorded. Reducing the number of features pertinent to intrusion detection analysis provides better data manageability, lowers computing resource requirements, and usually better results.

Dissertation Goal

The goal of this research is to present a new method that correctly identifies relevant features from an intrusion detection dataset that reduces the amount of data required for anomalous activity detection while maintaining the integrity of the data set. By reducing the redundant features, irrelevant features, and noise, better results may be gained in the analysis of the data for identifying anomalous activities.

The expected results of this research included the following goals:

1. Methods to identify relevant features and minimize the number of features selected from a source of network traffic data without altering the characteristics of the data representation.
2. Compare results of correctly classified and incorrectly classified as percentages, and the features selected with those published by Sharma and Mukjherjee (2012), and Lima, et al., (2012) for the KDD CUP 99 data (KDD Cup 99 Data, 1999).
3. Using real-world data from the SRI Cyber-Threat Analysis Project, apply the methods used in goals 1 and 2 to compare and contrast the results with a second set of data for correctly classification of attacks and the features selected from the analysis.

Relevance and Significance

This research focuses on methods that select features from a set of intrusion detection system data in an efficient manner while maintaining the integrity of the data to represent the traffic and events collected. Many approaches have addressed the problem of feature selection. Even with the successes, a significant amount of work is still needed to find improved methods of feature selection from intrusion detection system data. Tavallaee, Bagheri, Lu, and Ghorbani (2009) state that current approaches to intrusion detection are not a mature technology. This problem is still relvant as identified by the research of Zuech, Khoshgoftaar, and Wald (2015). Improving detection and feature selection are important to provide better analysis results for anomaly detection in identifying attacks on network systems.

Data sources from intrusion detection systems provide a large quantity of data for analysis. Since most of the raw intrusion detection data sets contain duplicate and irrelevant features, the selection of significant and relevant features is important. The feature selection process attempts to discard superfluous data and noise, which in turn reduces the overall volume of the data set while maintaining its integrity. This reduced data set, in turn, yields to a faster and more accurate analysis. In order to carry out this data reduction effort, classification applications analyze the data and identify appropriate categories. In addition to the classification, elimination of redundant features from the data is necessary. Without doing so makes patterns more difficult to detect (Gheyas & Smith, 2010).

Barriers and Issues

The problem presented is an on-going issue for selecting features within a data set that accurately represents the activity of the collected data. Often these files are large since they are generated from disparate sources. The quantity of data must be reduced and categorized into a set of events called attributes (Lima et al., 2012). Within the large files, the data must be normalized and attributes that best represent the activity must be present, while duplicate and non-essential information is eliminated. This may result in improved performance and outcome. Having clean and usable data provides the analytic applications with a higher probability of obtaining usable results.

This goal of efficient feature selection is not always met. Even though there may be a large volume of anomalous data, not all attacks may appear within the data. In addition, there may not be a sufficient number of events present to identify the anomaly as an attack or identify it correctly. Properly identifying the features to use is a problem since different attacks may need different attributes for the correct identification.

The research conducted by Lima, et al. (2012) used the C4.5 decision tree model based on entropy and compared these results with three other attribute selection methods. They conducted their evaluation was using the KDD CUP 99 (KDD Cup 99 Data, 1999) data set.

Tavallaee et al. (2009) described the different attack categories in the KDD CUP 99 (KDD Cup 99 Data, 1999) data in the following list.

- **Denial of Service Attack (DOS):** denies legitimate users access to a system by consuming computing and memory resources.

- **User to Root (U2R):** An attacker gains legitimate access to a system and exploits a vulnerability to escalate their privileges to root access.
- **Remote to Local (R2L):** A user who does not have an account for legitimate access to a system, gains remote access to it through exploiting a vulnerability by sending packets over a network.
- **Probing Attack:** Gathering information about a network and its computers to circumvent its security.

The analysis by Lima et al. (2012) used three different entropy approaches, and each approach produced a different set of attributes for identifying the type of attack group. Even though some of the features selected were the same, there was overlap in the parameter selection, the results were different. Their work showed that varying approaches affects results.

In conducting this research with a feature selection algorithm using entropy as a factor in the classification and selection process, evaluating which entropy calculation best fits a specific attack, attack type, or a generalized application for all attacks were among some of the challenges for consideration. The Shannon entropy is the most established measure of uncertainty and mutual information (Alvim, Andrés & Palamidessi, 2010). Other entropy methods, such as the Rényi entropy and the Tsallis entropy, shared some of the properties with Shannon's approach (Harremoës, 2006). Lima et al. (2012) used the Rényi entropy and the Tsallis entropy as additional entropy measures in their research.

Alazab, Hobbs, Abawajy, and Alazab (2012) identified attack patterns within the attack types. Their focus on the U2R attack type in which they identified four new attack patterns: httptunnel, ps, sqlattack, and xterm. The following table detailed their categorization of the attacks and attack patterns.

Attack Type	Attack Pattern
Probe	Ipsweep, nmap, portsweep, satan, mscan, saint
DoS	back, land, neptune, pod, smurf, teardrop, apache2, mailbomb, processtable, udpstorm
U2R	Buffer_overflow, loadmodule, perl, rootkit, httptunnel, ps, sqlattack, xterm
R2L	ftp_write, guess_password, imap, multihop, phf, spy, warezclient, warezmaster, xlook, xsnoop, snmpguess, worm

Other research data sources, such as those referenced in research performed by Nguyen, Franke, and Petrović (2012), used the ECLM/PKDD 2007 data and the CSIC 2010 data. Both of these data sets were tested using data from web application firewalls. The ECLM/PKDD 2007 data was from the 18th European Conference on Machine Learning and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (Gallagher, & Eliassi-Rad, 2008). Another set of data used in research was by the Spanish National Research Council that developed the CSIC data set. This data provided a set of http transactions from an e-commerce site. The purpose was to test the protection of web applications (HTTP DATASET CSIC, 2010).

Even though the KDD CUP 99 data set was not an ideal source, according to Tavallaee et al. (2009), variants include the NSL-KDD (NSL-KDD Data Set, 2009) data

set. This alleviated and reduced some of the problems with the original KDD CUP 99 dataset identified by Tavallaee et al. (2009).

To improve the data available for off-line intrusion detection system research, Vasudevan, Harshini, and Selvakumar (2011) evaluated the KDD CUP 99 data set and identified a number of shortcomings. They developed their own set of intrusion detection system data to represent current network activities. Some of the weaknesses of the KDD CUP 99 data set identified included:

- Many of the attacks used in the data set were fixed and do not exist anymore.
- Attack sophistication increased while knowledge needed to launch an attack has decreased.
- The attacks were in a naive form and do not represent network behavior.
- All attacks were preplanned and mixed between host and network.

Guillén, Rodriguez, Páez, and Rodriguez (2012) also supported the concept of the KDD CUP 99 data set being outdated. However, they qualified this statement by indicating that the results were reliable for analysis purposes, and the data was usable to analyze new intrusion detection approaches for machine learning or computational intelligence. Their research included using a DARPA data set and a software package named Spleen along with the KDD CUP 99 data. Even though the KDD CUP 99 data contained shortfalls as noted, it was still considered satisfactory for use with the analysis proposed for this research.

The availability of publically available labeled data sets for intrusion detection research was limited. The KDD CUP 99 data set was the most recognizable data store

publically available for intrusion detection research. Another set of data that was of potential use was named PREDICT was supported by the US Department of Homeland Security, Science & Technology Directorate. Users of PREDICT must be vetted and agreements signed as to the nature of its usage and disclosure. PREDICT data was not labeled, and therefore not satisfactory for this research.

Other possible data sources for use in the research included the CAIDA (n.d.) and SRI (n.d.) data repositories. These sources contained various types of data from internet traces. The possibility also existed that data from SRI International located in Menlo Park, California had merit since it contained timing data and attack information to provide a labeled data set (SRI, n.d.). In reviewing the data sources for a second analysis using different data, it was decided that the data from SRI would be the best choice.

Another area of difficulty was the integration of new calculation algorithms into the existing applications chosen for classification and feature selection analysis. Depending on the openness and complexity of the applications, incorporating custom entropy algorithms into the structure of the programs may be difficult. This challenge was overcome by the use of tutorials and papers that described modification of analysis applications for customized calculations.

Assumptions, Limitations, and Delimitations

Since intrusion detection systems data often contained a collection of logs from multiple sources, analyzing and interpreting the data was a challenge. This research made two primary assumptions about the data. First was the quality of the data, in that each record provides an accurate representation of the information contained within the complete packet. The second assumption addressed the problem of consistency in the meaning and relationship of the data across the different fields within the record. Since one of the data sets was the KDD CUP 99, this collection of data was used in many analyzed research projects. Although deficiencies were noted in the Barriers and Issues section of this thesis, the KDD CUP 99 is widely accepted as a standard data store for this type of analysis. The second set of data for this research originated from the SRI System Design Laboratory (SRI, n.d.). Other sources considered were the PREDICT, and the CAIDA data repository (CAIDA - The Cooperative Association for Internet Data Analysis, n.d.). None of these sources were as thoroughly tested and researched as the KDD CUP 99 data.

Because the fields within each record may be an aggregation of data from more than one source, the meanings of similar fields from each source may not be the same. This causes inaccuracies in the calculations of results along with a potential bias of the data, which impacts the results. There was no control of the representation of the data as it was presented in the initial stages of the research. As the research effort progressed, adjustments were made as needed to normalize the individual data fields for more accurate representation of their intended meanings.

An additional data source which has labels and time markings was collected at SRI by their Cyber-Threat Analysis Project. An arrangement was made between the researcher and provider for use of this data for the analysis (Personal communications with Phillip Porras of SRI, May 25, 2014).

The researcher, to establish boundaries for this research, imposed delimiting factors. Since Approximate entropy and Sample entropy are time based, some data, which does not have timings associated with the records, as in the KDD CUP 99 data set, were simulated. No research was located that assigned timings to the entries in the KDD CUP 99 data store. Fares, Sharawy, and Zayed (2011) identified timing in their research, where they described the taxonomy of intrusion detection but never applied it in the analysis.

The manner of simulating periodicity within the KDD CUP 99 was established for this research. Simulation consisted of applying different windowing sizes and statistics to the data. One example used the order of the data as provided and assigned windowing intervals based upon recommendations of Yentes et al. (2013). Another method was to vary the windowing intervals. In addition, the ordering was assigned to the data analyzed at the time. The ordering and windowing in the KDD CUP 99 data was needed and used to calculate the Approximate entropy and Sample entropy. This research addressed the issue of windowing with a selection of data from the KDD CUP 99 data set. Ordering of the records by attack type represent the timing in which they occurred. The results may or may not show that ordering was highly critical in the election of attributes for the analysis. Experiments during the research indicated how the windowing and parametric variation impacted the analysis.

The second data set was from SRI. It was a series of data files including a number of them in pcap format. Pcap stands for packet capture and contains network traffic information. A number of Unix/Linux based utilities were available that deconstructed the contents of pcap files. The files from SRI contained timing points and labels incorporated within the collection.

Summary

This research provided additional viewpoints for the use of entropy in feature selection. The number of features available in a set of data collected from intrusion detection systems may be quite large and unmanageable for manual human manipulation and for analysis by computer applications. By reducing the number of features in the data set, the goal was to make it more manageable for analysis and enhance the accuracy of the results.

This section also identified some of the challenges that made this research difficult. One of the more challenging and difficult problems was the availability of valid labeled data which was satisfactory for use in this context. The KDD CUP 99 data set was the most widely used and accepted for intrusion detection research purposes. Other sources were primarily accessed from non-public sources, which may place restrictions on its use. SRI granted permission for this researcher to use data from its Cyber-Threat Analysis Project for this research (Personal email Communications with SRI Researcher, August 26, 2014).

Chapter 2

Review of Literature

This literature review is a study of research work discussing the development of intrusion detection methods and current advances in the field and focuses on the methods of feature selection using intrusion detection data. Within the feature selection process, the metric of entropy is used in different aspects of data analysis. Subsections in this chapter are:

- Background of Intrusion Detection Systems
- Data Mining and Feature Selection Methods
- Entropy Calculations used in feature Selection
- Shannon entropy
- Rényi and Tsallis entropy
- Approximate entropy and Sample entropy
- Methods and applications used in this research

Background of Intrusion Detection Systems

Anderson (1980) introduced the concept of auditing and surveillance as a way to improve the security of a customer's computer systems. The research focused on the use of security audit trails as an important role in detecting unauthorized access to a data set or system. He also defined the concepts of threat, risk, vulnerability, attack, and penetration. The types of intrusions identified were an internal penetration from within the system or an attacker from outside the system via communications lines. Also

included was the application of statistics to collected data in order to identify abnormal use of the systems. For systems with a large number of users, it was necessary to reduce the volume of the data. One of the methods proposed by Anderson was through sampling data on a periodic basis. These techniques proposed some of the first methods to monitor the security of computer systems.

Denning (1986) expanded the concept of intrusion detection and developed a model for a real-time intrusion detection system. Denning proposed using a real-time collection of audit records from attempted break-ins, system penetrations, and abuses through the use of system monitoring. Abnormal use information was categorized into bundles, called tuples, and models were applied to the data. Denning's analysis detected a wide range of intrusions. Some of the detected intrusions identified were without knowledge of system vulnerabilities.

As the complexity of systems grew, the quantity of collected data increased to the point where manual processing was impractical and automation was needed. Development of automated systems that merged data from multiple sources provided a vast array of different aspects of system activity. Collections, such as these, result in many dimensions, including possible duplicates, irrelevant features, and general noise. Identifying anomalous behavior from bloated data sets produced bad results and taxed computational resources (Lima et al., 2012).

As data was collected from the system and the network device logs, it was analyzed, and the results used to protect the systems by developing information, which analyzed the traffic and determined if an attack might be taking place. Determining if an attack took place was the result of analyzed system and network device logs. Through

this effort, different techniques to identify intrusions were developed. The two main approaches to categorize intrusion detection systems were misuse detection, and anomaly detection (Sharma, & Mukherjee, 2012). A misuse-based system examined the packets looking for patterns and signatures of known attacks on the network. Anomaly based systems used statistical analysis to compare features of the traffic with a profile of what normal traffic flow should look like. A majority of the commercial intrusion detection systems used today implemented misuse-based detection because of its high accuracy (Tavallaee et al., 2009). An example of a misuse-based detection program is Snort (n.d.). However, the academic community considered anomaly based detection a more powerful method due to its potential to detect novel attacks (Tavallaee et al., 2009).

Research conducted by Gupta, Nath, and Kotagiri (2010) developed a layered approach to intrusion detection in which their work used layers in series to identify anomalous activity. Each layer detected one of the four groups of intrusions included in the KDD CUP 99 data set. A feature selection process was run for each of the four intrusion types with the results having a different set of attributes identified. Improved accuracy and performance was evident with this model.

When an intrusion detection system identifies an attack through misuse detection or anomaly detection, the action taken may be passive or reactive. A passive intrusion detection system logs information when it detects a potential security breach. The reactive intrusion detection system takes action when it detects suspicious behavior such as discontinuing service to the user or alerting a firewall to block traffic from a particular source (Sharma, & Mukherjee, 2012).

In both the misuse and anomaly-based intrusion detection systems, the data sources provided a large quantity of data for analysis. Since raw intrusion detection data sets often contained duplicate and irrelevant features, elimination of this superfluous data and noise reduced the volume, which, in turn, yielded a better analysis (Hammer & Villmann, 2002).

Sharma and Mukherjee (2012) focused on the detection of minority attacks since current standalone intrusion detection systems were not effective in finding these types of attacks. Within the KDD CUP 99 data set, there are four major attack types: DoS, Probe, R2L, and U2R. The work of Sharma and Mukherjee examined the attributes that detected the R2l and U2R attacks. Sharma and Mukherjee based their work on a layered approach by Gupta et al. (2010).

Anomaly detection included the problem of identifying patterns and behaviors that do not conform to the normal traffic data (Bhuyan, Bhattacharyya, & Kalita, 2011). Successfully identifying these anomalies had a higher probability when the intruder did not know what a legitimate user's activity should look like and what was considered anomalous. They refer to work by Kumar and Spafford (1994) regarding the four possibilities in detecting an intruder who has no knowledge of the system activity profile as denoted in the following list.

- Intrusive but not anomalous: a false negative since the activity was intrusive but not detected or identified as anomalous.
- Not intrusive but anomalous: a false positive since the legitimate user was conducting a non-malicious activity; however, identified as anomalous.

- Not intrusive and not anomalous: a true negative since the activity was not intrusive or anomalous.
- Intrusive and anomalous: a true positive since the activity was intrusive and identified as anomalous.

In the above list of an intruder's activities as defined by Kumar and Spafford (1994), the detection of anomalous behavior differed based on the metrics and approaches. The conclusion by Bhuyan et al. (2011) was that some anomaly detection methods were better than other methods and more work was needed to focus on lowering false alarm rates.

Data Mining and Feature Selection Methods

Methods for identifying pertinent features that represent the data include classification algorithms, genetic algorithms, statistics, and decision trees. These methods, developed over the years, usually focused on specific types of problems, such as those tuned for attacks, which are rare or minor (Sharma, & Mukherjee, 2012). Even though significant academic research and applied implementations focused on intrusion detection, these systems still had trouble detecting intrusive activities since new and novel attacks were constantly evolving (Sharma, & Mukherjee, 2012).

Feature selection algorithms used supervised learning when labeled data sets were available for training. Unsupervised learning used non-labeled data sets. With labeled data sets, the features could distinguish different classifications. Selection of the proper method for analyzing data was important, as each data set had its own statistical

properties. Using these methods along with classifiers and entropy combinations resulted in improved granularity for feature selection.

Lima et al. (2012) referred to the reduced number of features as attributes. These features contained information describing a particular aspect of the activity recorded. They accomplish this feature reduction through compressing the collected data with methods and applications used in biological research. The feature reduction processes conducted by Lima, et al. (2012) used applications that performed clustering, classification, and feature selection functions.

Research conducted by Yentes, Hunt, Schmid, Kaipust, McGrath, and Stergiou (2013) investigated the use of Approximate entropy and Sample entropy for the measurement of data in a time series. Their data source originated from physiological characteristics between young and older adults, such as their gait. These entropy calculations included use in a number of other biological research environments including heart rate and other biomedical data (Pan, Wang, Liang & Lee, 2011).

Yentes et al. (2013) used Approximate entropy and Sample entropy calculations in their feature selection research with biological data. Each of these different forms of entropy calculations provided additional views of the information extracted from the available data.

One of the data sources for this research included the KDD CUP 99 data (KDD CUP Data, 1999) set which contained approximately 5 million records of normal and attack traffic. Another possible data source was from the Cooperative Association for Internet Data Analysis (CAIDA, n.d.) data made available in association with PREDICT Repository (PREDICT - Protected Repository for the Defense of Infrastructure Against

Cyber Threats, n.d.) supported by the Department of Homeland Security, Science and Technology Directorate. In addition, data from SRI International's System Design Laboratory investigated intrusion-detection research since 1983 (SRI, n.d.).

Tavallaee et al. (2009) evaluated the KDD CUP 99 (KDD Cup 99 Data, 1999) data set and identified details of its attributes and shortcomings. They resolved a number of the issues that resulted in a new data set designated as NSL-KDD (NSL-KDD data set, 2009). Their new data set, NSL-KDD, has the following advantages:

1. No redundancy in the training data thereby reducing the bias towards those records.
2. No duplication in the test data thereby reducing the bias towards more frequent detection of those duplicates.
3. Better mix of the levels of difficulty resulting in classification learning rates with a more accurate evaluation of different learning techniques.
4. Record count in the training sets and test data set allowed learning and evaluation applications to use the complete range of data without random selection.

The three main characteristics of intrusion detection systems were accuracy, extensibility, and adaptability (Om & Kundu, 2012). They proposed a hybrid intrusion detection system that utilized incremental learning to detect future attacks. The goal for their method was to have a high detection rate and a low false positive rate. To profile the network, Om and Kundu (2012) used K-means clustering and K-Nearest Neighbor algorithms. Om and Kundu (2012) also used entropy as a feature based statistical method to select attributes and eliminate redundant attributes. Their process first removed

irrelevant features then calculated mutual information between features and the classification.

The next step was to cluster the data into similar types of objects without using classification labels, or unsupervised learning. This unsupervised learning approach created groups with different attributes, and the greater the differences occurring among the groups actually improved the clustering. When classifying data using an unsupervised learning approach, three different methods were used including Naïve Bayes, decision tree, and support vector machine.

A Naïve Bayes classifier algorithm computed the probability of the classes given the data, which assumed independence among the features for each class (Dougherty, Kohavi, & Sahami, 1995) implemented in the WEKA analysis package. WEKA is an acronym for Waikato Environment for Knowledge Analysis (WEKA 3, n.d.). In decision tree methods, continuous values were binned during the learning process and a dependency map was structured.

A support vector machine classifier automatically searched vectors with classification ability to maximize the margin between the classes. It had excellent generalization and high classification accuracy. The standard support vector machine algorithm calculated the vectors by solving a quadratic programming problem, whose time complexity was exponential. Thus, for large-scale training sets, the computation of the standard support vector machine was not practical (Songfeng, Xiaofeng, Nanning, & Weipu, 2003).

Nguyen et al. (2012) researched the use of pattern recognition for intrusion detection systems through the application of steadiness and consistency metrics to judge

the classifier's performance. Generic Feature Selection is one of the feature selection methods discussed by Nguyen et al. A steadiness metric, in the feature selection process, quantified and measured the parametric of a specific classifier's performance. The other metric implemented by Nguyen et al. was the consistency of the analysis that evaluated the feature selection process for a specific classifier. When a variable used in the calculation, α was equal to 1, the search strategy was said to be consistent.

Om and Kundu (2012) used the KDD CUP 99 data to train and test their model. They applied 10-fold cross validation to calculate classification accuracy using detection rate, false positive rate, classification rate, along with the true positive, true negative, and false negative. Their methodology was a three step process. The first step applied a feature selection algorithm, which used entropy as one of the statistics. The next step clustered the data with unlabeled data using K-means clustering and classification methods. The final step was a hybrid classification that assigned classification labels to objects. This was accomplished by using one of the following algorithms: K-Nearest Neighbor, Naïve Bayes, decision tree, or support vector machines. Om and Kundu (2012) concluded that with their hybrid approach and algorithms, they could detect differences between normal and anomalous data.

Lee, Gray, and Kim (2013) discussed the problem of high-dimensional data as being commonplace due to advanced sensing systems and storage technologies. These massively high-dimensional data sets introduced sparsity, redundancy, and computational complexity into the analysis. High-dimensional data usually had a limited number of degrees of freedom, which was the intrinsic dimensionality of the data (Lee et al., 2013).

Reducing the dimensionality of the data set reduced some of the problems relating to redundancy and computational complexity.

Research conducted by Zhai, Li, and Zhai (2011) reduced the computing resource requirements through the use of sample fuzzy entropy along with a condensed K-Nearest Neighbor rule method. This calculation used decision table, fuzzy entropy, and an algorithm to determine the fuzzy membership degree of instances in the training data set (Zhai et al., 2011). Their research developed two algorithms, which determined the fuzzy membership degree in the training data set. A third algorithm implemented the Condensed Fuzzy K-Nearest Neighbor (CFKNN) rule based on sample fuzzy entropy. Results showed their method reduced the complexity for K-Nearest Neighbor computations using fuzzy entropy. The authors recommend the use of the third algorithm, CFKNN, which they claim resulted in a feasible and effective solution.

Decision trees represented acquired knowledge. The strategy for decision trees implemented non-incremental learning from examples (Quinlan, 1986). Quinlan's research also provided a description of induction trees. This work led to the ID3 application, which evolved into C4.5, used by Lima et al. (2012). Quinlan (1986) discussed the concept of Top Down Induction of Decision Trees in which the classification was conducted from the top down by considering the frequency of occurrences within the data. Through the induction task, the set of objects were a collection of attributes where each object belonged to one of a set of mutually exclusive classes. The objects in the set of training data had a known class. The mission was to develop a classification rule that could determine the class from the attributes of any object. A subset of the training data was selected and used to train the classifier in an

iterative manner. The remaining entries in the training data were classified against the tree. If there were classification errors, these errors were added to the subset of training data and the tree were developed again. This process repeated until all the classification of the training data was correct. Used in tree generation, the algorithms calculated the information gained in an object of data through the use of entropy.

Alazab et al. (2012) defined classification as a learning function for categorizing unseen data into predefined classes. This implied that the data had the records labeled according to their classification. When working with cluster algorithms, the data was unlabeled. While in clustering, the classes were not predefined. Alazab et al. stated that further research into feature selection based intrusion detection was needed.

Even though a classifier completed its goals, the question arose as to whether the outcome of a classifier could be trusted (Nguyen et al., 2012). The feature selection process consisted of the method and search strategies for relevant features. Each dataset had its own statistical properties, where the feature selection process best represented the patterns of the data (Nguyen et al., 2012).

Bhuyan et al. (2011) further described an intrusion detection architecture. In this design, data collection, pre-processing, feature extraction, data typing, normalization, and an anomaly detection engine were functions of the system, which identified irrelevant parameters for anomaly detection. The detected anomalies were classified into three categories based on the following list:

- **Point anomalies**: An individual data point was anomalous with respect to the rest of the data.

- **Contextual anomalies**: These anomalies consisted of two types: contextual and behavioral. The contextual content was with respect to its relation to a certain set of attributes. The behavioral is with respect to non-contextual attributes.
- **Collective anomalies**: A single point was not anomalous but a collection of single points constituted anomalous activity. In order to detect this type of anomaly, the appropriate behavioral attributes in the data needed identification.

Feature selection involved maximizing classification accuracy of data. Multiple approaches were available for feature selection with the two main feature selection models being the wrapper model and filter model. The wrapper model used a learning algorithm on subsets of the features and the resulting feature set quality was determined by the prediction accuracy (Gheyas & Smith, 2010). In the filter model, statistical criteria generated scores and ranks for the features. This model determined the relevance of features through statistical techniques that were independent of any classifier.

Alelyani, Tang, and Liu (2013) differentiated feature selection and feature extraction as approaches to reducing the dimensionality of a data set. In feature extraction, features were projected into a new space with lower dimensionality, while feature selection took a subset of features that minimized redundancy while maximizing their relevance. Alelyani et al. (2013) expanded the feature selection models to include an embedded model and a hybrid model. In their proposed hybrid model, statistical measures were used like the filter model, and a subset of the data was chosen with the highest classification accuracy. This embedded model implemented feature selection and model fitting simultaneously where they selected a set of features based upon a particular

classifier. Overall, the filter model worked well with large data sets while the wrapper model improved classification accuracy.

Liu and Yu (2005) described a typical feature selection process that consisted of four steps.

1. Generate a subset of features from a set of data. This selection of features may be additive in that the null set was the basis and features were added, or it may be a subtractive process by starting with all features and removing them in a predetermined manner. A complete exhaustive search found optimal results. Other options were a sequential search and a random search.
2. Evaluated the subset based on a set of criterion. Different criteria evaluations techniques included distance measures, dependency measures, and consistency measures.
3. Determine if the resulting goals were met. This may be a specific boundary of features, a better solution was not produced from a previous result, or the results were satisfactory based on the classification error rate.
4. If the goals were met, results were validated and the process terminated. If prior knowledge was available, the results could be compared. Often prior knowledge was not available and other techniques were employed. These may be classification error rates, or conducting "before-and-after" experiments.

Liu & Yu (2005) further discuss the filter and wrapper methods and a hybrid combination of the two. Their discussion included real world applications with feature

selection and network security. They proposed using data mining algorithms for large audit data files to obtain frequency patterns. The patterns were used in automated learning and classifiers were applied to determine an intrusion or normal traffic.

Research conducted by Barot, Chauhan, and Patel (2014) used the KDD CUP 99 data set and applied different feature selection methods including a Naïve Bayes classifier, decision table, correlation based feature selection, and Chi-squared attribute selection. Their results showed that using five attributes produced very good performance. Using a correlation based feature algorithm along with the decision table majority produced the best results.

Entropy calculations used in Feature Selection

Entropy was defined as a statistical metric that related the amount of information into a random variable (Lima et al., 2012). Using this definition for entropy, parameters used for the identification of an intrusion from activity logs contained randomness within the data, which provided information about that data, to the analytic algorithms used. Nychis, Sekar, Andersen, Kim, and Zhang (2008) stated that little research has been conducted to understand the detection power of entropy-based analysis related to multiple traffic distributions.

Lima et al. (2012) used the Shannon entropy that was included in the WEKA, toolkit (Witten & Frank, 2005). Lima et al. then replaced the Shannon entropy with the Rényi and the Tsallis entropy formulas and compared the impact of the different entropy

calculations on feature selection functionality. The Rényi and Tsallis entropy calculations included an α term which adjusted the sensitivity to the probability distribution.

In a paper on Boltzmann's entropy, Lebowitz (1993) discusses how Boltzmann used entropy to describe associating different states of matter between microscopic and macroscopic in a statistical manner. The results were in terms of classical Newtonian mechanics based on Newton's laws of motion. The Boltzmann entropy was equal to the Boltzmann constant times the log of the absolute phase state (Γ) for a state of M. The point showed that entropy extended beyond not only information theory as proposed by Shannon (1948) but also had roots in mechanical and quantum systems.

Lee and He (2009) used entropy with the Chi-square goodness metrics and mean and variance to develop traffic profiles and behavior patterns. The concept of relative uncertainty created a data profile that used time series to find hidden features in the traffic. They used the KDD CUP 99 data set and developed a correlation matrix using different features against the true positive, true negative, false positive, and false negative measures. Their research reduced the false positives by 3 to 4 percent.

Barbará, Couto, and Li (2002) proposed a method that clustered the data to reduce the entropy rather than using a distance metric. Their approach yielded an NP-Complete problem that used heuristics to solve it. They applied this methodology to different types of data, including the KDD CUP 99 data set. The algorithm was effective and compared well to other algorithmic methods that used Shannon entropy.

Research conducted by Nychis et al. (2008) utilized entropy to analyze bidirectional traffic with the goal of improving granularity of detection from simple volume based metrics. The basis for the data was flow-headers and behavioral features.

The parametric data collected from the flow-header consisted of source and destination IP addresses and ports, and the flow size. The behavioral attributes were counts of specific addresses where an end-host communicated when entering and exiting the system. The data used in the analysis originated from collections made in 2005 at Carnegie Mellon University. The data consisted of 92TB of traffic with 2.5 billion flows. The data segments contained five-minute non-overlapping time slots, and anonymized IP addresses. The entropy for the parameters was normalized and computed. The researchers found strong correlation between address and port distributions. The results showed that with entropy based anomaly detection, traffic selection required more than simple port and address based distributions. Traffic features should originate from traffic distributions that complement each other. Also, unidirectional traffic could introduce bias into the computing traffic distributions.

Nychis et al. (2008) concluded that port and address distributions were strongly correlated when using entropy during time series analysis. They confirmed this with the behavioral metrics and from the analysis of synthetic data. Calculating correlations of entropy values during normal periods suggested a new way to provide anomaly detection services and they suggested this for future work.

Velayutham and Thangavel (2012) used entropy for feature selection with Rough Set Theory. In their work, both supervised and unsupervised sets of data showed how their process produced better results with the unsupervised data. One of their claims stated supervised data classification was often unknown or incomplete. In their demonstration example, the unsupervised data was grouped by like attributes and the entropy was calculated among their values. The minimal entropy was selected and

grouped with the remaining attribute sets. This continued iteratively until an entropy of zero resulted. The attributes in this set were the reduced group of features that provided information about the data set.

Yurtkan and Demirel (2013) used entropy based feature selection for facial recognition. The use of variance and entropy provided measures of uncertainty and information content. A high entropy indicated a feature's position was more variable and carried more information. A low entropy was considered a stable feature. Their research used Shannon entropy for feature selection in facial expressions. The higher the entropy value the greater the chance was that the feature was associated with different expressions.

Özçelik and Brooks (2015) discussed the use of entropy in identifying Distributed Denial of Service (DDoS) attacks in a network. If the attacker had knowledge of the network traffic entropy, the attacker could spoof the use of entropy to evade the identification of a DDoS attacks. With this information, an attack could be constructed to maintain the entropy of the traffic within the upper and lower bounds of the entropy range considered. Features used were in the packet headers. Similarly, the attacker could construct zombies to send dummy traffic/requests that generate false positives, which rendered the intrusion detection system unreliable. To counter this spoofing capability, the calculated standard deviation for the traffic was normalized in two limits by asymptotically increasing the entropy less than 1 to approaching 1, and conversely normalizing entropy larger than 0 to approaching 0. This method enabled the identification of spoofing attacks.

Shannon Entropy

Claude Shannon (1948) worked at Bell Laboratories and developed a useful definition of information produced. The definition stated that if the number of messages in a set was finite, then this number was a measure of information when one message was chosen from the set. This definition provided the basis for Forward Error Correction and communications security (Gappmair, 1999). Shannon's research in entropy and channel capacity became part of the common mechanisms used to monitor and evaluate communications systems. Shannon's application of entropy to information theory was the basis for describing variability in a signal.

Mathematical formulations of entropy in feature selection were as follows. Applying feature selection techniques to data sets using a random variable, C, with a discrete probability distribution, then the entropy of the expected information was determined by the Shannon (1948) entropy defined in the equation below.

Using this basic formula for Shannon entropy, there are multiple attributes (k), where i = 1, .. k.

$$H(C) = \sum_{i=1}^{k} p_i \log_2 p_i$$

Where:

H(C) is the entropy of variable C

p_i is the probability of element *i* in the distribution.

k is the number of elements.

Mutual information $I(C;A_i)$ measures the interdependence between two features, i.e. C and A_i, is shown as when using Shannon entropy (Lima et al., 2012).

$$I(C;A_i) = H(C) - H(C|A_i)$$

Where:

$I(C;A_i)$ is the mutual information denoting the dependence between C and A_i.

H(C) is the entropy of variable C

$H(C|A_i)$ is the conditional entropy of C given A_i.

Rényi & Tsallis Entropy

Both the Rényi and Tsallis entropy use a term in their equations identified as α. This term makes the entropy results more or less sensitive to the considered probability distribution shapes Lima et al. (2012).

The Rényi entropy is a measure of information of order α. For Rényi entropy, Shannon entropy is the limiting case Lima et al. (2012). The formula for Rényi entropy is as follows.

$$R_\alpha(C) = \frac{1}{(1-\alpha)} \times \log \sum_{i=1}^{k} p_i^\alpha, \alpha \geq 0, \alpha \neq 1$$

Where:

$R_\alpha(C)$ is Rényi entropy with factor alpha for term C

α is an exponential distribution where $0 \leq \alpha \neq 1$

p_i^{α} is the probability of element i in the distribution raised to the α term.

With $0 > \alpha < 1$, the mutual information is defined as follows (Lima et al., 2012).

$$I_R(C;A_i) = R_\alpha(C) - R_\alpha(C|A_i)$$

Where:

$I_R(C;A_i)$ is the mutual information of C given A using Rényi entropy

$R_\alpha(C)$ is Rényi entropy with factor alpha for term C

$R_\alpha(C|A_i)$ is Rényi entropy with factor alpha for term C given A_i

Constantino Tsallis, a Brazilian physicist, developed an entropy relationship integrated within the Boltzmann-Gibbs domain that defined entropy as follows (Johal & Tirnakli, 2004).

$$T_\alpha(C) = \frac{1}{(\alpha - 1)} \times \sum_{i=1} p_i^{\alpha}$$

Where:

$T_\alpha(C)$ is Tsallis entropy with factor alpha for term C

α is an exponential distribution where $0 \leq \alpha \neq 1$

p_i^{α} is the probability of element i in the distribution raised to the α term.

The mutual information as noted by $I_T(C;A)$ is for Tsallis entropy when $\alpha > 1$, the dependencies between two variables are defined as follows (Lima et al., 2012).

$$I_T(C; A_i) = T_\alpha(C) - T_\alpha(C|A_i)$$

Where:

$I_T(C;A_i)$ is the mutual information of C given A_i using Tsallis entropy

$T_\alpha(C)$ is Tsallis entropy with factor alpha for term C

$T_\alpha(C|A_i)$ is Tsallis entropy with factor alpha for term C given A_i

Approximate Entropy and Sample Entropy

Approximate entropy is the conditional probability of a set of data segments of the same duration. There is less complexity with a smaller Approximate entropy, which yields a higher probability. Its introduction quantified regularity in a time series (Liu & Zhao, 2011).

Pincus (1991) developed a method to determine the changing system complexity in which Approximate entropy could classify complex systems. The use of Approximate entropy was applicable to deterministic (predictable), and stochastic (non-deterministic) systems. This approximation was good for data sets containing at least 1000 points.

Approximate entropy is a widely used statistical index that quantifies the complexity of a signal used, especially in the fields of heat variability and endocrinology (Chen, Solomon, & Chon, 2005). This metric may provide quantitative information about noisy and short data in a small sample size. The data may have both deterministic and stochastic (non-random and random) attributes. Some of the problems with Approximate

entropy includes bias due to self-matches, or duplicates, and is very dependent on sample size.

Yentes et al. (2013) applied Approximate entropy as developed by Pincus (1991) for quantifying levels of complexity in time series. Sample entropy, developed by Richmond and Moorman (2000), was less sensitive to the number of data points than Approximate entropy and provided a better entropy method for data sets with less than 200 points. Approximate entropy does have some problems in its use. It is biased towards regularity, lacks relative consistency, and parameters must be the same when comparing two data sets.

The smaller the value of the Approximate entropy indicated less complexity within the data. This suggested that repeated patterns imply order and therefore resulted in a reduced entropy value (Lake, 2011). The calculation required a prior determination of two unknown parameters. The variable named ***r*** had a recommended value in the range of 0.1 to 0.2 times the standard deviation of the data. The other variable named ***m*** determined the length of the sequences, or window sizes. A third parameter used in the entropy equation is ***N*** which is the number of data points (Chon, Scully, & Lu, 2009).

Most entropy definitions were discontinuous to noise (Pincus 1991). Approximate entropy used three primary attributes in the calculation. The nomenclature was represented by *ApEn(m, r, N)* for Approximate entropy. Selection of the attributes affected results of the calculation. The **m** referred to the window size of how many points represented a reading. Pincus (1991) started with **m** equal to 2 as does Yentes et al. (2013). The **r** is a measure of the percentage of the standard deviation.

Sample entropy is a negative natural logarithm of the conditional probability that two samples of length ***m*** with tolerance ***r*** would match the next point in the series of ***m+1***. If ***m*** was too large or ***r*** was too small, the template match count would be inadequate for confidence estimation of the conditional probability. Conversely, if the ***m*** was too small and ***r*** was too large, all results matched and there would be no discrimination signals (Lake, 2011).

Liu, Liu, Shao, Li, Sun, Wang, and Liu (2011) determined the selection of the ***r*** variable was controversial. They referred to studies that indicated that as the performance of a time series became faster, the selection of ***r*** might lead to incorrect conclusions. Their work was based on heart failure rate among healthy subjects vs. those that had heart failure. They concluded the value of ***r*** had a big impact on the results and proposed the use of a value that maximized the Approximate entropy. This showed the true complexity of the different signals more clearly.

Six steps used to calculate Approximate entropy were described by Pincus and Keefe (1992) and are detailed below.

1. Develop an equally spaced time series:

   where u(1), u(2), ….. u(N) where N is the number of values

2. Define ***m*** and ***r***.

   m = length of the time sequence (windows) use 1, 2, 3, etc. and

   r = filter - usually between 10% to 25% of the standard deviation.

3. Define a set of vectors: x(1), x(2), x(3) ..... x(N)

   where: x(i) = (u(1). . . . . u(i + m – 1))

4. Use x(1), x(2), x(3) ..... x(N)

   for each i, $1 \leq I \leq N - m - 1$

   $C_i^m(r)$ = { number of **x**(j) such that d**[x**(i), **x**(j)**]** $\leq r$ } / (N + m -1)

   Where:

   d is the distance between vectors x(i) and x(j). It is defined as:

   d[**x**(i), **x**(j)] = max | u(i + k -1) - u(j + k -1) |

   for k = 1, 2, ... m.

5. Next define:

$$\Phi^m = (N-m+1)^{-1} + \sum_{i=1}^{N-m+1} \ln C_i^m(r)$$

   To this point, Approximate entropy yielded that

   $\Phi^{m+1}(r) - \Phi^m(r)$ = the average over i of

   ln [probability that | u(j+m) – u(i+m) | $\leq$ r

   given that

   | u (j+k) – u (i+k) | $\leq$ r for k = 0,1,....m-1]

6. (ApEn) Approximate entropy equation:

$$ApEn = \Phi^{m}(r) - \Phi^{m+1}(r) \quad \text{for } \mathbf{m} \text{ and } \mathbf{r} \text{ fixed as in step 2.}$$

Work by Manis (2008) developed a way to increase the speed of approximate calculations. In this method, data is assigned buckets, the buckets are examined for similarity of data pairs, and updates made to the overall calculation.

Sinai (2007) explained the entropy of dynamical systems and stated that entropy (h) of a measurable transformation of the dynamical system was in a set of entropy values for the entropy across the upper bounds of all finite partitions. Sinai (2007) further stated that Kolmogorov proved this theorem in a lecture on Bernoulli partitions where entropy must be positive.

Richman and Moorman (2000) developed Sample entropy, which was a variant of the Approximate entropy. Sample entropy does not count self-matches and is the negative natural logarithm of the conditional probability that two sequences for **m** points remain similar at the next point. Self-matches were not included in the probability calculation.

Approximate entropy quantifies information about complex data that may be noisy and corrupted in both deterministic and stochastic environments (Chen et al., 2005). With both Approximate entropy and its variant Sample entropy, the equations use two variables that must be predefined. One variable is the embedding dimension, **m**. The second variable is the threshold that acted as a noise filter with the designation of **r**. Chen et al. (2005) referred to a recommendation by Pincus (1991) for slow dynamic signals in which **r** should be 0.1 to 0.26 (10% to 26%) of the standard deviation of the data. They also recommended that **m** should be 1 or 2 for 100 to 5,000 data points. Chen

et al. (2005) detailed the calculation for Approximate entropy and Sample entropy in their research. Both calculations were six step processes not described in this thesis.

Methods and applications used in this research.

Lima et al. (2012) used different entropy calculations in their research. The most used calculation for entropy of computer communications work was that of Shannon (1948). Lima et al. (2012) extended the Shannon entropy calculation in the C4.5 classification algorithm to include Rényi and Tsallis variations of entropy. This compared the feature selection ability of the Rényi and Tsallis entropy calculations versus the Shannon method.

Very little research existed that addressed the use of Approximate entropy and Sample entropy for use with intrusion detection data. The focus of Approximate entropy and Sample entropy calculations was data that exhibited periodicity. The proper data must align with the Approximate and Sample entropy models used.

Sharma and Mukherjee (2012) utilized a Naïve Bayes classifier in WEKA that reduced the dimensionality of intrusion detection system data sets. The Naïve Bayes classifier worked well with high dimensionality data sets and had a strong independence relation assumption in which the features were independent of a class and the probability of one attribute did not influence the probability of the other. They used the entropy-based supervised discretization. This process transformed continuous models into discrete parts for analysis. In particular, the WEKA application calculates a result, iteratively removes a feature, and the results are compared for effectiveness.

The main classification algorithms used for feature selection were genetic algorithms, decision trees, Bayes networks, and neural networks. Lima, et al. (2012) used classification models implemented for medical data that included CLONal selection ALGorithm (CLONALG), Clonal Selection Classification Algorithm (CRCA), and Artificial Immune Recognition Systems (AIRS). The attribute selection method used by Lima et al. (2012) was C4.5. They modified the entropy calculations to include Rényi entropy and Tsallis entropy, in addition to the Shannon entropy calculations available in C4.5.

The data mining capabilities of the WEKA software used by Lima et al. (2012) provided an extensible environment to modify and insert custom calculations for the analysis. Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten (2009) discussed this flexibility in their paper on WEKA. Multiple forms of data entry were available, including comma separated variables as is contained in the KDD CUP 99 data set. The WEKA application was Java based and provisions were available to add custom software. The WEKA open-source project specifically focused on open-source data mining systems.

The research conducted by Lima et al. (2012) incorporated the use of the wrapper model into the C4.5 application for their model. They surmised that in general, the wrapper method was more effective in selecting the best features.

Chapter 3

Methodology

Overview of Research Methodology

This research incorporated the use of entropy in the statistical methods for feature selection to detect network intrusions. The goal was to reduce the number of features required to identify anomalies in a set of data from an intrusion detection system. Using Approximate and Sample entropy as metrics in the feature selection process was part of achieving that goal. The applicability of this method was adapted to the detection of different intrusion types that exhibited periodicity or modeled with periodicity. The focus was on the use of the entropy statistic to provide additional information regarding the content and variability of data.

The approach used was based on the use of entropy for feature selection as that conducted by Lima et al. (2012). The Lima et al. research utilized the C4.5 decision tree modeled with the Shannon, Rényi, and Tsallis entropy calculations as part of the statistics for attribute selections. Research conducted by Yentes et al. (2013) used Approximate entropy and Sample entropy to measure the randomness of periodic biomechanical data such as a person's walking gait. This research includes the Approximate entropy and Sample entropy within the C4.5 decision tree.

Specific Research Methods Employed

The Waikato Environment for Knowledge Analysis (WEKA 3, n.d.) software framework was used for the C4.5 decision tree generation that is designated as J48 in the WEKA classification analysis package. The 10-fold cross validation option validated the results. Programming modifications made to WEKA enabled the use of Rényi, Tsallis, Approximate, and Sample entropy calculations and combinations.

Data for the research originated from two different sources. The KDD CUP 99 data used by Lima et al. (2012) was one source. The second set of data was from SRI International in which the data was collected from real-world malware attacks. The use of these two different data sources supported a process for validating the methods used in identifying anomalies from the KDD CUP 99 data set. The second set of data used the same process in finding anomalous activity.

Following the finalization of the basic methods and techniques mentioned above, the applications for classification and feature selection were developed and programs written. The next set of activities identified the code development required for Rényi, Tsallis, Approximate, and Sample entropy calculations and statistical algorithms. Development also included writing programs that acquired, parsed, and formatted the data for use in the analysis programs and its associated results.

Analyzing the results from the decision trees and extracting the features provided data to compare metrics from established research by Lima et al. (2012). The metrics used for the comparisons examined the classification values and the features selected. This started an iterative process of working with both sets of data available, producing results, and comparing them with the selected standards. Adjustment made to the

methods or calculations to improve the performance and robustness of the results continued until no significant improvement was gained by more changes. As the research completed, new literature was reviewed for comparison of the techniques and methods to validate the conclusions. No additional research was identified that utilized similar methodologies.

Instrument Development and Validation

The primary application used during the analysis was the WEKA program (WEKA 3, n.d.). This application contained the tools required for the analysis. Modifications made to the WEKA application implemented the Rényi, Tsallis, Approximate, and Sample entropy statistic modules developed for this research. The general description of the Approximate entropy and Sample entropy algorithms were set forth in the paper by Pincus and Keefe (1992). The paper by Hall et al. (2009) provided a description of the WEKA program along with its history, accomplishments, and capabilities.

A web site called "The Code Project" contained the C++ code for both the Approximate entropy and Sample entropy algorithms that was posted by Chesnokov (2008). This downloaded code was validated for correctness by comparing the process described by Pincus and Keefe (1992). Modifications were made to the Chesnokov (2008) code to translate it into Java code. The Approximate entropy and Sample entropy algorithms were integrated into the WEKA J48 tree classification package for feature selection. A paper by Bouckaert, Frank, Hall, Holmes, Pfahringer, Reutemann, and Witten (2010) described the process of custom code integration into WEKA.

The research results presented an objective description of the outcomes with tables, graphics, and text along with a discussion of the methods used. Also included were references to research literature supporting or refuting the findings.

Resource Requirements

This research utilized the expertise of researchers, computer systems, applications, and data. Collaboration occurred via email with committee members, other researchers, and peers as needed. These researchers were considered knowledgeable and experts in intrusion detection mechanisms, data analysis, and statistics.

Windows and Linux based computer systems were used for data retrieval, storage, development, preparation, and processing. The primary computer used was an HP laptop running the Windows 8 operating system (Win8, n.d.). Linux was supported via a virtual machine using Oracle VM VirtualBox (VB, n.d.) on the HP laptop. VirtualBox allowed the instantiation of virtual machines to run on a system. The Windows 8 system ran VirtualBox to support the Ubuntu operating system (Ubuntu, n.d.). The version of Ubuntu used in VirtualBox was 14.04.1.

The Eclipse (n.d.) Integrated Development Environment was used for the Java applications and integration into WEKA. Additional development tools were available in a Linux environment including vi, javac, etc.

Two main data sources were used. One was the KDDCUP 99 data file, considered one of the standard data sets used for Intrusion Detection research (KDD Cup 1999 Data, 1999). The second data source originated from the SRI International Cyber-Threat

Analysis Project (SRI, n.d.) and this is the first time it was used for feature election in this manner.

Summary

This section described the methodologies used to conduct the research and include Approximate, and Sample entropy into the feature selection process. The goal was to determine if the Approximate and Sample entropies generated better results than the Shannon, Rényi, and Tsallis entropies used by Lima et al. (2012), and the Shannon entropy used by Sharma and Mukjherjee (2012). The data acquired for this research was KDDCUP 99 data, and data from the SRI Cyber-Threat Analysis project as the second source selected. The WEKA application, using the J48 decision tree analysis with the 10-fold cross validation option, was selected to conduct the analysis. The output from J48 provided a decision tree analysis for the selected features and classification statistics. The output values were extracted and compared with the selected features from Lima et al. (2012) and the work of Sharma and Mukjherjee (2012).

The modification and development of software was needed for this research. The open source WEKA application provided the primary package for generation of the decision tree analysis. The features identified were part of the decision tree output. The WEKA application was modified to include the Rényi, and Tsallis entropies that Lima et al. (2012) used, and included the Approximate and Sample entropies. Each of the added entropy calculations were made available for the analysis along with the Shannon entropy that was included in WEKA. Work by Chesnokov (2008) was the basis for the source code used in the Approximate and Sample entropies calculations. The code was then

modified for use in WEKA. The approach used for modifying WEKA and including new source code were identified by Bouckaert et al. (2010).

Chapter 4

Results

Introduction

Chapter 4 focuses on the processes used for this research, the results achieved, and the research accomplishments during the data acquisition, data preparation, and analysis. This work developed new methods for labeling activity in intrusion detection system data resulting in multiple views of different entropy calculations in the feature selection process. These views provide different options for the selection of relevant features from data sets that identify anomalous traffic from intrusion detection system. The analysis methodology used multiple entropy statistics developed for the C4.5 classification tree algorithm.

The primary analytic tool used for the analysis was the open source WEKA application written in Java by the Machine Learning Group at the University of Waikato in New Zealand. Being open source, all source code and binaries were available for downloaded and modification. Java and Linux shell scripts were the languages used to develop additional software applications for this research.

Anomalous and malware data acquired for this research originated from two sources. One was the KDD CUP 99 data used by many intrusion detection researchers for validating new intrusion detection processes and statistical evaluations. The second data source consisted of real-world data collected by the Computer Science Laboratory at SRI in Menlo Park, California in cooperation with its director, Phillip Porras. This SRI data

originated from malware alerts collected by the BotHunter (BotHunter, n.d.) application that identified malware in network traffic.

Computing System Environment

The computing environment for this research included both Microsoft Windows and Linux based operating systems that ran on an HP Envy laptop. The system's primary hardware consisted of an Intel I7 processor, 12 GB of memory, a 1 TB hard disk drive, and a 17-inch screen. Microsoft Windows 8.1 was the operating system on the laptop. The standard Microsoft Office applications suite was included along with the Oracle VM VirtualBox virtualization product.

Oracle VM VirtualBox version 4.3.12 is a type 2 hypervisor for virtual machine support to host the Linux kernel version 3.13.0-35-generic with the Ubuntu operating system version 14.04.1. This system configuration supported the concurrent use of a Microsoft Windows and Linux environment while also enabling the sharing of files between the two operating system applications and their utilities.

Directory structure for the file systems consisted of two types: the standard Microsoft Windows hierarchical structure, and the Linux hierarchical structure. A share point established within the MS Windows file system and the Linux file system provided a common point to mount file systems. Any files written below this share point in the file directory structure were accessible by both operating systems.

As identified in the introduction of this chapter, the WEKA source code and binaries were available for download from the WEKA website (WEKA 3, n.d.). In addition, the files were available for different operating systems. This research

downloaded and implemented the Linux version of WEKA as the main development and execution platform. WEKA Version 3.4.19 was acquired as a zip file, which was the same version used by Lima et al. (2012). The expanded zip package consisted of Java and binary code that installed into default directories.

Code Modification and Development

Eclipse version 3.8 provided the Integrated Development Environment to modify the WEKA Java source code and develop new Java classes. The WEKA download consisted of Java source code files, documentation, and the **build.xml** support file for use within Eclipse.

Several WEKA Java methods required modification in order to implement the new entropy calculations into the source code. Calls to the entropy calculations from the WEKA application were for the Java methods in the **EntropyBasedSplitCrit.java** file. The **EntropyBasedSplitCrit** class contained methods named **logFunc(), oldEnt(),** and **newEnt().** The J48 classification algorithms used these methods in the calculation.

In the downloaded WEKA code, only the Shannon entropy calculation was included in the source code. Modifications made to the **EntropyBasedSplitCrit** class added the Rényi, Tsallis, Approximate, and Sample entropy calculations. Selecting the entropy calculation to use was a run time configurable option defined in an external file that set the values for the current WEKA analysis. The J48 classification tree module read the configuration file at run-time. Parametric values, identified in the external file, determined the type of entropy algorithm to use and the values for the corresponding variables in the program.

In the **EntropyBasedSplitCrit** class, a method created for this analysis, named **varInitialize(),** defined the variables used with the five different entropy calculations. This method initiated a program call that read the configuration file then set the corresponding parameters. The Rényi and Tsallis entropy calculations were added to the existing Java code in the **EntropyBasedSplitCrit** class by modifying the **logFunc(), oldEnt(),** and **newEnt()** methods. Appendix A lists the modified **EntropyBasedSplitCrit** class Java code.

Four additional classes were developed. One was the **fileRead** class that read and parsed the configuration file. The second class, **entUtils**, provided utilities for use by the entropy application, which included a method to calculate standard deviation, **stdev()**, for data passed to it by methods in the classes that calculated Approximate and Sample entropies. In addition, a method added to the **entUtils** class handled the reading of the configuration file and printing its parameters. This method, called the **fileRead()** method from the **fileRead** class, used the results to print attributes and set variables for the subsequent calculations. Appendix D lists the Java code for the **fileRead** class and Appendix E lists the **entUtils** class Java code.

Configuration File

The configuration file, named **EntropyInfoFile.txt**, was read when the J48 tree classification module instantiated the **EntropyBasedSplitCrit** class. Within the configuration file, values used by different entropy calculations were initialized in WEKA. Depending on the entropy calculation used, only those variables required for the calculations were relevant, variables not needed were ignored.

Appendix F provides an example of a configuration file. Variables included in the external configuration file are listed in Table 1.

Name	Description
etype	A numeric designation of the entropy calculation to use. 0 = Shannon 1 = Rényi 2 = Tsallis 3 = Approximate 4 = Sample
alpha	The numerical value for the alpha term in the Rényi and Tsallis entropy calculations that denotes the sensitivity to the considered probability distribution shapes. Values proposed by Lima et al. (2012) are 0.5 for Rényi and 1.2 for Tsallis, however, these values may be set to a value suitable for the calculation
m	This is the window size for the Approximate and Sample entropy calculations. Research by Yentes et al. (2013) proposed a value of 2, but may be reset in this configuration file.
r	This is the amount of the variance to be used in the calculation of Approximate and Sample entropy. Research by Yentes et al. (2013) proposed a value of 0.2, but may be reset in this configuration file.
D	This is a Boolean variable used to turn on debugging during the development and modification of the application. It has no impact on the computations.

Table 1 Entropy Configuration File Variables

Algorithms for Rényi and Tsallis Entropy Calculations

The Literature Review chapter presented details of the Rényi and Tsallis entropy calculations. These equations were programmed in Java for this research and added to WEKA as **logFunc(), oldEnt(),** and **newEnt()** methods in the **EntropyBasedSplitCrit** class. The logic included a series of if statements based upon the **etype** parameter that sets the type of entropy based on the configuration file.

Algorithms for Approximate and Sample Entropy Calculations

New classes for the Approximate and Sample entropy calculations were developed and identified as **ApproximateEntropy**, and **SampleEntropy**. These new classes were downloaded in the C++ language from Chesnokov (2008), converted to Java code, and further modified for use within this research. The methods in these classes were programmatically called from the **logFunc(), oldEnt(),** and **newEnt()** methods of the **EntropyBasedSplitCrit** class. Appendix B and Appendix C list the modified **ApproximateEntropy** and **SampleEntropy** Java classes respectively.

Eclipse Usage

The Eclipse (n.d.) package provided an Integrated Development Environment. Its development began at IBM and then the Eclipse Foundation sponsored its support. Eclipse, an open source application, enabled the development of programming projects in different computer languages, including Java. For this research, Eclipse supported Java code development, modifications, compilations, and installation of the **weka.jar** file. The **build.xml** file, provided in the initial WEKA download, was updated for the purposes of

this research in order to compile and install the **weka.jar** file where needed. Appendix G lists the modified **build.xml** file.

Executing WEKA

The WEKA application ran in an Ubuntu environment within a virtual machine. The command line below started the WEKA GUI and included the required class paths.

```
java -cp /media/sf_nova/workspace/weka/dist/weka.jar:\
/media/sf_nova/data/KDD/wekaclassalgos/wekaclassalgos.jar \
-Xmx8192m weka.gui.GUIChooser
```

In the shell script, the memory allocation increased from the default of 512 MB to 8192 MB to accommodate large data files. The back slash "\" at the end of the line indicated a continuation of the command line. The WEKA source code version 3.4.19 was used to be consistent with Lima et al. (2012) work.

The J48 classification tree execution used the ten-fold validation option for the analysis. Saved results determined the features selected to construct the classification tree. Appendix L and Appendix M list the shell scripts that read the files containing the J48 classification tree results. These shell scripts extracted the features used to construct the classification tree from the KDD CUP 99 results and the SRI results respectively. Appendix N displays the output for the DOS category of the KDD CUP 99 data using Shannon entropy.

KDD CUP 99 Data Acquisition and Preparation

This data set contained approximately five million records of attacks and normal traffic. The file, downloaded from the KDD CUP 99 website (KDD Cup 1999 Data, 1999), was in a comma separated value (csv) format that enabled easy manipulation using Linux commands. Native WEKA data is in the Attribute-Relation File Format using the extension **arff**. The data set contained 42 columns that describe each entry in the file. Appendix J lists the features and their type corresponding to the data designation as represented in the **arff** file.

Research conducted by Lima et al. (2012) used a subset of the KDD CUP 99 data for their analysis. In order to replicate the work by Lima et al., similar attack and normal traffic counts were replicated as close as possible, to accurately reproduce their results. The specific lines used from the KDD CUP 99 were unknown. Table 2 displays a tally of the KDD CUP 99 data available by attack type, and labeled "**Available**". The column labeled "**Count**" identified the number of entries of available data used during the Lima et al. analysis and used in this research. A discrepancy identified in the multihop, phf, spy, and loadmodule attack counts used by Lima et al. indicated more data than supplied within the KDD CUP 99 dataset. For those instances where the "% of Total" was greater than 100%, the maximum attack counts of entries were used even when the count was less than what Lima et al. (2012) used in their research paper as noted in the "**Comment**" column.

Using the attack counts listed in Table 3 by Lima et al. (2012) and in this research, the files generated used a series of Linux commands pipelined together. This table showed the total line counts for each of the files using the required WEKA format, **arff**.

The command line below provides an example for the selection of the "**back**" attack from the DoS category, which originated from a file named **kdd.data.csv**. This Linux command string was repeated for each attack and appended to the proper attack file.

```
$ cat kdd.data.csv | grep back | shuf –n 1026 >>DoS.csv
```

Appendix H explains the commands used above. Appendix I lists the complete Linux shell script used to generate the different data files that reproduced Lima et al. (2012) results. The shell script wrote data to files used in WEKA. Additional information entered into the **arff** files defined the variable names and data types contained in the file. Since no timing information was associated with the KDD CUP 99 data, having the same attacks grouped together modeled the periodicity for the Approximate and Sample entropy. Appendix K provides a partial listing of the file contents for DoS.arff.

Category Attack	Available	Count	% of Total	Comment
DoS				
back.	2,203	1,026	46.57%	
land.	21	11	52.38%	
neptune.	1,072,017	10,401	0.97%	
pod.	264	69	26.14%	
smurf.	2,807,886	7,669	0.27%	
teardrop	979	15	1.53%	
Normal	972,781	2,573	0.26%	
Probe				
Ipsweep	12,481	586	4.70%	
Nmap	2,316	151	6.52%	
Pportsweep.	10,413	155	1.49%	
Ssatan.	15,892	16	0.10%	
Normal.	972,781	1,704	0.18%	
R2L				
ftp_write	8	5	62.50%	
guess_passwd	53	53	100.00%	
imap	12	11	91.67%	
multihop.	7	11	157.14%	Used 7
phf.	4	5	125.00%	Used 4
spy.	2	4	200.00%	Used 2
warezclient.	1,020	60	5.88%	
warezmaster.	20	20	100.00%	
Normal	972,781	1934	0.20%	
U2R				
loadmodule.	9	10	111.1%	Used 9
buffer_overflow	30	21	70.00%	
perl.	3	3	100.0%	
rootkit.	10	7	70.00%	
Normal.	972,781	1,676	0.17%	

Table 2- Attack Counts (Lima et al., 2012)

Category/ File name	Lines in arff file
DoS.arff	21,813
Probe.arff	2,661
R2L.arff	2,145
U2R.arff	1,765

Table 3 - Line counts in KDD CUP 99 .arff data files

KDD CUP 99 Analysis Results

This section describes the analytic results of this research by replicating the approach use by Lima et al. (2012), as closely as possible. The following descriptions and tables demonstrated the results were in close agreement with Lima et al. classifications. This agreement does not extend to the number of features selected. The feature selected and their counts vary significantly between Lima et al. work and this research.

The WEKA application used files generated for the DoS, Probe, R2L and U2R categories as input. Appendix K lists a portion of the **arff** file for the DoS attack category. Appendix N displays a sample of the J48 classification output for the DoS attack category. Appendix O shows results of the shell script execution that extracted the features from Appendix N.

Table 4 presents the results of this research using the KDD CUP 99 data with the WEKA analysis for the different entropy calculations. Note that for the Rényi entropy, the alpha value was 0.5, and for the Tsallis entropy, the alpha value was 1.2, as recommended by Lima et al. (2012). When specifying Approximate and Sample entropy, the window size, **m**, was "2" and the **r** value was "0.2" for both entropy calculations as specified by Yentes et al. (2013). The definitions below describe each column listed in the tables.

- **Attack** & **Entropy** = Attack type and entropy used in the calculation
- **Source** = Origin of calculation results:

  Research indicates the work conducted in this research.

  Lima described information from Lima et al. (2012).

  Sharma described information from Sharma and Mukherjee (2012).

- **CC** = Correctly classified attacks.
- **ICC** = Incorrectly classified attacks.
- **Features Selected** = the features published from Lima et al. (2012), Sharma and Mukherjee (2012), or the WEKA J48 classification analysis.
- **Qty** = Number of features identified.

Attack Entropy	Source	CC	ICC	Features Selected	Qty
DoS					
Rényi					
	Lima	99.9632%	0.0368%	**2**, **5, 7, 8**, **23**, 32, 35, 36, 39	9
	Research	99.4578%	0.5422%	4, **6, 7, 8**, 12, 13, **23**, 25, 27, 29, 32, 34, 37, 40	14
Shannon					
	Lima	99.9495%	0.0505%	**2**, **5, 7, 8**, **23**, 34, 36, 39	8
	Research	99.9541%	0.0459%	**2**, 3, **5, 7, 8**, 25, 29	7
	Sharma	99.9000%	0.1000%	**5,6**,24	3
Tsallis					
	Lima	99.9586%	0.0414%	**2**, **5, 7, 8**, **23**, 26, 34, 39	8
	Research	99.9357%	0.0643%	**2**, 3, 4, **5, 6, 7, 8**, 10, **23**, 24, 25, 26, 29, 31, 36, 37	16
ApEn					
	Research	99.8989%	0.1011%	**2**, 3, 4, **5, 6, 7**, 13, **23**, 24, 37	10
SampEn					
	Research	99.9081%	0.0919%	**2**, 3, 4, **5, 6, 7**, 13, **23**, 24, 36, 37	11
Probe					
Rényi					
	Lima	99.4266%	0.5734%	**1, 2, 5, 6**, 25, **30, 32, 33, 37**, 38, **40**	11
	Research	96.4778%	3.5222%	**2, 3, 4, 5, 6**, 23, 24, 25, 26, 27, 29, **30**, 31, **32, 33**, 35, **37**	17
Shannon					
	Lima	99.5031%	0.4969%	**1, 2, 4, 5, 6**, 23, **30, 33, 37**, 38, **40**	11
	Research	99.0046%	0.9954%	**3, 5, 6**, 12, 23, 25, 27, **32**, 34, 36, **37, 40**, 41	13
	Sharma	98.8000%	1.2000%	**1,5,6**,**30,33**	5
Tsallis					
	Lima	99.3119%	0.6881%	**1, 2, 4, 6**, 23, **30, 31, 33, 37**, 38, **40**	11
	Research	99.1577%	0.8423%	**2, 3, 5**, 25, 29, 34, 35, 36, **37,** 39, **40**, 41	12
ApEn					
	Research	98.4303%	1.5697%	**1, 2, 3, 4**, 5, 31, **32**, 34, 36, **37**	10
SampEn					
	Research	98.4303%	1.5697%	**1, 2, 3, 4, 5**, 31, **32,** 34, 36, **37**	10

Note: Table 4 continued on next page

Attack Entropy	Source	CC	ICC	Features Selected	Qty
R2L					
Rényi					
	Lima	98.9534%	1.4066%	2, **5, 6, 10, 11**, 12, **19**, 33, **35**, 37, 38, 39	12
	Research	95.6107%	4.3893%	**1**, 4, **10**, 13, 19, 22, 25, 34, 35, 36, 38	11
Shannon					
	Lima	98.9058%	1.0942%	**1, 3, 5, 6,** 9, **10, 11**, **17**, **19**, 22, 32, 33, **35**	13
	Research	98.4733%	1.5267%	**1, 3, 5, 6, 10, 11**, 12, 14, **17**, 33, **36**, 38, 39	13
	Sharma	97.0000%	3.0000%	**1, 3, 5, 6**, 23, 24, 30, 31, 32, **36**	10
Tsallis					
	Lima	98.8582%	1.1418%	**1,3, 5, 6, 10, 11, 17**, **19**, 22, 37, 38	11
	Research	98.1393%	1.8607%	**1,** 4, **5, 10, 11**, 13, **17, 19**, 22, 26, **36**, 39, 41	13
ApEn					
	Research	97.9485%	2.0515%	**1, 3,** 4, **5, 10**, 12, **17**, 18, 23, **35, 36**, 40	12
SampEn					
	Research	97.9485%	2.0515%	**1, 3,** 4, **5, 10,** 12, **17**, 18, 23, **35, 36**, 40	12
U2R					
Rényi					
	Lima	99.4758%	0.5242%	**13, 18**, **32**, 33, **36**	5
	Research	98.4848%	1.5152%	1, 4, **10, 13**, 19, 22, 25, 34, 35, **36**, 38	11
Shannon					
	Lima	99.5341%	0.4659%	**13,** 16, **17, 18, 32**, 33	6
	Research	99.0093%	0.9907%	3, 5, **13, 14,** 16, **17, 18**, 29, **32**, 34, **36**	11
	Sharma	80.8000%	19.2000%	1, 3, 5, 6, **10**, 11, **13**, **14**, 16, **17**, 31, **32**, 33, 34, **36**, 37	16
Tsallis					
	Lima	99.4176%	0.5824%	**13**, 16, **18**, **32**, 33	5
	Research	98.5431%	1.4569%	1, 4, 5, **10**, 14, **17**, **18**, 24, 30, **32**, 34, **36**, 37	13
ApEn					
	Research	98.6014%	1.3986%	2, 3, 6, **10**, 12, 14, **17, 18**	8
SampEn					
	Research	98.6014%	1.3986%	2, 3, 6, **10**, 12, 14, **17**, **18**	8

Table 4- Attack Classification Results

The features selected, and the quantity of features selected, varied significantly among the different entropy types and analysis sources. Table 4 details the results of these variations. The bolded feature numbers under the "**Features Selected**" column were common to more than 50% of the analysis results grouped by "**Attack**" type. The table also shows the relationship of the entropy type by attack type used in the calculation of correctly classified attacks and the number of features varied.

Table 5 presents a different view of the results from Table 4. Table 5 is sorted by the number of features found and entropy type, all grouped by attack type. It is observed that entries with the minimum number of features selected were not necessarily the best correctly classified results for the attack type. The results also showed that the lowest number of features selected varied between attack type and entropy type. The Sharma and Mukherjee (2012) results were not included since their approach did not use the J48 classification tree method.

The features selected by the approaches used by Lima et al. (2012) and this research produced varying results. However, there are commonalities among selected features. Table 6 lists the number of features selected by Lima et al. and this research along with the number of features selected which were in agreement with the results. These counts were taken from the preceding tables to compare the different entropy calculations used. This table demonstrates that even though significant variations in the results became evident, a subset of feature commonalities existed.

Attack	# Features	Source	Entropy	CC
DoS				
	7	Research	Shannon	99.9541%
	8	Lima	Tsallis	99.9586%
	8	Lima	Shannon	99.9495%
	9	Lima	Rényi	99.9632%
	10	Research	ApEn	99.8989%
	11	Research	SampEn	99.9081%
	14	Research	Rényi	99.4578%
	15	Research	Tsallis	99.9357%
Probe				
	10	Research	ApEn	98.4303
	10	Research	SampEn	98.4303
	11	Lima	Shannon	99.5031
	11	Lima	Rényi	99.4266
	11	Lima	Tsallis	99.3119
	12	Research	Tsallis	99.1577
	13	Research	Shannon	99.0046
	17	Research	Rényi	96.4778
R2L				
	10	Lima	Tsallis	98.8582%
	11	Research	Rényi	95.6107%
	12	Lima	Rényi	98.8582%
	12	Research	ApEn	97.9485%
	12	Research	SampEn	97.9485%
	13	Lima	Shannon	98.9058%
	13	Research	Shannon	98.4733%
	13	Research	Tsallis	98.1393%
U2R				
	5	Lima	Rényi	99.4758%
	5	Lima	Tsallis	99.4176%
	6	Lima	Shannon	99.5341%
	8	Research	ApEn	98.6014%
	8	Research	SampEn	98.6014%
	11	Research	Shannon	99.0093%
	11	Research	Rényi	98.4848%
	13	Research	Tsallis	98.5431%

Table 5 Listing of the Feature Selection Count sorted by Feature count

Category/ Entropy	Research	Lima	Agreement
DoS			
Rényi	14	9	4
Shannon	7	8	4
Tsallis	16	8	6
Probe			
Rényi	17	11	8
Shannon	13	11	5
Tsallis	12	11	3
R2L			
Rényi	11	12	4
Shannon	13	13	8
Tsallis	13	10	7
U2R			
Rényi	11	5	2
Shannon	11	6	5
Tsallis	13	5	2

Table 6 Summary of KDD CUP 99 attack selected features

In summary, the KDD CUP 99 analysis showed variations existed between the results presented by Sharma and Mukherjee (2012), Lima et al. (2012), and this research. The following were issues that influenced variations occurring in the results:

- Methodology used in the calculations for the Rényi and Tsallis entropies in the Lima et al. (2012), were very different from the approach used by Sharma and Mukherjee (2012).
- Subset of selected data for the analyses was different, since Lima et al. (2012) only listed attack counts.

SRI Malware Data

This section used a real-world data source to validated and compare the approach that analyzed the KDD CUP 99 data. This additional data source demonstrated the applicability of different entropy calculations for feature selection from a real-world collection of data, not previously used for feature selection. Phillip Porras, Director of the Computer Science Laboratory at SRI International, provided a source of collected malware intrusion data for this research. It consisted of files in pcap and other formats that contained malware alerts identified by the BotHunter (n.d.) project.

BotHunter was a project developed under the Cyber-TA research program by the Computer Science Laboratory at SRI International (BotHunter, n.d.). The system classified communications from both incoming and outgoing traffic at a network boundary. Algorithms detected potential malware intrusions by analyzing the sequence of events that occurred during the exchange using a customized version of Snort (n.d.), as noted in the BotHunter description. The events were classified and correlated to the activity of the malware life cycle model.

SRI Malware Data Acquisition and Analysis Method

Phillip Porras, at SRI, provided access to the Index of releases for a malware (n.d.) website used for this research. The website organized entries by days starting on May 1, 2008. Upon selection of a date, the corresponding page was displayed which was the "**SRI's Multiperspective Malware Infection Analysis Page**" for that date. Appendix P displays a sample of the web page.

Each row in "**SRI's Multiperspective Malware Infection Analysis Page**" contained an alert entry triggered by a match from a Snort rule. Each alert contained multiple links to individual text files, available for download, and the complete network communications session in compressed pcap format. For this research, ten days of alerts yielded 4,328 usable events. The following steps described the process used to acquire and synthesize the SRI data for this research.

1. **Primary web page.** The web page for each date was saved into a file named "**Multiperspective Malware Analysis Page.htm**". A page from each of the following dates was retrieved: 20080501, 20080502, 20080503, 20080504, 20080505, 20080506, 20080507, 20080508, 20080509, and 20080510. A separate directory hosted each date in which the associated malware files existed.

2. **Identification of files to retrieve.** Within the "**Multiperspective Malware Analysis Page.htm**" files, there were many html "href" tags referencing URL's to files for download. A Linux command string to read the "**Multiperspective Malware Analysis Page.htm**" files and select the URL's which contained character strings within the file name required for the analysis. The file types contained the character strings of "**pcap.gz**" and "**virus-labels**" within the URL. These character strings were entered into a file name "**ll**". Output of this command string created a script file named "***file.sh***" for subsequent execution. A Linux "**wget**" command prepended each

command line for the URL file retrieval. The following command string was used:

```
cat Multiperspective\ Malware\ Analysis\ Page.htm |fgrep -f ll|sed -e
"s/href/\r\nhref/g"|grep http|cut -f2 -d\"|sed -e "s/^/wget
/g"  >file.sh
```

The commands in this string sent the contents of the files to the **fgrep** command that selected lines contained in the character strings stored in file **ll**. The **sed** command put a carriage return and new line characters in front of the **href** tag. The **grep** selected lines that contained **http** character string. The **cut** selected the second field from the line using a double quote (**“)** as the delimiter. Lastly, the **sed** command put a **wget** character string at the beginning of the line.

3. **Download of files.** Appendix Q displays a portion of the resultant “**file.sh**” for the date of 20080501. This file ran on a command line with the named files downloaded to the current directory. Appendix R displays a sample listing of the downloaded files. The file extension designators are:
    a. **.pcap.gz** – the compressed pcap file from the session capture. Step 5 described the process that expanded these files.
    b. **.alerts** – contained the alerts generated by BotHunter.
    c. **.rules** – contained the Snort rules that generated an alert.
    d. **.alerts_botHunter.txt** – The report generated by BotHunter.

e. **.virus-labels** – Reported the analysis of the suspicious test results from VirusTotal (n.d.). It listed the viruses found in the pcap file from different Anti-Virus vendors. The file has a hex based name and may be associated with multiple pcap files. In some cases, multiple **.virus-labels** files are associated with one pcap file.

Note: Appendix U lists the above named files of **alerts**, **.rules**, **alerts_botHunter.txt**, and **virus-labels**.

4. **Organizing files.** As shown in Appendix R, the files names did not relate well to each other. In order to improve file management, a program listed in Appendix S, prepends a sequential numbering scheme to related files. Appendix T shows a partial listing of the files for 20080501 and their corresponding sequential numbering. The association showed the files with the date and numbering "**.associations**" extensions file in each date directory.

5. **Using tcpdump.** To process the binary **pcap.gz** files, the files were uncompressed using the Linux "**gunzip**" command. The "**tcpdump**" command produced a readable text of packet activity. The "**tcpdump**" command line used was used as follows:

```
cat pcap-file | tcpdump –r - > pcap-file.tcpdump
```

6. **Contents of tcpdump output.** The tcpdump files listed the packet activity that generated the malware attack records. WEKA used the features extracted from these records for the feature selection process using the J48 classification tree algorithm. Each of the expanded **pcap.gz** files generated between less than 100 lines to over 80,000 lines of activity after the files were processed by "**tcpdump**".

7. **Malware naming convention.** In the files with the **virus-labels** extension, a number of antivirus vendors were listed along with their assignment of their name for the malware evaluated. There was no standardized malware naming convention that existed among the different antivirus vendors. Only one vendor, **AntiVir**, produced malware entries in all of the files. Not all vendors had entries for all the malware files. The **AntiVir** vendor was selected as the antivirus program for naming the malware in this research. Using one vendor enabled a consistent and standard naming convention for this research; therefore, the malware named by **AntiVir** provided the naming convention for each occurrence.

8. **Raw data assembly.** The shell script named **bf.sh** collected all relevant raw data from the files and assembled the information into one file that generated the **.arff** file to use in WEKA. Appendix V lists the **bf.sh** script that ran in each date directory and the output generated was saved as ***date*.bf** such as

"**20080501.bf**". All ten of the ***date*.bf** files were concatenated into one file, named **SRI.bf**, which was used in the analysis.

9. **Feature selection.** The next task compiled all the features available for the assessment used in the analysis. The features selected originated from the **tcpdump** output, the **rules** files, and the **alerts** files. Reviewing the available features and identifying those that had an impact on this analysis resulted in a list of twenty-two features including the malware detected. Appendix W lists and describes the twenty-two features chosen from the available data.

10. **Extracting features.** Appendix Y lists the Java application that extracted the selected features from the data files. This program read the "**SRI.bf**" file as described in Step 8 and generated the results in a comma separated value (csv) format that was used in the data portion of the **arff** for WEKA. Additional information was manually added to the csv-formatted data to make it compliant with the WEKA **arff** formatting requirements. The resulting file header information was similar to that in Appendix K but designed for the SRI data. See Appendix X for a partial listing of the SRI data in **arff** format. The partial listing displayed the formatted header information for the **arff** information as required by WEKA at the start of the data portion. The total number of records for the data portion of the WEKA file was 4,328.

11. **Analysis in WEKA.** Analysis was conducted using WEKA with the **arff** file generated from the SRI Malware data store for May 1, 2008 through May 10, 2008. The file contained twelve malware attack types defined in Appendix U. The WEKA application ran each of the five entropy calculations using the J48 classification tree to determine which features influenced the results, and calculated statistics on the results.

SRI Malware Analysis Data Preparation

This section discusses the analysis of the real-world data that applied five entropy types to the feature selection process. In order to get different perspectives of the analysis, multiple sets of runs were conducted for each entropy calculation, varied the attributes, and focused on different labels and. The Shannon entropy was already included in the WEKA application and the Rényi, Tsallis, Approximate, and Sample entropy calculations were added to WEKA in a manner consistent with that used by Lima et al. (2012). The unique application of this research modeled the data to represent time-based sequences for the Approximate entropy (ApEn) and Sample entropy (SampEn) in the classification model similar to that of Yentes et al. (2013).

Defining the different combinations for the entropy calculations resulted in 221 unique parameter configurations that provided detailed results. Each run had the entropy type and associated parameters varied as described in Table 7. The tunable parameter for the Rényi and Tsallis entropy calculations, the α term, denoted the sensitivity to the probability distribution shapes. For the Approximate and Sample entropy calculations the tunable parameters were the window size, **m**, and the amount of the standard deviation

used, **r**. The results compared the number of features selected by the different entropy types. The data used was from the SRI BotHunter application and included the complete set of 4,328 records labeled with appropriate malware descriptors.

Entropy	Parametric values	Number of runs
Shannon	None	1
Rényi	**α** = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 0.99	11
Tsallis	**α** = 1.01, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, and 1.99	11
Approximate	**m** = 1, 2, 3, 4, 5, 6, 7, 8, and 9 **r** = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 0.99	99
Sample	**m** = 1, 2, 3, 4, 5, 6, 7, 8, and 9 **r** = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 0.99	99

Table 7 - Parametric values used in SRI analysis

Three sets of analyses conducted provided alternative views of the results. Each analysis utilized a different approach to the data labeling. The first analysis used the complete data set with the ten-malware types as listed in Appendix W. The goal was to identify the entropy type that produced the lowest number of features required when using all ten malware attacks as the labels.

The second analysis examined the ten-malware attacks individually by the generating ten different data sets, each analyzing one specific attack. This goal was to determine the least number of features required to identify individual malware attacks labeled within the whole set of data. Ten separate runs were made, one for each malware attack as the label.

This third analysis looked at the data by infection type labeled as the field **Enum** in Appendix W. This analysis grouped the different malware attacks based upon communication flows between the internal host and a set of external hosts into infection types identified by the BotHunter application Cheung and Valdes (2009). Each activity entry was labeled with the infection type for the analysis.

SRI Malware Results

Results of the three analyses are presented. Each one shows that the percent of correctly classified values were mostly very high and close to each other numerically, however, the number of features required to select the correct malware showed much wider variation. The details are discussed in the remainder of this section.

Analysis one – Data labeled by Malware attack

In this first analysis, the data was labeled with the appropriate malware attack as identified in Appendix W. Each entropy calculation varied the parameters with the number of runs as shown in Table 7. The complete results from the WEKA runs for this analysis are presented in Appendix Z. The lowest number of features required for each entropy type are listed in Table 8.

Results showed that Tsallis, Sample, and Rényi entropy required six features for correct classification of the ten-malware attacks. Tsallis entropy had the highest correctly classified value of 99.9312%, followed by Sample entropy at 99.2428%. Rényi entropy

also required six features, but the correctly classified percentage was 90.3167%, which was 8.9621% to 10.3888% lower than the Tsallis and Sample entropy correctly classified values. Shannon entropy had a correctly classified value of 99.8853% but required 8 features.

For the Tsallis, Sample, and Rényi entropy calculations, the minimum number of features occur at only one point for each of the parameter combinations. This is quite different for Approximate entropy since the minimum number of features occur for a wide range parameter combinations. As previously noted, all the results for this analysis are listed in Appendix Z. These results in Table 8 show that Tsallis and Sample entropy require the fewest number of features with the highest correct classification percentage.

Entropy	#FS	CC	Parameters
Tsallis	6	99.9312%	alpha=1.9
Sample	6	99.2428%	m=1; r=0.2
Rényi	6	90.3167%	alpha=0.5
Approximate	7	99.1051%	m=2;r=0.01, 0.1, 0.2 m=3,4; r=0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7 m=5,6,7,8,9; r=0.01,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,0.99
Shannon	8	99.8853%	

Table 8 – Lowest feature count by entropy type using the full set of SRI Malware data

Analysis two – Data labeled individually by Malware attack

Analysis two examined the number of features required to identify each specific malware attack by each entropy type using the entire data. In these runs, only one malware attack examined was labeled with the attack name, and the remaining entries

were labeled as "other". The number of WEKA runs for this analysis was 2,210 since there were 221 combinations of the entropy calculations used times the 10 malware attacks. WEKA was not able to produce the selection of features that identify the attack for several of the entropy combinations. In those cases, an "**NA**" was entered for the number of features. Different methods are available to provide data for missing points as in this case that are entered as "**NA**". Schafer and Graham (2002) identified different processes for interpreting missing data points. A discussion of these missing points is in Chapter 5. Table 9 shows the minimum number of features required for identifying the individual attacks by entropy type.

	Malware Type									
	BE1	**BE2**	**BE3**	**BH1**	**BH2**	**BH4**	**ET1**	**NB**	**SH**	**TFTP**
Approximate	NA	2	4	4	3	6	**3**	4	3	**1**
Renyi	NA	3	NA	**3**	3	NA	NA	**1**	**2**	NA
Shannon	**2**	**1**	**1**	**3**	3	4	**3**	**1**	3	**1**
Tsallis	NA	3	NA	**3**	3	**2**	**3**	**1**	**2**	**1**
Sample	NA	10	**1**	5	**2**	3	NA	**1**	NA	**1**
Min #FS	2	1	1	3	2	2	3	1	2	1

Table 9 – Minimum number of features required to identify the individual malware.

Results shown in Table 9 indicate that when considering only one malware attack at a time, the Shannon entropy has the highest success rate at requiring the minimum number of features for 7 malware types. The remaining order is Tsallis for 6 types, Sample for 4 types, Rényi for 3 types, and Approximate entropy for 2 types.

Analysis three – Data labeled individually by Infection type

The BotHunter application by SRI detected bot related malware into five common types of infections based upon the communications between the internal and external hosts (Cheung & Valdes, 2009). Two infections of the five identified by Cheung and Valdes are part of the SRI data store retrieved for this research and contain the ten different malware attacks as listed in Appendix W and described in Table 10. In this analysis, the label used was the infection designation: E2 or E3.

Infection	Description	Malware designations
E2	External-to-internal inbound exploit	BE1, NB, SH
E3	Internal-to-external binary acquisition	BE2, BE3, BH1, BH2, BH4, ET1, TFTP

Table 10 - Infection type description

The results presented in Table 11 show the minimum number of features required to identify the infection type for the data provided. All of the correct classification percentages are quite close to each other, so the primary difference is in the number of features required to identify the infection types in a single run by each entropy. The results showed that Sample and Rényi entropy each require 1 feature.

Infection	#FS	CC
Approximate	3	99.8853%
Renyi	1	99.6329%
Shannon	4	100.0000%
Tsallis	3	99.9312%
Sample	1	99.6329%

Table 11- Number of features selected for identifying E2 and E3 infection types

Summary

This chapter provided analytic details of the research using the KDD CUP 99 data and the SRI Malware data. Work consisted of algorithm development using Java for the Rényi, Tsallis, Approximate and Sample entropy algorithms and their integration into the open source WEKA application. Additional programs developed in Java and Linux shell scripts conducted data manipulation and management.

The first part of the research duplicated the work by Lima et al. (2012). This included creating files that represented the attacks from the KDD CUP 99 data in a manner that closely replicated the work of Lima et al. The classification mechanism was the WEKA J48 tree classification with the ten-fold validation option. The representative files were the run in separate WEKA executions using the Shannon, Rényi, and Tsallis entropy in the J48 decision tree calculations. From these runs, a comparison of the correct classification percentages and features selected were made. Results showed that the correct classification percentages were close to that of Lima et al.; however, the number of features selected varied. The next step included the use of Approximate and Sample entropy calculations in separate WEKA executions. In general, results showed that the correct classification percentages were close to that of Lima et al., however the number of features selected varied from the results of Lima et al. These summarized results are in Table 5 and Table 6.

The second part of the research applied a similar process as used for analysis of the KDD CUP 99 data to a set of real-world data that has not been used for this pupose. The source was from the SRI Cyber Threat Analysis lab using BotHunter application that captures malware attacks. Assembly of the data required the retrieveal and processing of

multiple files to identify and extract a set of features for the analysis. The WEKA application used the J48 classification model and determined the number of features required and correct classification percentage for the different sets of data. There were three different labeled groups of data used. Each set contained 4,328 activity entries. The first group used the different malware attacks as the labels to determine the number of featires needed to identify the attacks all together. The second set of data used all the data points one at a time. There were ten separate groups of runs made. The data containing the specific attack was labeled with that attack identifier, while the entries in the file were labeled as "other". The third set used the infection type as the label from which the data acquired contained two infection types. In addition to the three sets of data, the five entropy types had their parameters varied to cover a wide set of occurrences for a total of 221 runs per data set as detailed in Table 7.

The results showed the following:

- The first set of data labeled with all of the malware attacks showed that Shannon, Sample, and Rényi each required six features to identify the malware attacks. The Shannon and Sample entropies each had correct classifications above 99%. The Rényi entropy was at 90%. Approximate and Tsallis each had correct classifications over 99% but they required seven and eight features respectively. Table 8 contains these results.
- Data sets for the second analysis looked at each of the the malware attacks individually for a total of 2,210 runs. Results showed that all the entropies except Shannon had at least one instance at which it was not able to identify the specific malware. The minimum number of features required for the

Shannon had 7, Tsallis with 6, Sample with 4, Rényi with 3, and Approximate entropy with 2.

- The third analysis that labeled the data by infection type showed that Sample and Rényi entropies required only one feature, Tsallis entropy required 3 features, and Shannon entropy required 4 features. All of the entropy types had a correct classification rate of well over 99% with Shannon entropy at 100%.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

The work conducted in this research shows that using different entropy calculations and data labeling techniques in the feature selection process impacts the results when using intrusion detection data. This research discussed the use of entropy in feature selection and results achieved using the KDD CUP 99 data and SRI Malware data. Also identified were potential applications of this research to the intrusion detection processes and systems. The end of this chapter contains a summary of the entire research paper and provides a concise description of the research accomplished and potential applications in the use of this work.

Entropy calculations measure the randomness of data. Comparing two sets of entropy calculations provide a measure of information gained between the two measurements. Claude Shannon applied entropy to information processing in a paper in 1948. His formula was straightforward with no tunable parameters and adapted well for use with intrusion detection data. Lima et al. (2012) examined the impact of entropy on feature selection using Shannon's formula, they added Rényi and Tsallis entropy formulas, and analyzed the same set of data that produced different views of the results.

This research extended the techniques used by Lima et al. (2012) and added Approximate and Sample entropy to the feature selection process using intrusion detection data. Approximate and Sample entropy were typically used for biomechanical

data analysis in the past (Yentes et al. 2013). This is the first known use of Approximate and Sample entropy applied to intrusion detection data.

Rényi and Tsallis entropy contain a sensitivity factor, called alpha, within the calculation. This factor provided a mechanism that adjusted the impact of the probability distribution. Approximate and Sample entropy each have two variable parameters. Since these entropy types were time based, the variable **m** defined a sliding window that determined the number of points to consider at a time. The second variable is a multiplication factor, **r**, between zero and one, calculated a portion of the standard deviation for the number of points within the window.

Using a specified range of values for these parameters of the Rényi, Tsallis, Approximate, and Sample entropy enabled a profile of views of the results for the number of features, features selected, and correct classifications percentages. Different combinations of these variables produced significantly different results as shown is Appendix Z. To determine the best combination for the analysis, an inclusive range of variable values must obtain an overall view of the feature count, features selected, and the correctly classification result. This was accomplished by using the programs developed by the WEKA J48 tool during this research. The results produced a classification tree that showed how specific variable values and features contributed to the identification of the labeled data.

Lima et al. (2012) used Shannon, Rényi, and Tsallis entropy to select features from a subset of the KDD CUP 99 data set. Their results showed that the Rényi and Tsallis entropy calculation performed well with the C4.5 classification tree for feature selection with high correct percentage classification values. This research extended

Lima's work to include Approximate and Sample entropy. Results from Approximate and Sample entropy were in line with Lima et al. results for correct classification, however the feature count and features selected differed from the results Lima et al. reported. In order to provide validation to the inclusion of Approximate and Sample entropy to the C4.5 classification algorithm, an additional new intrusion detection data set was implemented. Data obtained from SRI Cyber Threat Analysis organization was assembled during this research into label data sets. This was the first time the SRI data was labeled and used to conduct feature selection research.

Research conducted by Lima et al. (2012) contained recommendations for Rényi and Tsallis entropy settings of the alpha term. Yentes et al. (2013) recommended settings for the variable parameter settings of window size and statistics terms in Approximate and Sample entropy. The recommendations from both these groups enabled their research to attain results that were conducive to their findings.

The work for this thesis extended these findings through the application of Rényi, Tsallis, Approximate, and Sample entropy into the C4.5 classification tree analysis using a well-known data set, the KDD CUP 99 data; and using a new set of real-world data not previously analyzed in this manner, the SRI Malware data. This unique approach produced results discussed in the following sections.

A data-mining package from the University of Waikato in New Zealand, called WEKA, was the data mining analysis tool used in this research. Supervised learning by the C4.5 decision tree method, developed by Quinlan (1986), was included in WEKA within the J48 classification module. Results from the J48 module produced a decision tree that identified features observed, their values in the tree, and correctly and

incorrectly classified results and percentages. A representation of this decision tree is located in Appendix N.

The C4.5 classification tree analysis was part of WEKA and implemented in Java in the J48 module. The J48 module included the Shannon entropy and this research developed the algorithms for Rényi, Tsallis, Approximate, and Sample entropy, and integrated them into the WEKA J48 calculation. Each of the Rényi, Tsallis, Approximate, and Sample entropy calculations contained parameters that were varied to optimize results. Based upon the entropy calculation used, and the parametric values chosen for those entropy calculations that contain variables, the resulting J48 classification tree identified the features required to classify the activity.

This research demonstrated that more than one view of the data provided additional options in the area of feature selection. The main methodology used to obtain different views included:

- **Labeling.** The classification tree lists the features and values needed to identify an attack or series of attacks based upon how the data was labeled.
- **Entropy and entropy variables values**. Created for the analysis, was a range of values for the alpha variable in the Rényi and Tsallis entropy calculation; and the **m** and **r** variables for Approximate, and Sample entropy calculations that generated a profile of the different values in the results.
- **Comparing different views**. Comparing the results from different entropy calculations or different views can identify commonalities of features that

impact the results. The results can also be used to identify unique features that define a specific attack that are not common to other attacks or views.

Using the output of the classification tree provided information for rule generation of an intrusion detection system. These rules can examine the traffic for capturing attacks traversing across the network. This assumed the static model was representative of the actual traffic.

In most dynamic networks, the traffic patterns change over time. To accommodate this change, periodically, another set of data would be collected and labeled appropriately. The new set of data was analyzed in a static manner as conducted in this research. Changes were made in the detection rules that were the most applicable to the traffic and resources available.

By having more than one view of the data available, some options become more applicable to different situations such as:

- Choosing entropy results that minimized the number of features and maximized correctly identified percentages.
- Tradeoffs among the number of features required, specific features identified, maximize correct performance, and the ability to extract features from the data stream.
- Select the labeling method that provided the greatest advantage for the situation. This may be a result that overlapped certain features for the developed rules that share common features.

These different scenarios described show some of the power and applicability of using the analysis techniques put forth in this research. One application of these results may be for rule development in that the information provides a potential starting point that can improve, reduce, or stop malicious activity from infecting the computing networks and computer systems. Traffic composition generally changes over time and a periodic re-evaluation of the parameters should be performed to maintain the freshness of the detection capabilities.

Another aspect of the data used in feature selection involved labeling of the data. Data labeling enabled multiple views of the results based the entropy, values used in the entropy calculation, and the labels of the attack categories. Labeled data enabled the use of supervised learning and the method of data labeling defined supplemented the view. In the KDD CUP 99 data set, the data was labeled by attack type. In other data sets, the labels may be a specific attack or attack groupings. The SRI results section discussed the different types of data labeling actions conducted during this research

KDD CUP 99 Data Conclusions

In the KDD CUP 99 data, each activity entry was labeled by a specific category. The J48 calculation and selected entropy produced results that named features and their associated values used in the classification tree. Table 2 lists the different categories and specific attacks within the KDD CUP 99 data. This labeling enabled the quantity and

identification of features that classified the specific attack and normal traffic. This analysis was conducted using the four attack categories with each of the five entropy types. Also, the J48 classification tree included the both the correctly and incorrectly classified results and percentages. This provided information as to the effectiveness of a specific entropy with the associated parametric settings.

Constructing sets of KDD CUP 99 data with the same distribution of attacks used by Lima et al. (2012) enabled the comparison of the results with this research. The analysis employed the WEKA J48 classification tree that identified features and the correct classification results for the four attack categories. Conclusions derived from the results displayed in Table 5 for the KDD CUP 99 data, were as follows:

- The correct classifications of attacks were very similar in the DoS (within 0.5%) and U2R (within 1.0%) categories but the number of features selected varied by a count of 8 for both DoS and U2R.
- The Probe and R2L attack categories varied more in the correct classification of the attacks. Nearly 3.0% for Probe and 3.3% for R2L. The overall feature counts varied by 7 for Probe and 3 for R2L.
- The Approximate and Sample entropy values for correctly classified attacks were within 1% of the best values and were the same for the Probe, R2L, and U2R attack categories. The features selected were also the same for these categories.
- The Approximate and Sample entropy values for the DoS attack category were also within 1% of the best values, however the Sample entropy was slightly better in the results.

- The features selected were different between the Approximate and Sample entropy values for the DoS attack category. This was due large to the number of duplicate entries in this attack category as compared to the other attack categories.
- The absence of Sharma and Mukherjee (2012) results was due to their use of a single entropy, that of Shannon, and they used the Naïve Bayes classifier as their method of feature selection. The results that they achieved produced lower correct classified values than in this research and that of Lima et al. (2012).

Observed differences in the results between Lima et al. (2012) and this research were partly due to the selection of the specific records from the KDD CUP 99 data set. Other differences observed included the implementation of the Rényi and Tsallis entropy calculations within WEKA. Multiple random selections of records from KDD CUP 99 resulted in varied correctly classified percentages. Since the exact KDD CUP 99 records used by Lima et al. were unknown, the approximation of the attack make up used the counts documented by Lima et al. Chapter 4 addressed inconsistencies with the attack counts in this research.

SRI Malware Data Conclusions

The WEKA J48 tree classification method calculated the results for this research using real-world SRI Malware data with the Shannon, Rényi, Tsallis, Approximate, and Sample entropies and this was the first time this data was used for entropy research.

Depending on the variable parametric values, each entropy type produced some results with a low number of features but most had high correct classification percentages as shown in Table 8.

Tsallis and Sample entropy produced the highest correctly classified results of 99.9312% and 99.2428% respectively, and with the least number of features of 6. Rényi entropy also required only 6 features but resulted in the lowest correctly classified results of 92.3818%. Appendix Z contains the complete results of the SRI data analysis showing the entropy calculations used, parameter settings, correctly classified results, and the number of features selected.

As seen in the results of this research with the SRI data, the agreement with the correct classification was within the same general range as that of the KDD CUP 99 data. In analyzing SRI data, no previous research existed for this type of study. Most of the results were high in values for the correct classification and low for the number of features selected, which varied minimally as shown in Table 8.

When considering the individual malware attacks, described in the analysis of the SRI data, the number of features required dropped significantly as detailed in Table 9. Examining an individual attack, and generalizing all the remaining data as "other", allowed the analysis to focus on the one specific type of attack and selected these features that only identified those activities.

When labeling the data by infection type, as denoted by **enum** in Appendix W, results were more pronounced as shown in Table 11. The correct classification percentages were all over 99% for each entropy type, but the number of features required to detect the infection was only 1 for Sample and Rényi entropies, 3 for Approximate and

Tsallis entropies, and 4 for Shannon. The infection types along with their associate malware attacks are in Table 10.

When evaluating results of the three analyses together, the averages showed how the different entropies and labeling strategies significantly impacted the number of features required. The row numbers in Table 12 below referenced a different table containing the data. In row 1, the number of features for each entropy type originated from Table 8 that corresponds to **Full Set of SRI data**. In row 2, the number of features for each entropy type is from Table 9 that corresponded to **Individual Malware**. The number of features for each entropy type in row 3 is from Table 11 that corresponds to **Infection Type**.

The WEKA J48 classification tree analysis was unable to produce results with certain attack/entropy combinations. Some of the data points in Table 9 contain an "**NA**" for values. In order to quantify these points, "**NA**", a method described by Schafer and Graham (2002) as the *Available-case analysis,* was used. This process considered a pairwise inclusion of data to estimate the missing values. The pair assumed for this analysis, is the maximum value determined by other calculations, which was 10. Values other than 10 for the substitution of "**NA**" produced similar representative results.

Labeling	Shannon	Rényi	Tsallis	Approximate	Sample	Table Ref
Full Set of SRI data	8.0	6.0	6.0	7.0	6.0	8
Individual Malware	2.2	6.2	4.2	4.2	5.9	9
Infection Type	4.0	1.0	3.0	3.0	1.0	11
Average	4.7	4.4	4.3	4.7	4.1	

Table 12 - Average number of features required from the three SRI analyses

Results, from Table 12, show how the number of features vary based upon the data labeling used. The feature count and average feature count varies for the different analyses by using the complete set of data and adjusting the labeling to focus on particular views.

Overall Conclusion

Labeling of data affects the results produced. The results from the classification tree provided the number of features, the features selected, and the correct classification percentages. The KDD CUP 99 data used only one labeling method in an effort to show how the Approximate and Sample entropy calculations impacted the classification tree results.

By having different labeling scenarios available, the selection of features that best identified a specific attack was possible in an intrusion detection system. This also supported the proposal that subsequent analyses be conducted periodically with a new collection of labeled data to keep intrusion detection current with changes occurring in network traffic patterns. Knowledge gained from previous results using combination of entropies and their variable assignments may shorten the analyses of the newly collected data.

Implications

This research supported previous work that showed the feature selection actually reduced the number of fields required to analyze intrusion detection data. By analyzing different models through the labeling of data to detect specific attacks or groupings of

attacks provided additional views of the results and assisted in the selection and application that best fit the purpose of the analysis. These techniques should apply to other types of data to be analyzed. Where others have used biomechanical data for classification and feature selection, this research used intrusion detection data.

Selection of tunable parameters for the entropy type, in conjunction with C4.5 classification tree analysis, produced differences in results especially in the features selected and number of features selected. Duplicate entries within the data set influenced the Sample entropy calculations. Since the Sample entropy suppressed duplicate records in the calculations, it generally produced better results than Approximate entropy especially with the malware data that contained a large number of duplicate records.

Recommendations

The use of different labels for the data provided more than one view of the data. This enabled the user and implementer additional information from which to choose the best results for their needs. These results can assist in better performance of an intrusion detection system for specific attacks or groups of attacks by selecting the best combination of features that identify the attack.

Varying the values of the tunable parameters in the entropy calculations affected the results of feature selection. To obtain the best results required multiple runs using sequential variation of the parameters that developed a set empirical data to identify the optimal number of features required. These runs should be conducted periodically to represent the current network traffic.

This research focused on the WEKA J48 classification method as used by Lima et al. (2012). Results of this thesis supported the additional study of the benefits of entropy using Shannon, Rényi, Tsallis, Approximate, and Sample entropy for intrusion detection systems data. Different models of the data looked at specific attacks or groupings of attacks through the labels associated with the activity. Variation of the tunable parameters demonstrated how the features selected could differ based upon the data labeling strategy chosen.

Summary

The goal of this research was to minimize the number of features required in a set of intrusion detection data that rapidly identifies malicious activity while maintaining the integrity of the data set. By reducing the redundant features, irrelevant features, and noise, it was possible to decrease the number of features required that rapidly analyze the data for identifying anomalous activities. This focus was on the application of entropy in feature selection process of labeled data sets. Feature selection is the process of identifying features from a set of data that are relevant to the analyses. Reasons for feature selection are to reduce the size of the data set and more efficiently use the computing resources. Using the C4.5 decision tree developed by Quinlan (1986), the results can be readily adapted for developing rules for intrusion detection systems. This work also showed the positive impact of how different labeling methods provide additional results for implementation of intrusion detection mechanisms.

This work began by replicating the work of Lima et al. (2012) who used C4.5 classification tree method in the WEKA data mining analysis tool to select features from intrusion detection data. They used a subset of the well-known KDD CUP 99 data file for selecting the features required to identify the four-attack types labeled in the data using Shannon, Rényi, and Tsallis entropy. WEKA includes the Shannon entropy and Lima et al. added the algorithms for Rényi and Tsallis.

The research for this thesis added the Rényi and Tsallis to WEKA, and replicated the results of Lima et al. (2012) to develop a base line of reference that validated the entropy calculations developed for the records and integrated into the data-mining tool.

Results showed good agreement with Lima et al. for correct classification percentages, however, the number of features and specific features varied.

This research extended the use of entropy in feature selection to include Approximate and Sample entropy typically used in time-series based data. Yentes et al. (2013) used Approximate and Sample entropy for feature selection with biomechanical data. Analysis using these two entropies demonstrated new results with the KDD CUP 99 data.

This research programmed four entropy calculations that were added to WEKA source code. Each formula contained variables that could be optimized to obtain the result. Rényi and Tsallis contained an alpha term that adjusted the sensitivity of the entropy calculation's impact on the results. Approximate and Sample entropy contained two adjustable parameters. Since they were time based, one parameter, named **m**, defined how many points were included in a calculation. The second variable, **r**, defined the portion of the standard deviation of the data within the window size, **m**, to use in the entropy calculation. Results of using Approximate and Sample were similar to correct classification percentage that Lima et al. (2012) achieved. Again, the number of features, and specific features were different.

In order to validate the process developed for this research, a second set of data from SRI Cyber-TA BotHunter (n.d.) project was used. This data was retrieved from a series of files that collected different features of the malware attack. Assembling the SRI data into a usable format for the WEKA analysis, required the development of additional applications using Java and Linux shell scripts that enabled the extraction, conversion, normalization, and labeling of the data.

Three types of analysis were conducted with the SRI data that used three different labeling methods. The first method labeled each row of activity by the specific malware identified for that row. The second method labeled considered only one malware attack for the rows of that malware and with the remaining rows labeled with "other" resulting in ten files for analysis. The third method labeled each row with one of the two infection types identified in the SRI data files. These different labeling methods enabled the results to show how the combination of labels, different entropy calculations, and variable values influenced the number of features, features selected, and the correct classification results.

The following lists the values for the variables chosen for the Rényi, Tsallis, Approximate, and Sample entropy calculations:

Rényi

alpha = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 0.99.

Tsallis

alpha = 1.01, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, and 1.99.

Approximate and Sample

m = 1, 2, 3, 4, 5, 6, 7, 8, and 9.

r = 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 0.99.

The C4.5 classification tree, developed by Quinlan (1986), was implemented in the WEKA analysis package as the J48 module programmed in Java. The code developed for this research for the Rényi, Tsallis, Approximate, and Sample entropy calculations was integrated into the J48 module. Output from the J48 module provided a classification

tree that identified the structure with features names, their values, and dependencies as shown below.

Partial sample of a classification tree

```
SWBn = 0
|   seqRange <= 102: other (349.0)
|   seqRange > 102
|   |   toPort <= 8147
|   |   |   priority <= 1: WIN (2127.0)
|   |   |   priority > 1
|   |   |   |   toPort <= 470: other (70.0)
|   |   |   |   toPort > 470: WIN (305.0)
|   |   toPort > 8147
|   |   |   toPort <= 9996: other (35.0)
|   |   |   toPort > 9996: WIN (16.0)
SWBn = 1
|   enum = E2
|   |   seqRange <= 1260: WIN (4.0)
|   |   seqRange > 1260: other (34.0)
|   enum = E3: other (1418.0)
```

o
o
o

In the table below, multiple labeling methods demonstrated how the number of features varied based upon how the data was labeled. The row labeled "Individual Malware" was an average of the number of features required to identify the ten unique malware attacks. The bottom row labeled "Average" was an equally weighted average of the three labeling methods in the table.

Labeling	Shannon	Rényi	Tsallis	Approximate	Sample
Full Set of SRI data	8.0	6.0	6.0	7.0	6.0
Individual Malware	2.2	6.2	4.2	4.2	5.9
Infection Type	4.0	1.0	3.0	3.0	1.0
Average	4.7	4.4	4.3	4.7	4.1

Since the overall composition of packets that flow across a network may evolve over time, analyzing the features and their values used for the rules periodically focused to identifying new attacks to the network and the computer systems attached. Conducting subsequent analyses would require collecting traffic into a set of files used for analysis. This collection may consist of a complete set of traffic at collection points, only packet headers, or the traffic may be a filtered to eliminate known good and/or known bad traffic.

This research showed how several different techniques in the use of entropy for feature selection provided benefits by reducing the volume of data for identifying attacks against computers and networks. Other accomplishments included:

- The five entropy calculations made showed how the results differ for number of features selected, specific features selected, and correctly classified results.
- The classification tree output provided the information needed to identify the number of features, the specific features, and the correctly classified results for the labeled set of data.
- Different labeling methodologies had an impact by constructing a table that shows the results referencing the number of features required across multiple scenarios.
- When using Rényi, Tsallis, Approximate, and Sample entropy, the values assigned to the parameters impacted the results. By substituting a range of

values for those parameters, a profile of the results created to determine the least number of features required to identify a specific attack.

- Using the classification tree output, the parameters for use in an intrusion detection are presented and directly applicable to rule development.

Other implications from this research were identified that the analysis must be conducted periodically to maintain operation of the intrusion detection system at optimal performance level as network traffic changes over time. This research showed positive impact and advanced the feasibility of using multiple entropy calculations to reduce the number of features required to identify specific methods for intrusion detection data. The full complements of analyses available demonstrated the different options available to identify malware. It also showed how labeling the data could optimize the number of features selected as shown in the different examples. An implementer can apply these results to the intrusion detection system based upon their needs and environment of the networks and computer systems.

Appendices

Appendix A: Listing of Modified EntropyBasedSplitCrit Class

This is the Java listing of the modified EntropyBasedSplitCrit Class.

```
/*
 *    This program is free software; you can redistribute it and/or modify
 *    it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation; either version 2 of the License, or
 *    (at your option) any later version.
 *
 *    This program is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *    GNU General Public License for more details.
 *
 *    You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software
 *    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

/*
 *    EntropyBasedSplitCrit.java
 *    Copyright (C) 1999 Eibe Frank
 *
 */


/*
 *    This class has the following modifications made in order
 *    to accommodate the use of different entropy calculations
 *
 *      Method varInitialize() = initializes entropy variable
 *      by instantiating a call to the method that reads the
 *      configuration file. Also prints out the parameters used.
 *
 *      Method logFunc() = modified to use the different entropy calculations
 *      calculations.
 *
 *      Method oldEnt() = modified to use the different entropy calculations
 *      calculations.
 *
 *
 *    Author: Frank Acker - December 2014
 */

package weka.classifiers.trees.j48;

/**
 * "Abstract" class for computing splitting criteria
 * based on the entropy of a class distribution.
 *
 * @author Eibe Frank (eibe@cs.waikato.ac.nz)
 * @version $Revision: 1.5 $
 */
public abstract class EntropyBasedSplitCrit extends SplitCriterion{

      private static final long serialVersionUID = 1L;

/** The log of 2. */
  protected static double log2 = Math.log(2);
  public double alpha; // for Reni and Tsallis entropy
  public int ET; // entropy type number
```

```
  public int m; // window size for ApEn and SampEn
  public double r; // & of standard deviation for ApEn and SampEn.
  public  boolean D; // Debugger

  private static ApproximateEntropy ae = new ApproximateEntropy();
  public static SampleEntropy se       = new SampleEntropy();

  private static boolean readFile = false; // read file once per execution
  public static entUtils eu = new entUtils();
  public static fileRead fr = new fileRead();



  public final void varInitialize() {
             eu.getFileInfo();

             ET    = fr.getEType();
             alpha = fr.getAlpha();
             m     = fr.getM();
           r     = fr.getR();
           D     = fr.getD();
           String C = ",";
           if (D) System.out.println("varInitialize - readFile");
           if (D) System.out.println("ET, alpha, m, r, D " + ET + C + alpha + C
+ m + C + r + C + D);
  }
  /**
   * Help method for computing entropy.
   */
  public final double logFunc(double num) {

         if (!readFile) {
              readFile = true;
         varInitialize();
         if (D) System.out.println("logFunc - readFile");
         }

    // Constant hard coded for efficiency reasons
    if (num < 1e-6) return 0;
    if (ET == 0) return num*Math.log(num)/log2;
    if (ET == 1) return Math.pow(num, alpha);
    if (ET == 2) return Math.pow(num, alpha);
    if (ET == 3) {
       return num*Math.log(num)/log2;
    }
    if (ET == 4) {
       return num*Math.log(num)/log2;
    }
    return -99.0; // entered to satisify eclipse
  }

/**
   * Computes entropy of distribution before splitting.
   */
  public final double oldEnt(Distribution bags) {

    double returnValue = 0;
    int j;
    if (D) System.out.println("oldEnt - bags.numClasses()="+bags.numClasses());

    if (ET == 0 || ET == 1 || ET == 2) {
       for (j=0;j<bags.numClasses();j++) {
           returnValue = returnValue + logFunc(bags.perClass(j));
```

```
          if (D) System.out.println("oldEnt - j="+j+"
bags.perClass(j)="+bags.perClass(j));
       }
    }
    if (ET == 3 || ET == 4) {
       double [] apsampClasses = new double[bags.numClasses()];
       for (j = 0; j < bags.numClasses(); j++) {
             apsampClasses[j] = bags.perClass(j);
             if (D) System.out.println("oldEnt AE - j="+j+"
bags.perClass(j)="+bags.perClass(j)+" apsampClasses[j]="+apsampClasses[j]);
       }
       if (ET == 3) returnValue = returnValue + ae.ApEn(apsampClasses,m,r);
             if (ET == 4) returnValue = returnValue +
se.SampEn(apsampClasses,m,r);
    }

    if (D) System.out.println("oldEnt - bags.total()="+bags.total());
    if (ET == 0 || ET == 3 || ET == 4) return logFunc(bags.total()) -
returnValue;
    if (ET == 1) return (((Math.log (logFunc(bags.total()) )/log2)/(1.0 -
alpha)) - ((Math.log(returnValue)/log2))/(1.0 - alpha));
    if (ET == 2) return ((logFunc(bags.total()))/(alpha - 1.0)) - (returnValue
/(alpha - 1.0));
    return 0.0;
  }

  /**
   * Computes entropy of distribution after splitting.
   */
  public final double newEnt(Distribution bags) {

    double returnValue = 0;
    int i,j;
    if (D) System.out.println("newEnt - bags.numBags="+bags.numBags());
    if (D) System.out.println("newEnt - bags.numClasses="+bags.numClasses());
    for (i=0;i<bags.numBags();i++){
      if (ET ==0 || ET == 1 || ET == 2) {
       for (j=0;j<bags.numClasses();j++) {
           returnValue = returnValue+logFunc(bags.perClassPerBag(i,j));
           if (D) System.out.println("newEnt - i,j="+i+","+j+"
bags.perClassPerBag(i,j)="+bags.perClassPerBag(i,j) + "
returnValue="+returnValue);
       }
      }
      if (ET == 3 || ET == 4) {
       double [] apsampClasses = new double[bags.numClasses()];
       for (j = 0; j < bags.numClasses(); j++) {
             apsampClasses[j] = bags.perClass(j);
             if (D) System.out.println("oldEnt AE/SE - j="+j+"
bags.perClass(j)="+bags.perClass(j)+" apsampClasses[j]="+apsampClasses[j]);
       }
       if (ET == 3) returnValue = returnValue + ae.ApEn(apsampClasses,m,r);
             if (ET == 4) returnValue = returnValue +
se.SampEn(apsampClasses,m,r);
      }
      returnValue = returnValue-logFunc(bags.perBag(i));
      if (D) System.out.println("newEnt - i="+i+"
bags.perBag(i)="+bags.perBag(i)+" returnValue="+returnValue);
    }
    return -returnValue;
  }

  /**
```

```
   * Computes entropy after splitting without considering the
   * class values.
   */
  public final double splitEnt(Distribution bags) {

    double returnValue = 0;
    int i;
    if (D) System.out.println("splitEnt");
    for (i=0;i<bags.numBags();i++)
      returnValue = returnValue+logFunc(bags.perBag(i));
    return logFunc(bags.total())-returnValue;
  }
}
```

Appendix B: Listing of ApproximateEntropy Class

```
package weka.classifiers.trees.j48;

/**
 * Class for computing Approximate Entropy
 * based on the entropy of a class distribution.
 *
 * @version $Revision: 1.0 $
 */
/* Approximate Entropy
 * basic code retrieved Oct 18, 2014 from
 * http://www.codeproject.com/Articles/27030/ \
 *     Approximate-and-Sample-Entropies-Complexity-Metric
 *
 * It has since been modified for use in this application.
 *
 *    Author: Frank Acker - December 2014
 */

public  class ApproximateEntropy extends EntropyBasedSplitCrit{

  /**
       *
       */
      private static final long serialVersionUID = 1L;


  /* Approximate Entropy
   * basic code retrieved Oct 18, 2014 from
   * http://www.codeproject.com/Articles/27030/Approximate-and-Sample-
Entropies-Complexity-Metric
   *
   * It has since been modified for use in this application
   */

public double ApEn(double data[], int m, double r) {

  /*
   *  data[] is an double array of the data collected for ApEn calculation
   *  m is the window size = default is 2
   *  r is the multiplier of the standard deviation to use  = default to 0.2
   */

      entUtils eu = new entUtils();
      fileRead fr = new fileRead();

      boolean D = fr.getD();

          int Cm = 0, Cm1 = 0, i, j, k;
          int N = data.length;
          // check that m is not less than the data length
          if (m > N)
               m = N;

          double err = 0.0, sum = 0.0;
          //double r = 0.2;
          // Calculate std dev
          double std = eu.stdev(data);
          err = std * r;
          if (D) System.out.println("ApEn - N="+N+" r="+r+" std="+std+"
err="+err);
```

```
          for ( i = 0; i < N - (m + 1) + 1; i++) {
             Cm = Cm1 = 0;
             for ( j = 0; j < N - (m + 1) + 1; j++) {
                boolean eq = true;
                for (k = 0; k < m; k++) {
                    if (D) System.out.println("ApEn - i,j,k="+i+","+j+","+k+"
data[i+k]="+data[i+k]+" data[j+k]="+data[j+k]+" err="+err);
                    if (Math.abs(data[i+k] - data[j+k]) > err) {
                       if (D) System.out.println("ApEn - Math.abs(data[i+k] -
data[j+k]) > err is true");
                          eq = false;
                          break;
                    }
                 }
                 if (eq) Cm++;
                 k = m;
                 if (eq && Math.abs(data[i+k] - data[j+k]) <= err)
                     Cm1++;
             }

             if (Cm > 0 && Cm1 > 0){
               double dCm  = (double)Cm;
               double dCm1 = (double)Cm1;
               if (D) System.out.println("ApEn -dCm="+dCm+" dCm1="+dCm1);

               sum += Math.log(dCm / dCm1)/log2;
             }
          }
          if (D) System.out.println("ApEn - N="+N+" m="+m+" sum="+sum);
          double apenreturnvalue;
          if ((N - m) == 0) {
               apenreturnvalue = 0;
          } else {
               apenreturnvalue = sum / (double)(N - m);
          }

          if (D) System.out.println("ApEn - apenreturnvalue="+apenreturnvalue);
          return apenreturnvalue;
    }
}
```

Appendix C: Listing of SampleEntropy Class

```
package weka.classifiers.trees.j48;

/**
 * Class for computing Sample Entropy
 * based on the entropy of a class distribution.
 *
 * basic code retrieved Oct 18, 2014 from
 * http://www.codeproject.com/Articles/27030/
 *
 *     Approximate-and-Sample-Entropies-Complexity-Metric
 *
 * It has since been modified for use in this application
 *
 *    Author: Frank Acker - December 2014
 *
 * @version $Revision: 1.0 $
 */

public class SampleEntropy extends EntropyBasedSplitCrit{

       private static final long serialVersionUID = 1L;

       public double SampEn(double data[], int m, double r) {

                /*
                 *  data[] is an double array of the data collected for ApEn
calculation
                 *  m is the window size = default is 2
                 *  r is the multiplier of the standard deviation to use  =
default to 0.2
                 */

         int N = data.length;
         // check that m is not less than the data length
         if (m > N)
          m = N;

         entUtils eu = new entUtils();

         fileRead fr = new fileRead();
         boolean D = fr.getD();

         int Cm = 0, Cm1 = 0, i, j, k;
         double std = eu.stdev(data);
         double err = std * r;
         if (D) System.out.println("SampEn - N="+N+" r="+r+" std="+std+"
err="+err);
         for (i = 0; i < N - (m + 1) + 1; i++) {
                 for (j = i + 1; j < N - (m + 1) + 1; j++) {
                         boolean eq = true;
                         //m - length series
                         for (k = 0; k < m; k++) {
                            if (D) System.out.println("SampEn -
i,j,k="+i+","+j+","+k+" data[i+k]="+data[i+k]+" data[j+k]="+data[j+k]+"
err="+err);
                            if (Math.abs(data[i+k] - data[j+k]) > err) {
                              if (D) System.out.println("SampEn -
Math.abs(data[i+k] - data[j+k]) > err is true");
                                eq = false;
                                break;
```

```
                        }
                    }
                    if (eq) Cm++;

                    //m+1 - length series
                    k = m;
                    if (eq && Math.abs(data[i+k] - data[j+k]) <= err)
                            Cm1++;
                }
            }

            if (Cm > 0 && Cm1 > 0) {
                 double dCm  = (double)Cm;
               double dCm1 = (double)Cm1;
                 return (Math.log(dCm / dCm1))/log2;
            } else {
                return 0.0;
            }
        }
}
```

Appendix D: Listing of fileRead Class

```
package weka.classifiers.trees.j48;
import java.io.*;

/*
 *    This class was written in order to
 *    support the reading of the configuration
 *    file and parse its parameters.
 *
 *    Author: Frank Acker - December 2014
 */


public class fileRead extends EntropyBasedSplitCrit{

       /* The Entropy Type (etype) is designated in the
        * Entropy Information file defined as variable
        * in this class as: "fileName".
        *
        * The entropy indicator number is as follows:
        * etype = 0 - Shannon
        * etype = 1 - Rényi
        * etype = 2 - Tsallis
        * etype = 3 - Approximate
        * etype = 4 - Sample
        */

       public static int etype;
       public static boolean D = false; // debugger switch

       /* The alpha term is used in the Rényi and Tsallis entropy calculations.
        * If no alpha term is defined in the value is set to a 0.
        * The default for Rényi entropy is 0.5.
        * The default for Tsallis entropy is 1.2.
        */
       public static double alpha;
       double defaultAlpha = 0;
       double defaultRényiAlpha = 0.5;
       double defaultTsallisAlpha = 1.2;

       /* "r" is a measure of the percentage of the standard deviation
        * to consider for ApEn and SampEn.
        * Default is 0.2 as defined by Yentes et al. (2013) but must
        * be entered in the ENtropy Information File
        */
       public static double r = 0.2;
       double defaultR = -1.0;
       public final String fileName = "/media/sf_nova/data/EntropyInfoFile.txt";

       /* "m" is a windows size to use for the series length
        * Used for ApEn and SampEn.
        * Default is 2 as defined by Yentes et al. (2013) but must
        * be entered in the ENtropy Information File
        */
       public static int m = 2;
       int defaultM = -1;
       public long filemod = 0;

       public boolean updatedFile(){
       File file = new File(fileName);
       long ifilemod = file.lastModified();
```

```
       if (ifilemod == filemod)
              return false;
       else {
              System.out.println("updtaedFile - ifilemod= "+ ifilemod + "
filemod="+ filemod);
              filemod = ifilemod;
              return true;
       }
       }

    public boolean fileRead() {

        // This will reference one line at a time
        String line = null;
        try {
            // FileReader reads text files in the default encoding.
            FileReader fileReader =
                new FileReader(fileName);

            // Always wrap FileReader in BufferedReader.
            BufferedReader bufferedReader =
                new BufferedReader(fileReader);

            while((line = bufferedReader.readLine()) != null) {

             // get rid of any spaces in line
                String line1 = line.replace(" ","");
                String[] parts = line1.split("=");

                if (parts[0].equals("etype")) {
                    etype = Integer.parseInt(parts[1]);
                    if (fr.D) System.out.println("=+=etype="+etype);

                }

                if (parts[0].equals("alpha")) {
                    alpha = Double.parseDouble(parts[1]);
                    if (fr.D) System.out.println("===ET="+etype+ " alpha ="+
alpha);

                }

                if (parts[0].equals("m")) {
                    m = Integer.parseInt(parts[1]);
                }

                if (parts[0].equals("r")) {
                    r = Double.parseDouble(parts[1]);
                }

                if (parts[0].equals("D")) {
                    System.out.println("fileRead - Hit D - parts[1]="+parts[1]);
                    if (parts[1].equals("true")) D = true;
                    if (parts[1].equals("false")) D = false;
                    //if (parts[1].equalsIgnoreCase("true")) D = true;
                    //if (parts[1].equalsIgnoreCase("false")) D = false;
                    System.out.println("D="+D);
                }
            }
            // Always close files.
            bufferedReader.close();
        }
```

```
        catch(FileNotFoundException ex) {
             System.err.println("Unable to open file '" + fileName + "'");
             return false;
        }
        catch(IOException ex) {
            System.err.println("Error reading file '" + fileName + "'");
            return false;
        }

        // check for valid etype
        if (etype <0 || etype >4) {
             System.out.println("Bad or no Entropy Type (etype) defined.
Entered value:" + etype);
             return false;
        }

        // check Rényi alpha
        if (etype == 1 && alpha == defaultAlpha) {
             alpha = defaultRényiAlpha;
             System.out.println("Alpha for Tsallis Entropy set to " + alpha);
        }

        // check Tsallis alpha
        if (etype == 2 && alpha == defaultAlpha) {
             alpha = defaultTsallisAlpha;
             System.out.println("Alpha for Tsallis Entropy set to " + alpha);
        }

        // check Approximate and Sample entropy values
        if (etype == 2 || etype == 3) {
             if (r == defaultR) {
                    System.out.println("No \"r\" value for Approximate Entropy
entered");
                    System.exit(1);
             }
             if (m == defaultM) {
                    System.out.println("No \"m\" value for Approximate Entropy
entered");
                    System.exit(1);
             }
        }
        return true;
    }

    public int getEType () {
       if (fr.D) System.out.println("getEType="+etype);
       return etype;
    }

    public double getAlpha() {
       return alpha;
    }

    public String getETName() {
       String etname[] = new String[] {"Shannon", "Rényi", "Tsallis",
"Approximate", "Sample"};
       return etname[etype];
    }

    public int getM() {
       return m;
    }
```

```
    public double getR() {
       return r;
    }

    public boolean getD() {
       if (D) System.out.println("getD D="+D);
       return D;
    }
}
```

Appendix E: Listing of entUtils Class

The following is a list of the entUtils class which is used in support of the entropy calculations.

```
package weka.classifiers.trees.j48;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;


/* entUtils
 * This class contains utilities for the entropy calculations,
 * standard deviation, and prints the configuration
 * files parameters settings.
 *
 * Author: Frank Acker December 2014.
 */


public  class entUtils extends EntropyBasedSplitCrit {

       private static final long serialVersionUID = 1L;

       public double stdev(double data[]) {
                      int i;
                      double mean = 0.0;
                      int dlen = data.length;
                      double sum1 = 0.0;
                      for (i = 1; i < dlen; i++)
                             mean += data[i]/dlen;
                      for (i = 1; i < dlen; i++)
                             sum1 += Math.pow((data[i] - mean),2)/dlen;
                      double result = Math.sqrt(sum1);
                      return result;
              }

       public void getFileInfo() {

                      DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy
HH:mm:ss");
                      Date d = new Date();
                      System.out.println("WEKA Analysis for Entropy Research");
                      System.out.print("\nDate and time for this run is ");
                      System.out.println(dateFormat.format(d));
                      fileRead fr = new fileRead();

                      //Read the configuration file if it hasn't been done already
                      if (!fr.fileRead()) System.exit (1);
                      // Get the entropy values for use and print out.
                      ET = fr.getEType();
                      alpha = fr.getAlpha();
                      m = fr.getM();
                      r = fr.getR();
                      D = fr.getD();
                      System.out.println("Using " + fr.getETName() + " Entropy.");
                      if (ET == 1 || ET == 2)
                         System.out.println("The alpha term is set to " + alpha);
                      if (ET == 3 || ET == 4) {
```

```
                    System.out.println("Window size (m)              = " + m);
                    System.out.println("% of standard deviation (r) = " + r);
                }
                System.out.println("Debugger is "+ D);
            }
}
```

Appendix F: Listing of EntropyFileInfo.txt

Below is a listing of the configuration file for an analysis using the Tsallis entropy calculation. The "#" symbol at the beginning of the line indicates a comment and the line is ignored. This file is changed for each type of entropy calculation or when parameter settings are made.

```
# The Entropy Type (etype) is designated in the
# Entropy Information file defined as variable
# in this class as: "fileName".
#
# The entropy indicator number is as follows:
#       * etype = 0 - Shannon
#       * etype = 1 - Rényi
#       * etype = 2 - Tsallis
#       * etype = 3 - Approximate
#       * etype = 4 - Sample
#
# The alpha term is used for Rényi and Tsallis
# entropy calculations. The work by Lima et al. (2012)
# determined the following values were best in
# research:
#    Rényi alpha   = 0.5
#    Tsallis alpha = 1.2
#
etype =  2
alpha = 1.2
#alpha = 0.5
m = 2
r = 0.2
#D = false
```

Appendix G: Listing of Modified build.xml File for Eclipse

Below is the modified file used by Eclipse to compile the Java code and install the current weka.jar file for use.

```
<project name="weka" default="compile" basedir=".">
<!--
  ============================================================================
   Ant build file for weka. Tested with ant 1.6.5 and Junit 3.8.2. Requires
   JavaMail and the java activation framework for mailing unit test results.

   Type ant -projecthelp for targets and descriptions.
   Assumes weka and tests (if unit testing) are in the same directory.
   Build file can reside and be executed from either inside weka or the
   directory containing weka.

   Modified to only build the components needed for the weka.jar file to
   Support the research.

   Author: Frank Acker October 2014.


   $Revision: 7185 $
  ============================================================================
-->

  <!-- set global properties for this build -->
  <property name="src" value="/media/sf_nova/weka/weka-src/weka-
src/src/main/java"/>
  <property name="src-test" value="/media/sf_nova/weka/weka-src/weka-
src/src/test/java"/>
  <property name="lib" value="/media/sf_nova/weka/weka-src/weka-src/lib" />
  <property name="regression_tests_root" value="src/test/resources/wekarefs"/>
  <property name="build" value="/media/sf_nova/workspace/weka/build"/>
  <property name="dist"  value="/media/sf_nova/workspace/weka/dist"/>
  <property name="doc"  value="doc"/>
  <property name="reports"  value="reports"/>
  <property name="javac_max_memory" value="4096m"/>
  <property name="run_tests_fail" value="true"/>
  <property name="headless" value="false"/>
  <property name="macdistrib" value="osx-distrib"/>
  <property name="debug" value="on" />

  <target name="init_all">
    <!-- Create the time stamp -->
    <tstamp/>
  </target>

   <!-- general classpath definition, incl. CLASSPATH env. variable,
   // but jars in lib directory have precedence over the CLASSPATH variable -->
  <path id="project.class.path">
    <fileset dir="${lib}">
      <include name="*.jar"/>
      <include name="*.zip"/>
    </fileset>
    <pathelement location="${build}/classes"/>
    <pathelement location="${build}/testcases"/>
    <pathelement path="${java.class.path}" />
  </path>
```

```
<!--
 ============================================================================
 Compilation and documentation making stuff
 ============================================================================
-->

  <target name="init_compile" depends="init_all">
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}/classes"/>
  </target>

  <!-- Compile the java code from ${src}weka into ${build}/classes -->
  <target name="compile" depends="init_compile"
   description="Compile weka and deposit class files in build/classes">
    <javac srcdir="${src}"
      fork="yes" memoryMaximumSize="${javac_max_memory}"
      destdir="${build}/classes"
      optimize="${optimization}"
      debug="${debug}"
      deprecation="${deprecation}"
      source="1.4" target="1.4">

      <classpath refid="project.class.path" />
    </javac>
    <copy todir="${build}/classes" >
       <fileset dir="${src}">
         <include name="weka/**/*.gif"/>
         <include name="weka/**/*.jpeg"/>
         <include name="weka/**/*.jpg"/>
         <include name="weka/**/*.props"/>
         <include name="weka/**/*.txt"/>
         <include name="weka/**/DatabaseUtils.props.*"/>
         <include name="weka/gui/beans/README*"/>
       </fileset>
    </copy>
    <rmic base="${build}/classes"
       classname="weka.experiment.RemoteEngine"/>
  </target>

<!--
  ============================================================================
  Release making stuff
  ============================================================================
-->

  <target name = "init_dist" depends="init_all">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}"/>
  </target>

  <!-- Put everything in ${path_modifier}${build}/classes into the weka.jar
file -->
  <target name="exejar" depends="compile, init_dist"
   description="Create an executable jar file in ./dist">
    <jar jarfile="${dist}/weka.jar"
      basedir="${build}/classes">
     <manifest>
       <attribute name="Main-Class" value="weka.gui.GUIChooser"/>
     </manifest>
    </jar>
  </target>
```

```
  <!-- Put all .java, and .props files into ${path_modifier}${dist}/weka-
src.jar-->
  <target name="srcjar" depends="init_dist, init_all"
   description="Create a jar file containing weka source in ./dist">
    <!-- jar up the source -->
    <jar jarfile="${dist}/weka-src.jar"
      basedir=".">
      <include name="*.xml"/>
      <include name="src/**/*.gif"/>
      <include name="src/**/*.java"/>
      <include name="src/**/*.jpeg"/>
      <include name="src/**/*.jpg"/>
      <include name="src/**/*.props"/>
      <include name="src/**/*.txt"/>
      <include name="src/**/*.xml"/>
      <include name="src/**/*.cost"/>
      <include name="src/**/*.arff"/>
      <include name="lib/**/*.jar"/>
      <include name="src/**/DatabaseUtils.props.*"/>
      <include name="src/**/weka/gui/beans/README*"/>
    </jar>
  </target>

  <!-- make a jar file containing just the stuff needed for running a remote
experiment server -->
  <target name="remotejar" depends="compile, init_dist"
   description="Create a jar file containing classes for remote experiments
in ./dist">
     <jar jarfile="${dist}/remoteEngine.jar"
      basedir="${build}/classes"

includes="weka/experiment/*_*.class,weka/experiment/RemoteEngine*.class,weka/ex
periment/Compute.class,weka/experiment/Task.class,weka/experiment/TaskStatusInf
o.class,weka/core/Queue*.class"/>
     <copy todir="${dist}" >
       <fileset dir="${src}/weka/experiment">
          <include name="remote.policy"/>
          <include name="remote.policy.example"/>
       </fileset>
    </copy>
    <jar jarfile="${dist}/remoteExperimentServer.jar"
     basedir="${dist}"
     includes="remoteEngine.jar,remote.policy,remote.policy.example"/>
     <delete file="${dist}/remoteEngine.jar"/>
     <delete file="${dist}/remote.policy"/>
     <delete file="${dist}/remote.policy.example"/>
   </target>


  <!-- Writes $release version number to weka/core/version.txt -->
  <target name="set_version">
    <echo message="${release}" file="${src}/weka/core/version.txt"/>
    <echo message="${release}" file="${build}/classes/weka/core/version.txt"/>
  </target>

  <!-- Make a release -->
  <target name="release" depends="set_version, exejar, remotejar, srcjar"
   description="Make a release in ${release}. Run with -Drelease=&lt;number of
release (eg. 3-4-1)&gt;.">
    <!-- copy the docs to dist/docs -->

    <copy todir="weka-${release}/weka-${release}/doc" >
       <fileset dir="${doc}"/>
```

```
    </copy>
    <copy todir="weka-${release}/weka-${release}">
       <fileset dir="${dist}"/>
    </copy>
    <copy todir="weka-${release}/weka-${release}/data">
       <fileset dir="../wekadocs/data"/>
    </copy>
    <copy todir="weka-${release}/weka-${release}">
       <fileset dir="../wekadocs">
         <include name="README*"/>
        <include name="*.pdf"/>
        <include name="COPYING"/>
         <include name="documentation.*"/>
         <include name="weka.gif"/>
         <include name="weka.ico"/>
       </fileset>
    </copy>
    <zip destfile="weka-${release}.zip"
     basedir="weka-${release}"/>
  </target>
</project>
```

Appendix H: Explanation of Linux commands in selecting attack lines

In generating the data files from the KDD CUP 99 data to mimic the data used by Lima et al. (2012), a series of Linux commands are used and pipelined together to produce the needed results as in the following entry.

```
$ cat kdd.data.csv | grep back | shuf –n 1026 >>DoS.csv
```

In the example above, a description of the Linux commands is as follows:

Command	Description
cat	List the contents of the given to standard out.
grep	This command looks for the given character string, in this case “back”, in each line. If it is found, the line is written to standard out.
shuf	This command reads from standard input and outputs the results in random order. The “-n 1026” options indicates to output 1,026 lines. It is similar to the “sort –R” command but runs much faster.
>>DoS.csv	This is a redirection of standard output to concatenate the results to the file DoS.csv. If DoS.csv does not exist, it will be created.

Appendix I: Listing of Linux shell script to generate KDD CUP 99 files

The listing for the Linus shell script to generate the files for use in reproducing the Lima et al. (2012) results is as follows:

```
echo `date`  Dos.arff
>Dos.arff
cat kddcup.data.csv |grep ,back.$|shuf -n 1026 >>DoS.arff
cat kddcup.data.csv |grep ,land.$|shuf -n 11 >>DoS.arff
cat kddcup.data.csv |grep ,neptune.$|shuf -n 10401 >>DoS.arff
cat kddcup.data.csv |grep ,pod.$|shuf -n 69 >>DoS.arff
cat kddcup.data.csv |grep ,smurf.$|shuf -n 7669 >>DoS.arff
cat kddcup.data.csv |grep ,teardrop.$|shuf -n 15 >>DoS.arff
cat kddcup.data.csv |grep ,normal.$|shuf -n 2573 >>DoS.arff

echo `date`  Probe.arff
>Probe.arff
cat kddcup.data.csv |grep ,ipsweep.$|shuf -n 586 >>Probe.arff
cat kddcup.data.csv |grep ,nmap.$|shuf -n 151 >>Probe.arff
cat kddcup.data.csv |grep ,portsweep.$|shuf -n 155 >>Probe.arff
cat kddcup.data.csv |grep ,satan.$|shuf -n 16 >>Probe.arff
cat kddcup.data.csv |grep ,normal.$|shuf -n 1704 >>Probe.arff

echo `date`  R2L.arff
>R2L.arff
cat kddcup.data.csv |grep ,ftp_write.$|shuf -n 5 >>R2L.arff
cat kddcup.data.csv |grep ,guess_passwd.$|shuf -n 53 >>R2L.arff
cat kddcup.data.csv |grep ,imap.$|shuf -n 11 >>R2L.arff
cat kddcup.data.csv |grep ,multihop.$|shuf -n 7 >>R2L.arff
cat kddcup.data.csv |grep ,phf.$|shuf -n 4 >>R2L.arff
cat kddcup.data.csv |grep ,spy.$|shuf -n 2 >>R2L.arff
cat kddcup.data.csv |grep ,warezclient.$|shuf -n 60 >>R2L.arff
cat kddcup.data.csv |grep ,warezmaster.$|shuf -n 20 >>R2L.arff
cat kddcup.data.csv |grep ,normal.$|shuf -n 1934 >>R2L.arff

echo `date`  U2R.arff
>U2R.arff
cat kddcup.data.csv |grep ,loadmodule.$|shuf -n 9 >>U2R.arff
cat kddcup.data.csv |grep ,buffer_overflow.$|shuf -n 21 >>U2R.arff
cat kddcup.data.csv |grep ,perl.$|shuf -n 3 >>U2R.arff
cat kddcup.data.csv |grep ,rootkit.$|shuf -n 7 >>U2R.arff
cat kddcup.data.csv |grep ,normal.$|shuf -n 1676 >>U2R.arff

echo `date`  Dos.arff
cat Dos.arff|cut -f42 -d,|sort|uniq -c
echo `date`  Probe.arff
cat Probe.arff|cut -f42 -d,|sort|uniq -c
echo `date`  R2L.arff
```

Appendix J: Listing of the KDD CUP 99 features

The table below is the KDD CUP 99 data set features and its definition in the arff file from the KDD CUP 99 web site (KDD Cup 1999 Data, 1999).

Number	Feature Name	Feature Type or values
1	duration	continuous.
2	protocol_type	symbolic.
3	service	symbolic.
4	flag	symbolic.
5	src_bytes	continuous.
6	dst_bytes	continuous.
7	land	symbolic.
8	wrong_fragment	continuous.
9	urgent	continuous.
10	hot	continuous.
11	num_failed_logins	continuous.
12	logged_in	symbolic.
13	num_compromised	continuous.
14	root_shell	continuous.
15	su_attempted	continuous.
16	num_root	continuous.
17	num_file_creations	continuous.
18	num_shells	continuous.
19	num_access_files	continuous.
20	num_outbound_cmds	continuous.
21	is_host_login	symbolic.
22	is_guest_login	symbolic.
23	count	continuous.
24	srv_count	continuous.
25	serror_rate	continuous.
26	srv_serror_rate	continuous.
27	rerror_rate	continuous.
28	srv_rerror_rate	continuous.
29	same_srv_rate	continuous.
30	diff_srv_rate	continuous.
31	srv_diff_host_rate	continuous.
32	dst_host_count	continuous.
33	dst_host_srv_count	continuous.
34	dst_host_same_srv_rate	continuous.
35	dst_host_diff_srv_rate	continuous.
36	dst_host_same_src_port_rate	continuous.
37	dst_host_srv_diff_host_rate	continuous.
38	dst_host_serror_rate	continuous.
39	dst_host_srv_serror_rate	continuous.
40	dst_host_rerror_rate	continuous.
41	dst_host_srv_rerror_rate	continuous.
42	class	back,buffer_overflow,ftp_write,guess_passwd,imap,ipsweep,land,loadmodule,multihop,neptune,nmap,normal,perl,phf,pod,portsweep,rootkit,satan,smurf,spy,teardrop,warezclient,warezmaster.

Appendix K: Partial listing of DoS.arff file

The following is a partial listing of the DoS.arff file. It shows the formatting requirements as defined at the WEKA website (WEKA 3, n.d.). The other arff files used in the analysis each follow this format. At the end of this listing, the data continues to the complete length of the data.

```
@relation "DoS data to reproduce Lima (2012) results"

@attribute duration numeric
@attribute protocol_type {tcp,icmp,udp}
@attribute service {aol,http_8001,http,smtp,finger,domain,domain_u,auth,telnet,
ftp,eco_i,ntp_u,ecr_i,other,private,pop_3,ftp_data,rje,time,mtp,link,remote_job
,gopher,ssh,name,whois,login,imap4,daytime,ctf,nntp,shell,IRC,nnsp,harvest,http
_443,http_2784,exec,printer,efs,courier,uucp,klogin,kshell,echo,discard,systat,
supdup,iso_tsap,hostnames,csnet_ns,pop_2,sunrpc,uucp_path,netbios_ns,netbios_ss
n,netbios_dgm,sql_net,vmnet,bgp,Z39_50,ldap,netstat,urh_i,X11,urp_i,pm_dump,tft
p_u,tim_i,red_i}
@attribute flag {SF,S1,REJ,S2,S0,S3,RSTO,RSTR,RSTOS0,OTH,SH}
@attribute src_bytes numeric
@attribute dst_bytes numeric
@attribute land {0,1}
@attribute wrong_fragment numeric
@attribute urgent numeric
@attribute hot numeric
@attribute num_failed_logins numeric
@attribute logged_in {0,1}
@attribute num_compromised numeric
@attribute root_shell numeric
@attribute su_attempted numeric
@attribute num_root numeric
@attribute num_file_creations numeric
@attribute num_shells numeric
@attribute num_access_files numeric
@attribute num_outbound_cmds numeric
@attribute is_host_login {0,1}
@attribute is_guest_login {0,1}
@attribute count numeric
@attribute srv_count numeric
@attribute serror_rate numeric
@attribute srv_serror_rate numeric
@attribute rerror_rate numeric
@attribute srv_rerror_rate numeric
@attribute same_srv_rate numeric
@attribute diff_srv_rate numeric
@attribute srv_diff_host_rate numeric
@attribute dst_host_count numeric
@attribute dst_host_srv_count numeric
@attribute dst_host_same_srv_rate numeric
@attribute dst_host_diff_srv_rate numeric
@attribute dst_host_same_src_port_rate numeric
@attribute dst_host_srv_diff_host_rate numeric
@attribute dst_host_serror_rate numeric
@attribute dst_host_srv_serror_rate numeric
@attribute dst_host_rerror_rate numeric
@attribute dst_host_srv_rerror_rate numeric
@attribute class {normal.,back.,land.,neptune.,pod.,smurf.,teardrop.}
```

```
@data
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,32,32,1.00,0.00,0.03,0.00,0.03,0.03,0.03,0.03,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,228,228,1.00,0.00,0.00,0.00,0.00,0.00,0.06,0.06,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,178,178,1.00,0.00,0.01,0.00,0.00,0.00,0.01,0.01,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.03,0.03,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.05,0.05,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.05,0.05,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.04,0.04,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.05,0.05,back.
7,tcp,http,RSTR,20440,1460,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.33,0
.33,1.00,0.00,0.00,26,26,1.00,0.00,0.04,0.00,0.00,0.00,0.35,0.35,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,3,0.00,0.00,0.00,0.3
3,1.00,0.00,0.67,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.03,0.03,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.05,0.05,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.01,0.01,0.04,0.04,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,172,172,1.00,0.00,0.01,0.00,0.00,0.00,0.01,0.01,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,164,164,1.00,0.00,0.01,0.00,0.00,0.00,0.01,0.01,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,5,7,0.00,0.00,0.00,0.2
9,1.00,0.00,0.43,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.05,0.05,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,5,6,0.00,0.00,0.00,0.1
7,1.00,0.00,0.33,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.04,0.04,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.01,0.01,0.04,0.04,back.
0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,255,255,1.00,0.00,0.00,0.00,0.00,0.00,0.03,0.03,back.
5,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.0
0,1.00,0.00,0.00,31,31,1.00,0.00,0.03,0.00,0.00,0.00,0.35,0.35,back.
```

Appendix L: Listings of KDD CUP 99 feature extraction shell script and files

The following is a listing of the shell script to extract the features used in the J48 classification tree results file for the KDD CUP 99 data.

```
TEMP1=attemp.txt
TEMP2=attlist.txt
TEMP3=attnums.txt
TEMP4=TEMP3$$
TEMP5=TEMP4$$
A=/media/sf_nova/data/KDD/attributes_names
f=$1
cat $f|sed -e "s/^/ /g" >$TEMP4
firstpipe=`grep -n "^ |" $TEMP4|head -1|cut -f1 -d:`
echo firstpipe = $firstpipe
starthere=`expr $firstpipe - 1`
echo starthere = $starthere
lastpipe=`grep -n "^ |" $TEMP4|tail -1|cut -f1 -d:`
echo lastpipe = $lastpipe
tail -n+$starthere $TEMP4|head -1>$TEMP1
grep -n "^ |" $TEMP4>>$TEMP1
>$TEMP2
>$TEMP3
atnum=0
for i in `cat $A`
do
   atnum=`expr $atnum + 1`
   c=`grep " $i" $TEMP1|wc -l`
   echo $atnum $i $c
   if
      [ $c -gt 0 ]
   then
      echo $atnum $i>>$TEMP2
      echo -n "$atnum " >>$TEMP3
   fi
done
echo >>$TEMP3
cat $TEMP3|sed -e "s/ /, /g"|sed -e "s/, $//" >$TEMP4
cat $TEMP4>$TEMP3
rm $TEMP4
nl $TEMP2
cat $TEMP3
rm -rf $TEMP1 $TEMP2 $TEMP3
```

(Continued on next page)

(Appendix L continued)

The following is a list of the feature names used by the shell script to parse the J48 results file from the KDD CUP 99 data.

```
duration
protocol_type
service
flag
src_bytes
dst_bytes
land
wrong_fragment
urgent
hot
num_failed_logins
logged_in
num_compromised
root_shell
su_attempted
num_root
num_file_creations
num_shells
num_access_files
num_outbound_cmds
is_host_login
is_guest_login
count
srv_count
serror_rate
srv_serror_rate
rerror_rate
srv_rerror_rate
same_srv_rate
diff_srv_rate
srv_diff_host_rate
dst_host_count
dst_host_srv_count
dst_host_same_srv_rate
dst_host_diff_srv_rate
dst_host_same_src_port_rate
dst_host_srv_diff_host_rate
dst_host_serror_rate
dst_host_srv_serror_rate
dst_host_rerror_rate
dst_host_srv_rerror_rate
attack
```

Appendix M: Listings of SRI feature extraction shell script and files

The following is a listing of the shell script to extract the features used in the J48 classification tree results file for the SRI data.

```
TEMP1=attemp.txt
TEMP2=attlist.txt
TEMP3=attnums.txt
TEMP4=TEMP3$$
TEMP5=TEMP4$$
A=/media/sf_nova/data/SRI/bin/SRIattributes_names
f=$1
cat $f|sed -e "s/^/ /g" >$TEMP4
firstpipe=`grep -n "^ |" $TEMP4|head -1|cut -f1 -d:`
echo firstpipe = $firstpipe
starthere=`expr $firstpipe - 1`
echo starthere = $starthere
lastpipe=`grep -n "^ |" $TEMP4|tail -1|cut -f1 -d:`
echo lastpipe = $lastpipe
tail -n+$starthere $TEMP4|head -1>$TEMP1
grep -n "^ |" $TEMP4>>$TEMP1
>$TEMP2
>$TEMP3
atnum=0
for i in `cat $A`
do
   atnum=`expr $atnum + 1`
   c=`grep " $i" $TEMP1|wc -l`
   echo $atnum $i $c
   if
      [ $c -gt 0 ]
   then
      echo $atnum $i>>$TEMP2
      echo -n "$atnum " >>$TEMP3
   fi
done
echo >>$TEMP3
cat $TEMP3|sed -e "s/ /, /g"|sed -e "s/, $//" >$TEMP4
cat $TEMP4>$TEMP3
rm $TEMP4
nl $TEMP2
cat $TEMP3
rm -rf $TEMP1 $TEMP2 $TEMP3
```

(Continued on next page)

The following is a list of the feature names used by the shell script to parse the J48 results file from the SRI data.

```
date
index
timemms
frPort
toPort
flags
seqRange
ack
win
pktLength
SWB
SWBn
smb
rrq
warn
nop
val
ecr
enum
priority
service
Mesg
```

Appendix N: Listing of the J48 Classification Results

The following is the output from WEKA with the KDD CUP 99 data using the J48 classification algorithm with Shannon entropy. There is one of these files for each run of the data and entropy combinations.

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     DoS data to reproduce Lima (2012) results
Instances:    21764
Attributes:   42
              duration
              protocol_type
              service
              flag
              src_bytes
              dst_bytes
              land
              wrong_fragment
              urgent
              hot
              num_failed_logins
              logged_in
              num_compromised
              root_shell
              su_attempted
              num_root
              num_file_creations
              num_shells
              num_access_files
              num_outbound_cmds
              is_host_login
              is_guest_login
              count
              srv_count
              serror_rate
              srv_serror_rate
              rerror_rate
              srv_rerror_rate
              same_srv_rate
              diff_srv_rate
              srv_diff_host_rate
              dst_host_count
              dst_host_srv_count
              dst_host_same_srv_rate
              dst_host_diff_srv_rate
              dst_host_same_src_port_rate
              dst_host_srv_diff_host_rate
              dst_host_serror_rate
              dst_host_srv_serror_rate
              dst_host_rerror_rate
              dst_host_srv_rerror_rate
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===
```

```
J48 pruned tree
------------------

same_srv_rate <= 0.48
|   src_bytes <= 14: neptune. (10354.0)
|   src_bytes > 14: normal. (28.0/1.0)
same_srv_rate > 0.48
|   src_bytes <= 20309
|   |   serror_rate <= 0.59
|   |   |   wrong_fragment <= 0
|   |   |   |   protocol_type = tcp: normal. (2015.0/1.0)
|   |   |   |   protocol_type = icmp
|   |   |   |   |   src_bytes <= 373: normal. (23.0)
|   |   |   |   |   src_bytes > 373: smurf. (7670.0/1.0)
|   |   |   |   protocol_type = udp: normal. (502.0)
|   |   |   wrong_fragment > 0
|   |   |   |   protocol_type = tcp: pod. (0.0)
|   |   |   |   protocol_type = icmp: pod. (68.0)
|   |   |   |   protocol_type = udp: teardrop. (14.0)
|   |   serror_rate > 0.59
|   |   |   land = 0: neptune. (47.0)
|   |   |   land = 1: land. (11.0)
|   src_bytes > 20309
|   |   service = aol: back. (0.0)
|   |   service = http_8001: back. (0.0)
|   |   service = http: back. (1025.0)
|   |   service = smtp: back. (0.0)
|   |   service = finger: back. (0.0)
|   |   service = domain: back. (0.0)
|   |   service = domain_u: back. (0.0)
|   |   service = auth: back. (0.0)
|   |   service = telnet: back. (0.0)
|   |   service = ftp: back. (0.0)
|   |   service = eco_i: back. (0.0)
|   |   service = ntp_u: back. (0.0)
|   |   service = ecr_i: back. (0.0)
|   |   service = other: back. (0.0)
|   |   service = private: back. (0.0)
|   |   service = pop_3: back. (0.0)
|   |   service = ftp_data: normal. (7.0)
|   |   service = rje: back. (0.0)
|   |   service = time: back. (0.0)
|   |   service = mtp: back. (0.0)
|   |   service = link: back. (0.0)
|   |   service = remote_job: back. (0.0)
|   |   service = gopher: back. (0.0)
|   |   service = ssh: back. (0.0)
|   |   service = name: back. (0.0)
|   |   service = whois: back. (0.0)
|   |   service = login: back. (0.0)
|   |   service = imap4: back. (0.0)
|   |   service = daytime: back. (0.0)
|   |   service = ctf: back. (0.0)
|   |   service = nntp: back. (0.0)
|   |   service = shell: back. (0.0)
|   |   service = IRC: back. (0.0)
|   |   service = nnsp: back. (0.0)
|   |   service = harvest: back. (0.0)
|   |   service = http_443: back. (0.0)
|   |   service = http_2784: back. (0.0)
|   |   service = exec: back. (0.0)
|   |   service = printer: back. (0.0)
```

```
|   |   service = efs: back. (0.0)
|   |   service = courier: back. (0.0)
|   |   service = uucp: back. (0.0)
|   |   service = klogin: back. (0.0)
|   |   service = kshell: back. (0.0)
|   |   service = echo: back. (0.0)
|   |   service = discard: back. (0.0)
|   |   service = systat: back. (0.0)
|   |   service = supdup: back. (0.0)
|   |   service = iso_tsap: back. (0.0)
|   |   service = hostnames: back. (0.0)
|   |   service = csnet_ns: back. (0.0)
|   |   service = pop_2: back. (0.0)
|   |   service = sunrpc: back. (0.0)
|   |   service = uucp_path: back. (0.0)
|   |   service = netbios_ns: back. (0.0)
|   |   service = netbios_ssn: back. (0.0)
|   |   service = netbios_dgm: back. (0.0)
|   |   service = sql_net: back. (0.0)
|   |   service = vmnet: back. (0.0)
|   |   service = bgp: back. (0.0)
|   |   service = Z39_50: back. (0.0)
|   |   service = ldap: back. (0.0)
|   |   service = netstat: back. (0.0)
|   |   service = urh_i: back. (0.0)
|   |   service = X11: back. (0.0)
|   |   service = urp_i: back. (0.0)
|   |   service = pm_dump: back. (0.0)
|   |   service = tftp_u: back. (0.0)
|   |   service = tim_i: back. (0.0)
|   |   service = red_i: back. (0.0)

Number of Leaves  :       81

Size of the tree :  91


Time taken to build model: 0.63 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       21754               99.9541 %
Incorrectly Classified Instances        10                0.0459 %
Kappa statistic                          0.9993
Mean absolute error                      0.0002
Root mean squared error                  0.0115
Relative absolute error                  0.1098 %
Root relative squared error              3.8271 %
Total Number of Instances            21764

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.999     0          0.997     0.999     0.998    normal.
  0.999     0          1         0.999     1        back.
  1         0          1         1         1        land.
  1         0          1         1         1        neptune.
  0.986     0          1         0.986     0.993    pod.
  0.999     0          1         0.999     1        smurf.
  0.933     0          1         0.933     0.966    teardrop.

=== Confusion Matrix ===
```

```
   a     b     c     d     e     f     g   <-- classified as
2571     0     0     1     0     1     0 |     a = normal.
   1  1025     0     0     0     0     0 |     b = back.
   0     0    11     0     0     0     0 |     c = land.
   0     0     0 10401     0     0     0 |     d = neptune.
   1     0     0     0    68     0     0 |     e = pod.
   5     0     0     0     0  7664     0 |     f = smurf.
   1     0     0     0     0     0    14 |     g = teardrop.
```

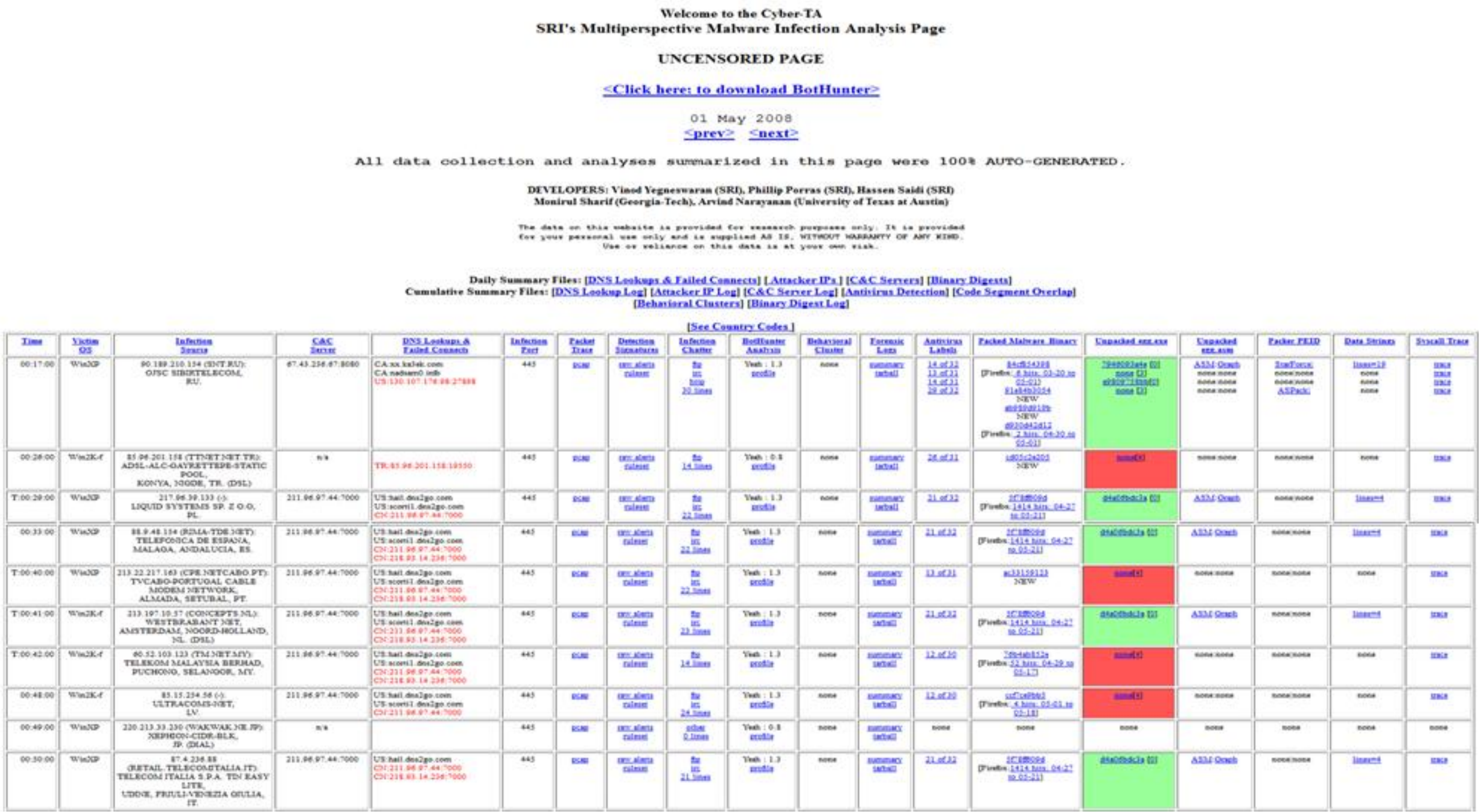
Appendix O: Listing of the feature extraction shell results

Below is the output of the shell script showing the features used in the J48 Classification tree for the KDD CUP 99 DoS attack category using Shannon entropy. The last line of the output is the feature numbers used.

```
1 duration 0
2 protocol_type 6
3 service 70
4 flag 0
5 src_bytes 6
6 dst_bytes 0
7 land 2
8 wrong_fragment 2
9 urgent 0
10 hot 0
11 num_failed_logins 0
12 logged_in 0
13 num_compromised 0
14 root_shell 0
15 su_attempted 0
16 num_root 0
17 num_file_creations 0
18 num_shells 0
19 num_access_files 0
20 num_outbound_cmds 0
21 is_host_login 0
22 is_guest_login 0
23 count 0
24 srv_count 0
25 serror_rate 2
26 srv_serror_rate 0
27 rerror_rate 0
28 srv_rerror_rate 0
29 same_srv_rate 1
30 diff_srv_rate 0
31 srv_diff_host_rate 0
32 dst_host_count 0
33 dst_host_srv_count 0
34 dst_host_same_srv_rate 0
35 dst_host_diff_srv_rate 0
36 dst_host_same_src_port_rate 0
37 dst_host_srv_diff_host_rate 0
38 dst_host_serror_rate 0
39 dst_host_srv_serror_rate 0
40 dst_host_rerror_rate 0
41 dst_host_srv_rerror_rate 0
42 attack 0
     1 2 protocol_type
     2 3 service
     3 5 src_bytes
     4 7 land
     5 8 wrong_fragment
     6 25 serror_rate
     7 29 same_srv_rate
2, 3, 5, 7, 8, 25, 29
```

Appendix P: Partial view of Malware Infection Analysis Page

This image below shows the page from the SRI's Multiperspective Malware Infection Analysis Page. It consists of multiple columns and links to other files for down load.



Appendix Q: Partial listing of file.sh

```
# head -40 file.sh
wget http://www.cyber-ta.org/releases/malware/2008-05-
01-analysis/ARCHIVE/90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/90.189.210.154_130.107.176.98_10.2.32.213.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts_botHu
nter.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/84cf85439891727b7c6d6e32f2caca7e/84cf8543989172
7b7c6d6e32f2caca7e.virus-labels
wget http://www.cyber-
ta.org/releases/malware/SOURCES/91e84b30547650f710f220117e031029/91e84b30547650
f710f220117e031029.virus-labels
wget http://www.cyber-
ta.org/releases/malware/SOURCES/ab989d919b6d0eb454a24f5ace298dc0/ab989d919b6d0e
b454a24f5ace298dc0.virus-labels
wget http://www.cyber-
ta.org/releases/malware/SOURCES/d930d42d1283f036888801c27f486285/d930d42d1283f0
36888801c27f486285.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/217.96.39.133_130.107.251.229_10.2.32.216.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts_botHu
nter.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/5f78ff609da4fc5e699ccf4cbac77bc1/5f78ff609da4fc
5e699ccf4cbac77bc1.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/88.9.48.154_130.107.215.192_10.2.32.212.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts_botHunt
er.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/5f78ff609da4fc5e699ccf4cbac77bc1/5f78ff609da4fc
5e699ccf4cbac77bc1.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.22.217.163_130.107.192.209_10.2.32.216.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz.alerts_botH
unter.txt
wget
http://www.cyber-ta.org/releases/malware/SOURCES/ac331591236cd22abd082d1b9ab488
e2/ac331591236cd22abd082d1b9ab488e2.virus-labels
```

```
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.197.10.57_130.107.208.13_10.2.32.207.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz.alerts_botHun
ter.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/5f78ff609da4fc5e699ccf4cbac77bc1/5f78ff609da4fc
5e699ccf4cbac77bc1.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/60.52.103.123_130.107.245.17_10.2.32.205.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz.alerts_botHun
ter.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/76b4ab852ec50e9b1a959dd8139a41f5/76b4ab852ec50e
9b1a959dd8139a41f5.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz.alerts
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/85.15.254.56_130.107.209.212_10.2.32.201.rules
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz.alerts_botHun
ter.txt
wget http://www.cyber-
ta.org/releases/malware/SOURCES/ccf7ce9bb50a0861e755df41dce9528d/ccf7ce9bb50a08
61e755df41dce9528d.virus-labels
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/220.213.33.230_130.107.167.170_10.2.32.212.pcap.gz
wget http://www.cyber-ta.org/releases/malware/2008-05-01-
analysis/ARCHIVE/220.213.33.230_130.107.167.170_10.2.32.212.pcap.gz.alerts
```

Appendix R: Partial list of downloaded files

The following is a partial list of downloaded files from the Linux "ls" command.

```
213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz
213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz.alerts
213.197.10.57_130.107.208.13_10.2.32.207.pcap.gz.alerts_botHunter.txt
213.197.10.57_130.107.208.13_10.2.32.207.rules
213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz
213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz.alerts
213.22.217.163_130.107.192.209_10.2.32.216.pcap.gz.alerts_botHunter.txt
213.22.217.163_130.107.192.209_10.2.32.216.rules
217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz
217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts
217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts_botHunter.txt
217.96.39.133_130.107.251.229_10.2.32.216.rules
220.213.33.230_130.107.167.170_10.2.32.212.pcap.gz
220.213.33.230_130.107.167.170_10.2.32.212.pcap.gz.alerts
220.213.33.230_130.107.167.170_10.2.32.212.rules
5f78ff609da4fc5e699ccf4cbac77bc1.virus-labels
5f78ff609da4fc5e699ccf4cbac77bc1.virus-labels.1
5f78ff609da4fc5e699ccf4cbac77bc1.virus-labels.2
60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz
60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz.alerts
60.52.103.123_130.107.245.17_10.2.32.205.pcap.gz.alerts_botHunter.txt
60.52.103.123_130.107.245.17_10.2.32.205.rules
76b4ab852ec50e9b1a959dd8139a41f5.virus-labels
84cf85439891727b7c6d6e32f2caca7e.virus-labels
85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz
85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz.alerts
85.15.254.56_130.107.209.212_10.2.32.201.pcap.gz.alerts_botHunter.txt
85.15.254.56_130.107.209.212_10.2.32.201.rules
88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz
88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts
88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts_botHunter.txt
88.9.48.154_130.107.215.192_10.2.32.212.rules
90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz
90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts
90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts_botHunter.txt
90.189.210.154_130.107.176.98_10.2.32.213.rules
91e84b30547650f710f220117e031029.virus-labels
ab989d919b6d0eb454a24f5ace298dc0.virus-labels
ac331591236cd22abd082d1b9ab488e2.virus-labels
ccf7ce9bb50a0861e755df41dce9528d.virus-labels
d930d42d1283f036888801c27f486285.virus-labels
```

Many more files are downloaded per the commands in the "files.sh" file.

Appendix S: Listing of associations.sh script

The following is a listing of the script that generates the associations to improve the naming conventions for the downloaded files.

```
# cat associate.sh
# associate.sh
# this script looks through files.sh and
# creates an association for each pcap.gz
# file. The contents contain:
#    pcap.gz filename
#
Date=20080501
Seq=0
for f in `cat file.sh|cut -f8 -d\/`
do
   p=`echo $f|grep "pcap.gz$"|wc -l`
   if
      [ $p -eq 1 ]
   then
      Seq=`expr $Seq + 1`
   fi
   OutFile=$Date-$Seq
   F=$OutFile-$f
   A=$OutFile-associations.file
   echo $f ==> $F
   if
      [ $p -eq 1 ]
   then
      #Seq=`expr $Seq + 1`
      echo Pcap and Other associated files >$A
   fi
   echo $F >>$A
   cat $f >$F
done
```

Appendix T: Partial List of files prepended with association names

Below is a partial list of the files for the date of 20080501 with the date and association numbers prepended to the file names.

```
20080501-1-84cf85439891727b7c6d6e32f2caca7e.virus-labels
20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap
20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts
20080501-1-
90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts_botHunter.txt
20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts.E3
20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap.tcpdump
20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.rules
20080501-1-91e84b30547650f710f220117e031029.virus-labels
20080501-1-ab989d919b6d0eb454a24f5ace298dc0.virus-labels
20080501-1-associations.file
20080501-1-.AV
20080501-1-d930d42d1283f036888801c27f486285.virus-labels
20080501-2-85.96.201.158_130.107.212.30_10.2.32.201.pcap
20080501-2-85.96.201.158_130.107.212.30_10.2.32.201.pcap.gz.alerts
20080501-2-
85.96.201.158_130.107.212.30_10.2.32.201.pcap.gz.alerts_botHunter.txt
20080501-2-85.96.201.158_130.107.212.30_10.2.32.201.pcap.gz.alerts.E3
20080501-2-85.96.201.158_130.107.212.30_10.2.32.201.pcap.tcpdump
20080501-2-85.96.201.158_130.107.212.30_10.2.32.201.rules
20080501-2-associations.file
20080501-2-.AV
20080501-2-cd05c2e205bc9a84ad14e188d17eadd4.virus-labels
20080501-3-217.96.39.133_130.107.251.229_10.2.32.216.pcap
20080501-3-217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts
20080501-3-
217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts_botHunter.txt
20080501-3-217.96.39.133_130.107.251.229_10.2.32.216.pcap.gz.alerts.E3
20080501-3-217.96.39.133_130.107.251.229_10.2.32.216.pcap.tcpdump
20080501-3-217.96.39.133_130.107.251.229_10.2.32.216.rules
20080501-3-5f78ff609da4fc5e699ccf4cbac77bc1.virus-labels
20080501-3-associations.file
20080501-3-.AV
20080501-4-5f78ff609da4fc5e699ccf4cbac77bc1.virus-labels
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.pcap
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts_botHunter.txt
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.pcap.gz.alerts.E3
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.pcap.tcpdump
20080501-4-88.9.48.154_130.107.215.192_10.2.32.212.rules
20080501-4-associations.file
20080501-4-.AV
```

Appendix U: Listing of an alerts, rules, BotHunter reports, and virus-labels files

The following is an alerts file.

```
$ cat 20080501-158-201.250.57.61_130.107.136.236_10.2.32.214.pcap.gz.alerts
05/01-20:02:15.451167  [**] [1:3000006:99] E3[rb]  BotHunter MALWARE executable
upload [**] [Classification: Misc activity] [Priority: 3] {TCP}
201.250.57.61:4915 -> 130.107.136.236:445
05/01-20:02:15.478900  [**] [1:299998:1] E2[rb] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1] {TCP}
201.250.57.61:4915 -> 130.107.136.236:445
05/01-20:02:15.478900  [**] [1:21390:5] E2[rb] REGISTERED FREE SHELLCODE x86
inc ebx NOOP [**] [Classification: Executable code was detected] [Priority: 1]
{TCP} 201.250.57.61:4915 -> 130.107.136.236:445
05/01-20:02:15.504635  [**] [1:299998:1] E2[rb] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1] {TCP}
201.250.57.61:4915 -> 130.107.136.236:445
05/01-20:02:15.504635  [**] [1:21390:5] E2[rb] REGISTERED FREE SHELLCODE x86
inc ebx NOOP [**] [Classification: Executable code was detected] [Priority: 1]
{TCP} 201.250.57.61:4915 -> 130.107.136.236:445
05/01-20:02:17.855834  [**] [1:2000427:9] E3[rb] ET POLICY PE EXE Install
Windows file download [**] [Classification: Misc activity] [Priority: 3] {TCP}
201.250.57.61:1156 -> 130.107.136.236:1033
05/01-20:03:08.315644  [**] [1:2404005:1142] E4[rb] ET DROP Known Bot C&C
Server Traffic (group 6)  [**] [Classification: A Network Trojan was detected]
[Priority: 1] {TCP} 130.107.136.236:1034 -> 211.96.97.44:7000
05/01-20:04:42.444922  [**] [1:2000352:6] E6[rb] ET ATTACK RESPONSE IRC - dns
request on non-std port [**] [Classification: Potential Corporate Privacy
Violation] [Priority: 1] {TCP} 130.107.136.236:1034 -> 211.96.97.44:7000
```

The following is a rules file

```
$ cat 20080501-158-201.250.57.61_130.107.136.236_10.2.32.214.rules
alert tcp $EXTERNAL_NET any -> $HOME_NET 445 (msg:"E3[rb]  BotHunter MALWARE
executable upload"; flow:established,to_server; content:"ftp"; content: "echo";
content: ".exe"; nocase; classtype: misc-activity; sid:3000006; rev:99; )
```

The following is a botHunter.txt file which reports the BotHunter findings.

```
cat 20080501-158-
201.250.57.61_130.107.136.236_10.2.32.214.pcap.gz.alerts_botHunter.txt
Score:            1.8 (>= 0.8)
Infected Target:  130.107.136.236
Infector List:    201.250.57.61
Egg Source List:  201.250.57.61
C & C List:       211.96.97.44 (3)
Peer Coord. List: <unobserved>
Resource List:    <unobserved>
Observed Start:   05/01/2008 20:02:15.000 PDT
Report End:       05/01/2008 20:02:15.504 PDT
Gen. Time:        05/01/2008 20:04:44.289 PDT

INBOUND SCAN
    <unobserved>

EXPLOIT
    201.250.57.61 (6) (20:02:15.000 PDT-20:02:15.504 PDT)
       event=1:1390 (2) {tcp} E2[rb] REGISTERED FREE SHELLCODE x86 inc ebx NOOP
          2: 445<-4915 (20:02:15.478 PDT-20:02:15.504 PDT)
       -------------------------
```

```
       event=1:2001944 {tcp} E2[rb] BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill
ASN1 exploit attempt
          445<-4915 (20:02:15.478 PDT)
       -------------------------
       event=1:3003 {tcp} E2[rb] NETBIOS SMB-DS Session Setup NTMLSSP unicode
asn1 overflow attempt
          445<-4915 (20:02:15.000 PDT)
       -------------------------
       event=1:99998 (2) {tcp} E2[rb] SHELLCODE x86 inc ebx NOOP
          2: 445<-4915 (20:02:15.478 PDT-20:02:15.504 PDT)

EXPLOIT (slade)
    <unobserved>

EGG DOWNLOAD
    201.250.57.61 (2) (20:02:15.451 PDT)
       event=1:2001684 {tcp} E3[rb] BLEEDING-EDGE Malware Windows executable
sent from remote host, Win32
          1033<-1156 (20:02:17.855 PDT)
       -------------------------
       event=1:3000006 {tcp} E3[rb]  BotHunter MALWARE executable upload
          445<-4915 (20:02:15.451 PDT)

C and C TRAFFIC
    211.96.97.44 (3) (20:04:41.993 PDT)
       event=1:2000345 {tcp} E4[rb] BLEEDING-EDGE ATTACK RESPONSE IRC - Nick
change on non-std port
          1034->7000 (20:04:41.993 PDT)
       -------------------------
       event=1:2002024 {tcp} E4[rb] BLEEDING-EDGE TROJAN IRC NICK command
          1034->7000 (20:04:41.993 PDT)
       -------------------------
       event=1:2002025 {tcp} E4[rb] BLEEDING-EDGE TROJAN IRC JOIN command
          1034->7000 (20:04:42.223 PDT)

PEER COORDINATION
    <unobserved>

OUTBOUND SCAN
    46.113.10.222 (20:04:44.289 PDT)
       event=1:2001569 {tcp} E5[rb] BLEEDING-EDGE Behavioral Unusual Port 445
traffic, Potential Scan or Infection
          1046->445 (20:04:44.289 PDT)

    201.250.57.61 (20:04:44.289 PDT)
       event=555:5555005 {tcp} E5[sc] scade detected scanning of 6 IPs (fail
ratio=0:0/6):
          0->0 (20:04:44.289 PDT)

ATTACK PREP
    <unobserved>

DECLARE BOT
    <unobserved>

tcpslice 1209697335.000 1209697335.505 inputFile.tcpd | tcpdump -r - -w
outputFile.tcpd 'host 130.107.136.236'

============================== SEPARATOR ================================
```

The following is virus-labels file..

```
$ cat 20080501-158-7e28dac8de2cdb7f5f03766ff6500063.virus-labels
Antivirus Detection Summary: file 7e28dac8de2cdb7f5f03766ff6500063

        1: AhnLab-V3              found [Win32/Kolab.worm.200441]
        2: AntiVir                found [TR/Crypt.XPACK.Gen]
        3: Authentium             found nothing
        4: Avast                  found nothing
        5: AVG                    found nothing
        6: BitDefender            found [Packer.PrivateExeProtector.A]
        7: CAT-QuickHeal          found [I-Worm.Kolab.re]
        8: ClamAV                 found nothing
        9: DrWeb                  found [Win32.IRC.Bot]
       10: eSafe                  found [Suspicious File]
       11: eTrust-Vet             found [Win32/ForBot.VC]
       12: Ewido                  found nothing
       13: F-Prot                 found nothing
       14: F-Secure               found [Net-Worm.Win32.Kolab.qw]
       15: FileAdvisor            found nothing
       16: Fortinet               found [W32/Kolab.QW!worm.im]
       17: Ikarus                 found [Packer.PrivateExeProtector.A]
       18: Kaspersky              found [Net-Worm.Win32.Kolab.qw]
       19: McAfee                 found nothing
       20: Microsoft              found nothing
       21: NOD32v2                found [Win32/Kolab.QW]
       22: Norman                 found [W32/Smalltroj.DYQU]
       23: Panda                  found nothing
       24: Prevx1                 found [WORM.VARIANT!WORM]
       25: Rising                 found nothing
       26: Sophos                 found [Mal/Generic-A]
       27: Sunbelt                found nothing
       28: Symantec               found [W32.Spybot.Worm]
       29: TheHacker              found nothing
       30: VBA32                  found [Net-Worm.Win32.Kolab.qw]
       31: VirusBuster            found nothing
       32: Webwasher-Gateway      found [Trojan.Crypt.XPACK.Gen]


CREDITS:  Antivirus malware test results are from submissions to
www.virustotal.com.
```

Appendix V: Listing of bf.sh script and a portion of results

The following is a listing of the bf.sh script used to extract pertinent data from the existing files in each date directory.

```
# cat bf.sh
for j in `ls *alerts|grep "-"|cut -f2 -d-|sort -n`
do
   E=`ls 20??????-$j-*alerts`
   T=`ls 20??????-$j-*tcpdump`
   R=`ls 20??????-$j-*rules`
   echo "++++++++++++++++++++++++++++++++++"
   echo "++++++++++++++++++++++++++++++++++"
   echo E=$E
   echo T=$T
   echo R=$R

   for i in `grep E3 $E|cut -f2-3 -d:|cut -f1 -d\ `
   do
      echo Search Term
      echo $i
      echo =====tcpdump entry
      echo -n TP=
      grep $i $T|grep -v 10.2.
      echo =====Alert entry
      echo -n AE=
      grep $i $E
      echo =====Actual Alert
      echo -n AA=
      grep $i $E | cut -f4 -d\]|cut -f1 -d\[
      echo =============
      echo
   done
done
```

Below is a portion of the results of running the bf.sh script for the date of 20080501. The length of the files for each date is 3,229 lines to 8,649 lines long.

```
++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
E=20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap.gz.alerts
T=20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.pcap.tcpdump
R=20080501-1-90.189.210.154_130.107.176.98_10.2.32.213.rules
File Reference
FR=20080501-1
Search Term
ST=17:51.525715
=====tcpdump entry
TP=02:17:51.525715 IP 90.189.210.154.4619 > 130.107.176.98.445: Flags [.], seq
1165743534:1165744974, ack 3737182244, win 64711, length 1440SMB-over-TCP
packet:(raw data
or continuation?)
=====Alert entry
```

```
AE=05/01-00:17:51.525715  [**] [1:3000006:99] E3[rb]  BotHunter MALWARE
executable upload [**] [Classification: Misc activity] [Priority: 3] {TCP}
90.189.210.154:4619 ->
130.107.176.98:445
=====Actual Alert
AA=  BotHunter MALWARE executable upload
==============

File Reference
FR=20080501-1
Search Term
ST=17:54.399343
=====tcpdump entry
TP=02:17:54.399343 IP 90.189.210.154.1596 > 130.107.176.98.1033: Flags [P.],
seq 1196786334:1196786846, ack 3738074074, win 64800, length 512
=====Alert entry
AE=05/01-00:17:54.399343  [**] [1:2001683:3] E3[rb] BLEEDING-EDGE Malware
Windows executable sent from remote host [**] [Priority: 0] {TCP}
90.189.210.154:1596 -> 130.107
.176.98:1033
=====Actual Alert
AA= BLEEDING-EDGE Malware Windows executable sent from remote host
==============

File Reference
FR=20080501-1
Search Term
ST=17:54.399343
=====tcpdump entry
TP=02:17:54.399343 IP 90.189.210.154.1596 > 130.107.176.98.1033: Flags [P.],
seq 1196786334:1196786846, ack 3738074074, win 64800, length 512
=====Alert entry
AE=05/01-00:17:54.399343  [**] [1:2001683:3] E3[rb] BLEEDING-EDGE Malware
Windows executable sent from remote host [**] [Priority: 0] {TCP}
90.189.210.154:1596 -> 130.107
.176.98:1033
=====Actual Alert
AA= BLEEDING-EDGE Malware Windows executable sent from remote host
==============
```

Appendix W: Listing of Features selected from SRI data

Fields used for the feature selection analysis in WEKA.

Field Num	Name	Description	Values
1	Date	Date of data in yyyymmdd	20080501 … 20080510
2	Index	Reference of data file	Numeric
3	Timemms	Time of day of incident in microseconds	Numeric
4	frPort	From port number	Numeric
5	toPort	To port number	Numeric
6	Flags	TCP flags in packet	NA, ACK, flags, PUSH&ACK
7	seqRange	Range of pack numbers	Numeric
8	ack	Byte count in exchange	Numeric
9	win	Bytes count of window size	Numeric
10	pktLength	Byte count of data sent	Numeric
11	SWR	Additional info with length	NA, RRQ, SMB, WARNING
12	SWRn	Indicator if RRQ, SMB, or Warning present	0, 1
13	Smb	Server Message Block message	NA, SMB-over-TCP
14	Rrq	Read Request message	NA, svchost.exe
15	Warn	Warning of packet continuation	NA, Packet continued
16	Nop	Count of NOP in options	Numeric
17	Val	Sender timestamp info	Numeric
18	Ecr	Echo reply timestamp info	Numeric
19	Enum	Alert E indicator	E2, E3
20	Priority	Priority of alert	Numeric
21	Service	Type of protocol used	TCP, UDP
22	aMesg	Alert message indicator	BE1, BE2, BE3, BH1, BH2, BH4, ET1, NB, SH, TFTP

The first three fields are used as references for trace back to the initial data file. Fields 13, 14, and 15 have a value taken from the tcpdump line for the incident. An NA or other value are acceptable for this field as identified in the above table.

Below is an expanded description of the fields:

1. The **date** is that designated by the SRI web site which contains the data.
2. **Index** is a number generated to keep better management of the different files for each day. All the files with the same index are from the same incident. The main files for each incident are the tcpdump file, the alert file, virus-labels, rules, and the associations file.
3. **Timemms** is the time of day in microseconds. The time taken from the tcpdump files for the incident which matches the search term.
4. The **frPort** field is port number contained in the source IP field in the tcpdump file.
5. The **toPort** field is port number contained in the destination IP field in the tcpdump file.
6. **Flags** are the TCP flags in many of the tcpdump entries. In the data used for the analysis, the following flags were used: 1) ACK 2) PUSH&ACK.
7. The **seqRange** feature uses the difference of the values in the seq parameter from the tcpdump output.
8. The **ack** is the number of bytes in the exchange.
9. The **win** is the widow size in bytes.
10. The **pktLength** is the number of bytes in the packet.
11. **SWR** is a feature which indicates which, if any, additional information is appended to the packet length parameter. The possibilities are “RRQ”, “SMB”, or “WARNING”.

12. The **SWRn** feature indicates if a “RRQ”, “SMB”, or “WARNING” was present.
13. The **Smb** is the type of Server Message Block test presented. For the data used, it is only “SMB-over-TCP”.
14. The **Rrq** is the Read Request message. For the data used, it is only “svchost.exe”.
15. The **Warn** id the “Warning of packet continuation”. For the data used, it is only “Packet continued”.
16. The **Nop** is in the options field in the tcpdump output. This parameter contains a count of the number of “nop” entries there are between the brackets in the options field.
17. The **Val** value is a time stamp in the options field. The parameter “TS val” between the brackets is a timestamp from the sender.
18. The **Ecr** value is a time stamp in the options field. The parameter “ecr” between the brackets is an echo reply timestamp from the sender.
19. The **Enum** feature comes from the rules files which is a BotHunter message. This E number is associated with the infection type. For the data used, the values are “E2”, or “E3”.
20. The **Priority** feature is a numeric value in the alerts file .
21. The Service feature is the type of protocol in the tcpdump file. For the data used, the values are “TCP”, or “UDP”.
22. The aMesg feature is a shortened version of the Alert message indicator from the alerts file. For the data used, the values are “BE1”, “BE2”, “BE3”, “BH1”, “BH2”, “BH4”, “ET1”, “NB”, “SH”, or “TFTP”. The table following this list shows the expanded description of each value.

aMesg	Enum	Description	Count
BE1	E2	BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt	4
BE2	E3	BLEEDING-EDGE Malware Windows executable sent from remote host, Win32	2128
BE3	E3	BLEEDING-EDGE VIRUS Sasser Transfer _up.exe	35
BH1	E3	BotHunter HTTP-based .exe Upload on backdoor port	353
BH2	E3	BotHunter MALWARE executable upload	1356
BH4	E3	BotHunter Scrip-based Windows egg download .exe	170
ET1	E3	ET POLICY PE EXE Install Windows file download	150
NB	E2	NETBIOS SMB-DS Session Setup NTMLSSP unicode asn1 overflow attempt	22
SH	E2	SHELLCODE x86 inc ebx NOOP	12
TFTP	E3	TFTP GET .exe from external source	128

Appendix X: Partial Listing of SRI.arff file

```
@relation "SRI data"

@attribute date {20080501,20080502,20080503,20080504,20080505,20080506,20080507
,20080508,20080509,20080510}
@attribute index NUMERIC
@attribute timemms NUMERIC
@attribute frPort NUMERIC
@attribute toPort NUMERIC
@attribute flags {NA,ACK,flags,PUSH&ACK}
@attribute seqRange NUMERIC
@attribute ack NUMERIC
@attribute win NUMERIC
@attribute pktLength NUMERIC
@attribute SWB {NA,RRQ,SMB,WARNING}
@attribute SWBn {0,1}
@attribute smb {NA,SMB-over-TCP}
@attribute rrq {svchost.exe,NA}
@attribute warn {NA,Packet_continued}
@attribute nop NUMERIC
@attribute val NUMERIC
@attribute ecr NUMERIC
@attribute enum {E2,E3}
@attribute priority NUMERIC
@attribute service {TCP,UDP}
@attribute Mesg {BE1,BE2,BE3,BH1,BH2,BH4,ET1,NB,SH,TFTP}

@data
%date,index,timemms,frPort,toPort,flags,seqRange,ack,win,pktLength,SWR,SWRn,smb
,rrq,warn,nop,val,ecr,enum,priority,service,aMesg
20080501,1,8271525715,4619,445,ACK,1440,3737182244,64711,1440,SMB,1,SMB-over-
TCP,NA,NA,0,0,0,E3,3,TCP,BH2
20080501,1,8274399343,1596,1033,PUSH&ACK,512,3738074074,64800,512,NA,0,NA,NA,NA
,0,0,0,E3,0,TCP,BE2
20080501,1,8274399343,1596,1033,PUSH&ACK,512,3738074074,64800,512,NA,0,NA,NA,NA
,0,0,0,E3,0,TCP,BE2
20080501,1,8297720956,1038,80,PUSH&ACK,45,3632878526,17520,45,NA,0,NA,NA,NA,0,0
,0,E3,3,TCP,BH1
20080501,1,8297722080,1039,80,PUSH&ACK,47,851293562,17520,47,NA,0,NA,NA,NA,0,0,
0,E3,3,TCP,BH1
20080501,1,8297723954,1040,80,PUSH&ACK,44,311488835,17520,44,NA,0,NA,NA,NA,0,0,
0,E3,3,TCP,BH1
20080501,2,8762681249,48983,445,ACK,1452,1619850869,64217,1452,SMB,1,SMB-over-
TCP,NA,NA,0,0,0,E3,3,TCP,BH2
20080501,2,8765115020,49214,1028,ACK,1452,1620640693,64240,1452,NA,0,NA,NA,NA,0
,0,0,E3,3,TCP,ET1
20080501,3,8991897517,1630,445,ACK,1460,26535918,64151,1460,SMB,1,SMB-over-
TCP,NA,NA,0,0,0,E3,3,TCP,BH2
20080501,3,8994461706,1705,1033,ACK,1460,27935892,64240,1460,NA,0,NA,NA,NA,0,0,
0,E3,3,TCP,ET1
20080501,4,9206779357,4004,445,ACK,1440,488988015,65446,1440,SMB,1,SMB-over-
TCP,NA,NA,0,0,0,E3,3,TCP,BH2
20080501,4,9209229243,4010,1033,ACK,1440,489801999,65535,1440,NA,0,NA,NA,NA,0,0
,0,E3,3,TCP,ET1
20080501,5,9610941864,3786,445,ACK,1448,64185918,64052,1448,SMB,1,SMB-over-
TCP,NA,NA,2,831416,2662,E3,3,TCP,BH2
20080501,5,9612600730,3820,1033,ACK,1448,64785892,64064,1448,NA,0,NA,NA,NA,2,83
1433,0,E3,3,TCP,ET1
20080501,6,9689209897,3750,445,ACK,1460,3862234534,17431,1460,SMB,1,SMB-over-
TCP,NA,NA,0,0,0,E3,3,TCP,BH2
```

Appendix Y Listing of Java files to read and parse SRI data

This appendix contains the listings of five Java classes use to parse the SRI data and fomat it for usin in WEKA. The order of the listings are:

- util1.java
- util2.java
- util3.java
- util4.java
- dv.java

==========

util1.java

```
import java.io.*;

/*
 * This class is supports the retrieval of the raw SRI
 * malware data from multiple files. The output writes
 * the data into an intermediate file that is read by
 * other applications.
 *
 * Author: Frank Acker, December 2014
 */


public class util1 {

  util2 u2 = new util2();
  util3 u3 = new util3();

  public void po () {
     System.out.println("FR="+dv.FRinfo);
     System.out.println("ST="+dv.STinfo);
     System.out.println("TP="+dv.TPinfo);
     System.out.println("AE="+dv.AEinfo);
     System.out.println("AA="+dv.AAinfo);
  }

  public void nullInfo() {
     dv.FRinfo     = dv.NULL;
     dv.STinfo     = dv.NULL;
     dv.TPinfo     = dv.NULL;
     dv.AEinfo     = dv.NULL;
     dv.AAinfo     = dv.NULL;
     dv.frIP       = "0";
     dv.frPort     = "0";
     dv.toIP       = "0";
     dv.toPort     = "0";
     dv.Flags      = "0";
     dv.seqRange   = 0;
     dv.ackThere   = 0;
```

```
    dv.ackValue   = 0;
    dv.winThere   = 0;
    dv.winValue   = 0;
    dv.pktLength  = 0;
    dv.smbType    = dv.NA;
    dv.rrqType    = dv.NA;
    dv.smbFound   = false;
    dv.warnType   = dv.NA;
    dv.warnFound  = false;
    dv.flagsNum   = 0;
    dv.seqNum     = 0;
    dv.winNum     = 0;
    dv.lengthNum  = 0;
    dv.ackNum     = 0;
    dv.optType    = dv.NA;
    dv.optNum     = 0;
    dv.rrqNum     = 0;
    dv.nopCount   = 0;
    dv.valNum     = -99;
    dv.ecrNum     = -99;
    dv.valValue   = 0;
    dv.ecrValue   = 0;
    dv.attackType = dv.NULL;

    // From AE line
    dv.aeEnum      = dv.NULL; // E3
    dv.aeAMsg      = dv.NULL; // Alert message
    dv.aeAMsgDesig = dv.NULL; // Alert message designator
    dv.aePriority  = 0;    // Alert Priority
    dv.aeService   = dv.NULL; // Alert network service used
  }

  //static String[] stuff = new String[256];
   public long timecalcmms(String stuff) {
      if (stuff.length() == 0) return 0;
      String [] ts = stuff.split(":");
      long hr = 0, min = 0;
      try {
         hr = Long.parseLong(ts[0]) * 3600 * 1000000;
      }catch(NumberFormatException ex){
         System.out.println("OOPS in timecalamms-hr
string="+stuff+"=");
         System.out.println("OOPS in timecalamms-hr
ts[0]="+ts[0]+"=");
      }
      try {
         min = Long.parseLong(ts[1]) * 60 * 1000000;
      //}catch(NumberFormatException ex){
      }catch(ArrayIndexOutOfBoundsException ex){
         System.out.println("OOPS in timecalamms-min
string="+stuff+"=");
         System.out.println("OOPS in timecalamms-min
ts[1]="+ts[1]+"=");
      }
```

```
      Double sec = 0.0;
      try {
         sec = Double.parseDouble(ts[2]) *1000000.0;
      //}catch(NumberFormatException ex){
      }catch(ArrayIndexOutOfBoundsException ex){
         System.out.println("OOPS");
         System.out.println("OOPS in timecalamms-sec
string="+stuff+"=");
         System.out.println("OOPS in timecalamms-sec
ts[2]="+ts[2]+"=");
      }
      Long mmsec = hr + min + sec.longValue();
      return mmsec;

   }

   public void getFlags(String [] TPvar) {
      String s;
      String flagInfo = TPvar[dv.flagsNum+1];
      if (dv.flagsNum > 0) {

         String [] FlagSym = {".", "F","F.","FP","P","P.","R",
                              "R.","S","S."};
         String [] FlagDesc = {"ACK", "Finish","Finish&ACK",
            "FIN&Pish","PUSH","PUSH&ACK","RST","RST&ACK",
            "SYN","SYN&ACK"};
         flagInfo = flagInfo.replace("[","");
         flagInfo = flagInfo.replace("]","");
         flagInfo = flagInfo.replace(",","");

         int i;
         for (i=0; i<FlagSym.length; i++) {
            if (flagInfo.equals(FlagSym[i])) {
               dv.Flags = FlagDesc[i];
               break;
            }
         }
      }

      // Seq
      if (dv.seqNum > 0 )
         dv.seqRange = getSeqRange(TPvar[dv.seqNum+1]);

      // ack
      if (dv.ackNum > 0) {
         dv.ackThere = 1;
         s = TPvar[dv.ackNum+1].replace(",","");
         dv.ackValue = Long.parseLong(s);
      }

      // win
      if (dv.winNum > 0) {
         dv.winThere = 1;
         s = TPvar[dv.winNum+1].replace(",","");
```

```
      dv.winValue = Integer.parseInt(s);
   }

   // options
   if (dv.optNum >0)
      u3.optManage(TPvar);


   // length
   if (dv.lengthNum > 0) {
      s = TPvar[dv.lengthNum+1].replace(",","");
      s = s.replace(":","");
      // check if alphas are butted up to length
      boolean ok = true;
      if (s.contains(dv.aSMB) && ok) {
         u3.lenSplit(s, TPvar, dv.aSMB);
         ok = false;
      }
      if (s.contains(dv.aRRQ) && ok) {
         u3.lenSplit(s, TPvar, dv.aRRQ);
         ok = false;
      }
      if (s.contains(dv.aWARN) && ok) {
         u3.lenSplit(s, TPvar, dv.aWARN);
         ok = false;
      }
      if (ok)
         dv.pktLength = Integer.parseInt(s);
   }
}


public Long getSeqRange(String data) {
   data = data.replace(",","");
   String [] numz = data.split(":");
   if (numz.length >1) {
      long num1 = Long.parseLong(numz[0]);
      long num2 = Long.parseLong(numz[1]);
      return (long) (num2-num1);
   }
   return (long)0;
}

public Boolean allInfo() {
   if (dv.FRinfo.equals(dv.NULL) ||
       dv.STinfo.equals(dv.NULL) ||
       dv.TPinfo.equals(dv.NULL) ||
       dv.AEinfo.equals(dv.NULL) ||
       dv.AAinfo.equals(dv.NULL) ) {
      return false;
   }
   return true;
}
```

```
   // method to sort input string to proper variables
   public void inSort (String input) {
      String c2 = "";

      if (input.length() > 2) {
         c2 = input.substring(0,2);
      }
      if (c2.equals(dv.FR)) { // File Reference
         dv.FRinfo = input;
      }

      if (c2.equals(dv.ST)) { // Search Term
         dv.STinfo = input;
      }

      if (c2.equals(dv.TP)) { // TCP Dump
         dv.TPinfo = input;
      }

      if (c2.equals(dv.AE)) { // Alert Entry
         dv.AEinfo = input;
      }

      if (c2.equals(dv.AA)) { // Actual Alert
         dv.AAinfo = input;
      }

   }
}
```

==========

util2.java

```
import java.io.*;

/*
 * This class is supports the retrieval of the raw SRI
 * malware data from multiple files. The output writes
 * the data into an intermediate file that is read by
 * other applications.
 *
 * Author: Frank Acker, December 2014
 */

public class util2 {

    util3 u3 = new util3();
    util4 u4 = new util4();
    boolean header = true;

   public void pdvFlags() {
```

```
      System.out.println("dv.flagsNum="+dv.flagsNum);
      System.out.println("dv.seqNum="+dv.seqNum);
      System.out.println("dv.winNum="+dv.winNum);
      System.out.println("dv.lengthNum="+dv.lengthNum);
      System.out.println("dv.ackNum="+dv.ackNum);
      System.out.println("dv.optNum="+dv.optNum);
      System.out.println("dv.valNum="+dv.valNum);
      System.out.println("dv.ecrNum="+dv.ecrNum);
      System.out.println("dv.rrqNum="+dv.rrqNum);
   }

   public void setTPNums(String[] TPvar) {
      String t[] = dv.TPinfo.split("=");
      //String[] TPvar = t[1].split(" ");
      int j;
      for (j=0; j<TPvar.length; j++) {
         System.out.println(j + "  " + TPvar[j]);
         if (TPvar[j].equals(dv.aFlags))  dv.flagsNum  = j;
         if (TPvar[j].equals(dv.aSeq))    dv.seqNum    = j;
         if (TPvar[j].equals(dv.aWin))    dv.winNum    = j;
         if (TPvar[j].equals(dv.aLength)) dv.lengthNum = j;
         if (TPvar[j].equals(dv.aAck))    dv.ackNum    = j;
         if (TPvar[j].contains(dv.aOpt))    dv.optNum    = j;
         if (TPvar[j].contains(dv.aVal)) {
            dv.valNum    = j;
            dv.valValue  = u3.justNums(TPvar[dv.valNum + 1]);
         }
         if (TPvar[j].contains(dv.aEcr)) {
            dv.ecrNum    = j;
            dv.ecrValue  = u3.justNums(TPvar[dv.ecrNum + 1]);
         }
         if (TPvar[j].contains(dv.aRRQ)) {
            dv.rrqNum = j;
            dv.rrqType = TPvar[j+1].replace("\"","");
         }
      }
   }

   public void getIPnPort(String [] TPvar) {
      String ipInfo[] = TPvar[2].split("\\.");
      // get the port numbers
      String s=ipInfo[0];
      s=s.concat(".");s=s.concat(ipInfo[1]);s=s.concat(".");
      s=s.concat(ipInfo[2]);s=s.concat(".");s=s.concat(ipInfo[3]);
      dv.frIP=s;
      dv.frPort = ipInfo[4];
      ipInfo = TPvar[4].split("\\.");
      s=ipInfo[0];
      s=s.concat(".");s=s.concat(ipInfo[2]);s=s.concat(".");
      s=s.concat(ipInfo[1]);s=s.concat(".");s=s.concat(ipInfo[3]);
      dv.toIP=s;
      dv.toPort = ipInfo[4].replace(":","");
   }
   public void prtOutVars() {
```

```
   // From TCP Dump line
      System.out.println("+++OUTPUT VALUES+++");
      System.out.println("frIP="+dv.frIP);
      System.out.println("frPort="+dv.frPort);
      System.out.println("toIP="+dv.toIP);
      System.out.println("toPort="+dv.toPort);
      System.out.println("Flags="+dv.Flags);
      System.out.println("seqRange="+dv.seqRange);
      System.out.println("ackThere="+dv.ackThere);
      System.out.println("ackValue="+dv.ackValue);
      System.out.println("winThere="+dv.winThere);
      System.out.println("winValue="+dv.winValue);
      System.out.println("pktLength="+dv.pktLength);
      System.out.println("smbType="+dv.smbType);
      System.out.println("rrqType="+dv.rrqType);
      System.out.println("warnType="+dv.warnType);
      System.out.println("optType="+dv.optType);
      System.out.println("nopCount="+dv.nopCount);
      System.out.println("valValue="+dv.valValue);
      System.out.println("ecrValue="+dv.ecrValue);

      System.out.println("aeEnum="+dv.aeEnum);
      System.out.println("aeAMsg="+dv.aeAMsg);
      System.out.println("aeAMsgDesig="+dv.aeAMsgDesig);
      System.out.println("aePriority="+dv.aePriority);
      System.out.println("aeService"+dv.aeService);

      if (dv.header) u4.csvHeader();
      u4.csvOut();
   }
}
```

==========

util3.java

```
import java.io.*;

/*
 * This class is supports the retrieval of the raw SRI
 * malware data from multiple files. The output writes
 * the data into an intermediate file that is read by
 * other applications.
 *
 * Author: Frank Acker, December 2014
 */

public class util3 {

   public void lenSplit(String s, String [] TPvar, String type) {
      String stLength = dv.NULL;
```

```
      String msg      = dv.NULL;
      int i;
      for (i=1; i<=s.length(); i++) {
         String c = s.substring(i-1,i);
         if (dv.nums.contains(c)) {
            stLength += c;
         } else {
            break;
         }
      }
      int ii;
      for (ii= i; ii<= s.length(); ii++) {
         String c = s.substring(ii-1,ii);
         msg += c;
      }
      s = stLength;

      dv.pktLength = Integer.parseInt(s);
      for (i=dv.lengthNum+2; i<TPvar.length; i++) {
         msg += " ";
         msg += TPvar[i];
      }
System.out.println("type="+type+"   msg="+msg);
      if (type.contains(dv.aSMB))  dv.smbType  = "SMB-over-TCP";
      if (type.contains(dv.aWARN)) dv.warnType = "Packet_continued";
      if (type.contains(dv.aRRQ))  dv.rrqType  = msg;
   }

   public void optManage(String [] TPvar) {

      dv.optType = dv.NULL;
      int i;
      for (i=dv.optNum+1; i<dv.lengthNum; i++) {
         dv.optType += TPvar[i];
         dv.optType += " ";
      }
      dv.optType = dv.optType.replace("[","");
      dv.optType = dv.optType.replace("]","");
      String [] optTemp = dv.optType.split(",");
//System.out.println("Looking for NOP");
      for (i=0; i<optTemp.length; i++) {
         if (optTemp[i].contains(dv.aNop)) dv.nopCount++;
      }

   }

   public int justNums (String s) {
      int i, n;
      String snum = dv.NULL;
      for (i=1; i<=s.length(); i++) {
         String c = s.substring(i-1,i);
         if (dv.nums.contains(c)) {
            snum += c;
         } else {
```

```
            break;
         }
      }
      return(Integer.parseInt(snum));
   }
}
```

==========

util4.java

```
import java.io.*;

/*
 * This class is supports the retrieval of the raw SRI
 * malware data from multiple files and reads
 * the data from the intermediate file and formats it for
 * use in WEKA.
 *
 * Author: Frank Acker, December 2014
 */

public class util4 {

   public void csvHeader() {

      System.out.print("csvHead=");
      System.out.print("frPort");
      System.out.print(","+"toPort");
      System.out.print(","+"flags");
      System.out.print(","+"seqRange");
      System.out.print(","+"ack");
      System.out.print(","+"win");
      System.out.print(","+"pktLength");
      System.out.print(","+"smb");
      System.out.print(","+"rrq");
      System.out.print(","+"warn");
      System.out.print(","+"nop");
      System.out.print(","+"val");
      System.out.print(","+"ecr");
      System.out.print(","+"enum");
      System.out.print(","+"aMesg");
      System.out.print(","+"priority");
      System.out.print(","+"service");

      System.out.println();

      dv.header = false;
   }
   public void csvOut() {

      System.out.print("csvOut=");
      System.out.print(dv.frPort);
```

```
      System.out.print(","+dv.toPort);
      System.out.print(","+dv.Flags);
      System.out.print(","+dv.seqRange);
      System.out.print(","+dv.ackValue);
      System.out.print(","+dv.winValue);
      System.out.print(","+dv.pktLength);
      System.out.print(","+dv.smbType);
      System.out.print(","+dv.rrqType);
      System.out.print(","+dv.warnType);
      System.out.print(","+dv.nopCount);
      System.out.print(","+dv.valValue);
      System.out.print(","+dv.ecrValue);

      System.out.print(","+dv.aeEnum);
      System.out.print(","+dv.aeAMsgDesig);
      System.out.print(","+dv.aePriority);
      System.out.print(","+dv.aeService);

      System.out.println();


   }

   public void aeParse(String aeLine) {

      int i, e;
      // find the " E" char string
      e = aeLine.indexOf(" E");
      // get the E and number after it
      dv.aeEnum = "E";
      dv.aeEnum += aeLine.charAt(e+2);

      dv.aeAMsg = dv.NULL;
      // move ahead 8 spaces. the next should be the
      // beginning of the Alert message
      e +=8;
      char c;
      while ((c = aeLine.charAt(e++)) != '[') {
         dv.aeAMsg += c;
      }
      // get the attack designator
      int msgLen2 = dv.aeAMsg.length()/2;
      String msg2 = dv.aeAMsg.substring(msgLen2/2,msgLen2);

      for (i=0; i<dv.attacks.length; i++) {
         //if (dv.aeAMsg.contains(dv.attacks[i])) {
         if (dv.attacks[i].contains(msg2)) {
            dv.aeAMsgDesig = dv.attackDesig[i];
            break;
         }
      }

      // get the Priority
      e = aeLine.indexOf("Priority");
```

```
      c = aeLine.charAt(e + 10);
      dv.aePriority = Character.getNumericValue(c);

      // get the service type
      e = aeLine.indexOf("{") + 1;
      while ((c = aeLine.charAt(e++)) != '}') {
         dv.aeService += c;
      }

   }
}
```

==========

dv.java

```
import java.io.*;

/* This class contains the variables used by the different
 * methods in this set of java programs
 *
 * Author: Frank Acker, December 2014
 */


public class dv {

   static String NULL = "";
   static String NA = "NA";
   static String EQ = "=";  // Equals sign

   // Triggers from bf files
   static String FR = "FR"; // File Reference
   static String ST = "ST"; // Search Term
   static String TP = "TP"; // Tcpdump entry
   static String AE = "AE"; // Alert entry
   static String AA = "AA"; // Actual Alert

   // Contents of complete lines
   static String FRinfo = NULL; // File Reference info
   static String STinfo = NULL; // Search Term info
   static String TPinfo = NULL; // Tcpdump entry info
   static String AEinfo = NULL; // Alert entry info
   static String AAinfo = NULL; // Actual Alert info

   // Search terms in lines
   static String aFlags  = "Flags";
   static String aSeq    = "seq";
   static String aWin    = "win";
   static String aLength = "length";
   static String aAck    = "ack";
```

```
static String aSMB    = "SMB";
static String aWARN   = "WARNING";
static String aRRQ    = "RRQ";
static String aOpt    = "options";
static String aNop    = "nop";
static String aEcr    = "ecr";
static String aVal    = "val";

// Position markers
static int flagsNum   = 0;
static int  seqNum    = 0;
static int  winNum    = 0;
static int  lengthNum = 0;
static int  ackNum    = 0;
static int  optNum    = 0;
static int  rrqNum    = 0;
static int  valNum    = -99;
static int  ecrNum    = -99;

// Variables used in the output file
// From TCP Dump line
static String frIP     = "0";
static String frPort   = "0";
static String toIP     = "0";
static String toPort   = "0";
static String Flags    = "0";
static long seqRange   = 0;
static int ackThere    = 0;
static long ackValue   = 0;
static int winThere    = 0;
static int winValue    = 0;
static int pktLength   = 0;
static String smbType  = NA;
static String rrqType  = NA;
static String warnType = NA;
static String optType  = NA;
static int nopCount    = 0;
static int valValue    = 0;
static int ecrValue    = 0;
static String attackType = NULL;

// From AE line
static String aeEnum      = NULL; // E3
static String aeAMsg      = NULL; // Alert message
static String aeAMsgDesig = NULL; // Alert message designator
static int    aePriority  = 0;    // Alert Priority
static String aeService   = NULL; // Alert network service used

// Other variables for use
static boolean smbFound = false;
static boolean warnFound = false;
static String nums = "0123456789";
static boolean header = true;
```

```
   static String [] attacks = {
     "BLEEDING-EDGE EXPLOIT MS04-007 Kill-Bill ASN1 exploit attempt
",
     "BLEEDING-EDGE Malware Windows executable sent from remote
host, Win32",
     "BLEEDING-EDGE VIRUS Sasser Transfer _up.exe ",
     "BotHunter HTTP-based .exe Upload on backdoor port",
     "BotHunter MALWARE executable upload",
     "BotHunter Malware Windows executable (PE) sent from remote
host",
     "BotHunter Scrip-based Windows egg download .exe",
     "ET POLICY PE EXE Install Windows file download",
     "ET WORM Sasser Transfer _up.exe ",
     "NETBIOS SMB-DS Session Setup NTMLSSP unicode asn1 overflow
attempt ",
     "SHELLCODE x86 inc ebx NOOP",
     "TFTP GET .exe from external source"};

     static String [] attackDesig = {"BE1","BE2","BE3","BH1","BH2",
          "BH3","BH4","ET1","ET2","NB","SH","TFTP"};

}
```

Appendix Z – Results of Analysis using SRI Malware data

ENTROPY	PARAMS	CC	#FS
Approximate			
	m=1;r=0.01	99.0821	7
	m=1;r=0.1	99.0821	7
	m=1;r=0.2	99.0821	7
	m=1;r=0.3	99.0821	7
	m=1;r=0.4	99.0821	7
	m=1;r=0.5	99.0821	7
	m=1;r=0.6	99.0821	7
	m=1;r=0.7	99.0821	7
	m=1;r=0.8	99.0821	7
	m=1;r=0.9	99.0821	7
	m=1;r=0.99	99.0821	7
	m=2;r=0.01	99.1051	7
	m=2;r=0.1	99.1051	7
	m=2;r=0.2	99.1051	7
	m=2;r=0.3	99.0821	7
	m=2;r=0.4	99.0821	7
	m=2;r=0.5	99.0821	7
	m=2;r=0.6	99.0821	7
	m=2;r=0.7	99.0821	7
	m=2;r=0.8	99.0821	7
	m=2;r=0.9	99.0821	7
	m=2;r=0.99	99.0821	7
	m=3;r=0.01	99.1051	7
	m=3;r=0.1	99.1051	7
	m=3;r=0.2	99.1051	7
	m=3;r=0.3	99.1051	7
	m=3;r=0.4	99.1051	7
	m=3;r=0.5	99.1051	7
	m=3;r=0.6	99.1051	7
	m=3;r=0.7	99.1051	7
	m=3;r=0.8	99.0821	7
	m=3;r=0.9	99.0821	7
	m=3;r=0.99	99.0821	7
	m=4;r=0.01	99.1051	7
	m=4;r=0.1	99.1051	7
	m=4;r=0.2	99.1051	7
	m=4;r=0.3	99.1051	7
	m=4;r=0.4	99.1051	7
	m=4;r=0.5	99.1051	7
	m=4;r=0.6	99.1051	7
	m=4;r=0.7	99.1051	7
	m=4;r=0.8	99.0821	7
	m=4;r=0.9	99.0821	7
	m=4;r=0.99	99.0821	7

ENTROPY	PARAMS	CC	#FS
Approximate			
	m=5;r=0.01	99.1051	7
	m=5;r=0.1	99.1051	7
	m=5;r=0.2	99.1051	7
	m=5;r=0.3	99.1051	7
	m=5;r=0.4	99.1051	7
	m=5;r=0.5	99.1051	7
	m=5;r=0.6	99.1051	7
	m=5;r=0.7	99.1051	7
	m=5;r=0.8	99.1051	7
	m=5;r=0.9	99.1051	7
	m=5;r=0.99	99.1051	7
	m=6;r=0.01	99.1051	7
	m=6;r=0.1	99.1051	7
	m=6;r=0.2	99.1051	7
	m=6;r=0.3	99.1051	7
	m=6;r=0.4	99.1051	7
	m=6;r=0.5	99.1051	7
	m=6;r=0.6	99.1051	7
	m=6;r=0.7	99.1051	7
	m=6;r=0.8	99.1051	7
	m=6;r=0.9	99.1051	7
	m=6;r=0.99	99.1051	7
	m=7;r=0.01	99.1051	7
	m=7;r=0.1	99.1051	7
	m=7;r=0.2	99.1051	7
	m=7;r=0.3	99.1051	7
	m=7;r=0.4	99.1051	7
	m=7;r=0.5	99.1051	7
	m=7;r=0.6	99.1051	7
	m=7;r=0.7	99.1051	7
	m=7;r=0.8	99.1051	7
	m=7;r=0.9	99.1051	7
	m=7;r=0.99	99.1051	7
	m=8;r=0.01	99.1051	7
	m=8;r=0.1	99.1051	7
	m=8;r=0.2	99.1051	7
	m=8;r=0.3	99.1051	7
	m=8;r=0.4	99.1051	7
	m=8;r=0.5	99.1051	7
	m=8;r=0.6	99.1051	7
	m=8;r=0.7	99.1051	7
	m=8;r=0.8	99.1051	7
	m=8;r=0.9	99.1051	7
	m=8;r=0.99	99.1051	7
	m=9;r=0.01	99.1051	7
	m=9;r=0.1	99.1051	7
	m=9;r=0.2	99.1051	7
	m=9;r=0.3	99.1051	7
	m=9;r=0.4	99.1051	7
	m=9;r=0.5	99.1051	7
	m=9;r=0.6	99.1051	7
	m=9;r=0.7	99.1051	7
	m=9;r=0.8	99.1051	7
	m=9;r=0.9	99.1051	7
	m=9;r=0.99	99.1051	7

ENTROPY	PARAMS	CC	#FS
Sample			
	m=1;r=0.01	95.732	7
	m=1;r=0.1	95.8926	8
	m=1;r=0.2	99.2428	6
	m=1;r=0.3	98.6232	10
	m=1;r=0.4	97.2006	9
	m=1;r=0.5	97.7283	9
	m=1;r=0.6	96.5351	9
	m=1;r=0.7	96.8105	10
	m=1;r=0.8	96.6269	10
	m=1;r=0.9	96.4892	9
	m=1;r=0.99	96.581	9
	m=2;r=0.01	93.9881	11
	m=2;r=0.1	93.9192	11
	m=2;r=0.2	95.2501	10
	m=2;r=0.3	96.2827	9
	m=2;r=0.4	95.8697	8
	m=2;r=0.5	95.8697	9
	m=2;r=0.6	95.8008	10
	m=2;r=0.7	95.8467	13
	m=2;r=0.8	95.6631	9
	m=2;r=0.9	95.9615	9
	m=2;r=0.99	95.2731	10
	m=3;r=0.01	93.7357	10
	m=3;r=0.1	93.7357	10
	m=3;r=0.2	93.7357	10
	m=3;r=0.3	93.4144	10
	m=3;r=0.4	93.8045	10
	m=3;r=0.5	93.6668	11
	m=3;r=0.6	93.3685	11
	m=3;r=0.7	93.4374	11
	m=3;r=0.8	93.4144	10
	m=3;r=0.9	94.0799	10
	m=3;r=0.99	93.0932	10
	m=4;r=0.01	93.0014	10
	m=4;r=0.1	93.0014	10
	m=4;r=0.2	93.0014	10
	m=4;r=0.3	93.0014	10
	m=4;r=0.4	93.0014	10
	m=4;r=0.5	93.0014	10
	m=4;r=0.6	93.0014	10
	m=4;r=0.7	93.7816	10
	m=4;r=0.8	93.7816	10
	m=4;r=0.9	93.7816	10
	m=4;r=0.99	94.1028	11

ENTROPY	PARAMS	CC	#FS
Sample			
	m=5;r=0.01	92.9555	10
	m=5;r=0.1	92.9555	10
	m=5;r=0.2	92.9555	10
	m=5;r=0.3	92.9555	10
	m=5;r=0.4	92.726	10
	m=5;r=0.5	92.726	10
	m=5;r=0.6	93.0702	10
	m=5;r=0.7	93.0702	10
	m=5;r=0.8	93.3915	10
	m=5;r=0.9	93.6668	10
	m=5;r=0.99	93.598	10
	m=6;r=0.01	92.1983	10
	m=6;r=0.1	92.1983	10
	m=6;r=0.2	92.1983	10
	m=6;r=0.3	92.1983	10
	m=6;r=0.4	92.1983	10
	m=6;r=0.5	92.1983	10
	m=6;r=0.6	92.1983	10
	m=6;r=0.8	93.0014	10
	m=6;r=0.9	93.0014	10
	m=6;r=0.99	93.0702	10
	m=7;r=0.01	92.1983	10
	m=7;r=0.1	92.1983	10
	m=7;r=0.2	92.1983	10
	m=7;r=0.3	92.1983	10
	m=7;r=0.4	92.1983	10
	m=7;r=0.5	92.0606	10
	m=7;r=0.6	91.877	10
	m=7;r=0.7	92.6801	10
	m=7;r=0.8	92.5425	10
	m=7;r=0.9	92.5425	10
	m=7;r=0.99	92.5425	10
	m=8;r=0.01	92.1983	10
	m=8;r=0.1	92.1983	10
	m=8;r=0.2	92.1983	10
	m=8;r=0.3	92.1983	10
	m=8;r=0.4	92.1983	10
	m=8;r=0.5	92.1983	10
	m=8;r=0.6	92.1983	10
	m=8;r=0.7	92.1983	10
	m=8;r=0.8	92.1983	10
	m=8;r=0.9	92.1983	10
	m=8;r=0.99	92.1983	10
	m=9;r=0.01	92.1983	10
	m=9;r=0.1	92.1983	10
	m=9;r=0.2	92.1983	10
	m=9;r=0.3	92.1983	10
	m=9;r=0.4	92.1983	10
	m=9;r=0.5	92.1983	10
	m=9;r=0.6	92.1983	10
	m=9;r=0.7	92.1983	10
	m=9;r=0.8	92.1983	10
	m=9;r=0.9	92.1983	10
	m=9;r=0.99	92.1983	10

ENTROPY	PARAMS	CC	#FS
Shannon		99.8853	8

ENTROPY	PARAMS	CC	#FS
Tsallis			
	alpha = 1.01	99.9082	7
	alpha = 1.1	99.9082	8
	alpha = 1.2	99.7476	10
	alpha = 1.3	99.7935	10
	alpha = 1.4	99.8853	10
	alpha = 1.5	99.9082	7
	alpha = 1.6	99.8853	10
	alpha = 1.7	99.9082	8
	alpha = 1.8	99.9312	8
	alpha = 1.9	99.9312	6
	alpha = 1.99	99.8623	7

ENTROPY	PARAMS	CC	#FS
Rényi			
	alpha = 0.01	90.8903	7
	alpha = 0.1	90.9133	7
	alpha = 0.2	92.3818	7
	alpha = 0.3	91.7164	7
	alpha = 0.4	91.3722	9
	alpha = 0.5	90.3167	6
	alpha = 0.6	89.7201	6
	alpha = 0.7	88.9399	9
	alpha = 0.8	88.894	6
	alpha = 0.9	87.4484	7
	alpha = 0.99	73.4511	7

References

Alazab, A., Hobbs, M., Abawajy, J., & Alazab, M. (2012). Using feature selection for intrusion detection system. *2012 International Symposium on Communications and Information Technologies*, 296-301.

Alelyani, S., Tang, J., & Liu, H. (2013). Feature Selection for Clustering: A Review. Retrieved March 12, 2013 from

Alvim, M., Andrés, M., & Palamidessi, C. (2010). Probabilistic information flow. *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer* Science, 314-321.

Anderson, J. (1980). Computer security threat monitoring and surveillance. (Technical Report), Washington, PA, James E Anderson Co.

Barbará, D., Couto, J., & Li, Y. (2002). COOLCAT: an entropy-based algorithm for categorical clustering. *Proceedings of the eleventh international conference on Information and knowledge management*, 582-589.

Barot, V., Chauhan, S., & Patel, B. (2014). Feature Selection for Modeling Intrusion Detection. *International Journal of Computer Network and Information Security*, 7, 56-62.

BotHunter (n.d.). BotHunter Central. Retrieved from http://www.bothunter.net/

Bhuyan, M., Bhattacharyya, D., & Kalita, J. (2011). Survey on incremental approaches for network anomaly detection. *International Journal of Communications and Information Security*, 3(3), 226-239.

Bouckaert, R., Frank, E., Hall, M., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. (2010). WEKA---Experiences with a Java Open-Source Project. *The Journal of Machine Learning Research*, 9999, 2533-2541.

CAIDA - The Cooperative Association for Internet Data Analysis. (n.d.). Retrieved August 5, 2014 from http://www.caida.org

Chen, X., Solomon, I., & Chon, K. (2005). Comparison of the use of Approximate entropy and sample entropy: applications to neural respiratory signal. In *Engineering in Medicine and Biology Society, 2005. 27th Annual International Conference of the IEEE-EMBS 2005*, 4212-4215.

Chesnokov, Y. (2008). Approximate and sample entropies complexity. Retrieved April 3, 2014 from http://www.codeproject.com/Articles/27030/Approximate-and-Sample-Entropies-Complexity-Metric

Cheung, S., & Valdes, A. (2009). Malware characterization through alert pattern discovery. *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*.

Chon, K., Scully, C., & Lu, S. (2009). Approximate entropy for all signals. *Engineering in Medicine and Biology Magazine*, 28(6), 18-23.

Denning, D. (1986). An intrusion-detection model. *Software Engineering, IEEE Transactions on*, (2), 222-232.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *Proceedings of the Twelfth International Conference of Machine Learning*, 194-202.

Eclipse (n.d.). Eclipse. Retrieved from www.eclipse.org.

Fares, A., Sharawy, M. I., & Zayed, H. H. (2011). Intrusion Detection: Supervised Machine Learning. *Journal of Computing Science and Engineering*, 5(4), 305-313.

Gallagher, B., & Eliassi-Rad, T. (2008). Classification of http attacks: a study on the ECML/PKDD 2007 discovery challenge. *Center for Advanced Signal and Image Sciences Workshop*.

Gappmair, W. (1999). Claude E. Shannon: The 50th anniversary of information theory. *Communications Magazine*, 37(4), 102-105.

Gheyas, I., & Smith, L. (2010). Feature subset selection in large dimensionality domains. *Pattern Recognition*, 43(1), 5-13.

Guillén, E., Rodriguez, J., Páez, R., & Rodriguez, A. (2012). Detection of Non-Content Based Attacks Using GA with Extended KDD Features. In *Proceedings of the World Congress on Engineering and Computer Science* (1).

Gupta, K., Nath, B., & Kotagiri, R. (2010). Layered approach using conditional random fields for intrusion detection. *IEEE Transactions on Dependable and Secure Computing*, 7(1), 35-49.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1), 10-18.

Harremoës, P. (2006). Interpretations of Rényi entropies and divergences. *Physica A: Statistical Mechanics and its Applications*, *365*(1), 57-62.

Hammer, B., & Villmann, T. (2002). Generalized relevance learning vector quantization. *Neural Networks*, 15(8), 1059-1068.

HTTP DATASET CSIC 2010. (2010). Retrieved February 3, 2013 from http://iec.csic.es/dataset/

Index of releases for malware. (n.d.). Retrieved from http://www.cyber-ta.org/releases/malware/

Johal, R. S., & Tirnakli, U. (2004). Tsallis versus Rényi entropic form for systems with *q*-exponential behaviour: the case of dissipative maps. *Physica A: Statistical Mechanics and its Applications*, 331(3), 487-496.

KDD Cup 1999 Data. (1999). Retrieved March 5, 2012 from http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Kumar, S., & Spafford, E. (1994). An application of pattern matching in intrusion detection. *Purdue University, Technical Report CSD-TR-94-013*.

Lake, D. (2011). Improved entropy rate estimation in physiological data. *Engineering in Medicine and Biology Society*, 1463-1466

Lebowitz, J. (1993). Boltzmann's entropy and time's arrow. *Physics Today, 46*, 32-32.

Lee, K., Gray, A., & Kim, H. (2013). Dependence maps, a dimensionality reduction with dependence distance for high-dimensional data. *Data Mining and Knowledge Discovery*, 26(3), 512-532.

Lee, T., & He, J. (2009). Entropy-based profiling of network traffic for detection of security attack. *TENCON 2009-2009 IEEE Region 10 Conference*, 1-5.

Lima, C., de Assis, F. & de Souza, C. (2012). An empirical investigation of attribute selection techniques based on Shannon, Rényi and Tsallis entropies for network intrusion detection. *American Journal of Intelligent Systems*, 2(5), 111-117.

Liu, C., Liu, C., Shao, P., Li, L., Sun, X., Wang, X., & Liu, F. (2011). Comparison of different threshold values r for Approximate entropy: application to investigate the heart rate variability between heart failure and healthy control groups. *Physiological Measurement*, 32(2), 167.

Liu, C., & Zhao, L. (2011). Using fuzzy measure entropy to improve the stability of traditional entropy measures. *Computing in Cardiology*, 681-684.

Liu, H., & Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4), 491-502.

Manis, G. (2008). Fast computation of approximate entropy. *Computer methods and programs in biomedicine*, 91(1), 48-54.

Nguyen, H., Franke, K., & Petrović, S. (2012). Reliability in a feature-selection process for intrusion detection. *Reliable Knowledge Discovery*, 203-218.

NSL-KDD Data Set (2009). The NSL-KDD Data Set. Retrieved March 25, 2013 from http://nsl.cs.unb.ca/NSL-KDD/

Nychis, G., Sekar, V., Andersen, D., Kim, H., & Zhang, H. (2008). An empirical evaluation of entropy-based traffic anomaly detection. *Proceedings of the 8th ACM SIGCOMM Conference on Internet measurement*, 151-156.

Özçelik, İ., & Brooks, R. (2015). Deceiving entropy based DoS detection. *Computers & Security*, *48*, 234-245.

Om, H., & Kundu, A. (2012, March). A hybrid system for reducing the false alarm rate of anomaly intrusion detection system. *1st International Conference on Recent Advances in Information* Technology, 131-136.

Pan, Y., Wang, Y., Liang, S., & Lee, K. (2011). Fast computation of sample entropy and Approximate entropy in biomedicine. *Computer methods and programs in biomedicine*, 104(3), 382-396.

Pincus, S. (1991). Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, 88(6), 2297-2301.

Pincus, S., & Keefe, D. (1992). Quantification of hormone pulsatility via an Approximate entropy algorithm. *Am J Physiol*, 262(5 Pt 1), E741-E754.

PREDICT - Protected Repository for the Defense of Infrastructure Against Cyber Threats. (n.d.). Retrieved October 24, 2013 from https://www.predict.org/.

Quinlan, J. (1986). Induction of decision trees. *Machine learning*, 1, (1), 81-106.

Richman, J., & Moorman, J. (2000). Physiological time-series analysis using Approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, 278(6), H2039-H2049.

Schafer, J. L., & Graham, J. W. (2002). Missing data: our view of the state of the art. *Psychological methods*, 7(2), 147.

Shannon, C. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27, 379-423.

Sharma, N., & Mukherjee, S. (2012). Layered approach for intrusion detection using naïve Bayes classifier. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 639-644.

Sinai, Y. (2007). Metric entropy of dynamical system. Retrieved March 20, 2013 from http://web.math.princeton.edu/facultypapers/Sinai/MetricEntropy2.pdf

Snort. (n.d.) Welcome to the new Snort.org. Retrieved July 28, 2014 from https://www.snort.org/

Songfeng, Z., Xiaofeng, L., Nanning, Z., & Weipu, X. (2003). Unsupervised clustering based reduced support vector machines. *Acoustics, Speech, and Signal Processing*, II-821 – II-824.

SRI. (n.d.). Intrusion detection. Retrieved July 22, 2014 from http://csl.sri.com/programs/intrusion/

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A detailed analysis of the KDD CUP 99 data set. *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*.

Ubuntu. (n.d.), Ubuntu Operating System. Retrieved from http://www.ubuntu.com

Vasudevan, A., Harshini, E., & Selvakumar, S. (2011). SSENet-2011: A network intrusion detection system dataset and its comparison with KDD CUP 99 dataset. *2011 Second Asian Himalayas International Conference on Internet*, 1-5.

VB. (n.d.). Oracle VM VirtualBox. Retrieved from https://www.virtualbox.org/

Velayutham, C., & Thangavel, K. (2012, March). A novel entropy based unsupervised feature selection algorithm using rough set theory. *Advances in Engineering, Science and Management (ICAESM)*, 156-161

VirusTotal (n.d.). VirusTotal. Retrieved from https://www.virustotal.com.

WEKA 3. (n.d.) Weka 3: Data mining software in Java. Retrieved from http://www.cs.waikato.ac.nz/ml/weka/

Win8. (n.d.) Microsoft Windows 8 Operating System. Retrieved from http://windows.microsoft.com/en-us/windows/home

Witten, I., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Yentes, J., Hunt, N., Schmid, K., Kaipust, J., McGrath, D., & Stergiou, N. (2013). The Appropriate use of Approximate entropy and sample entropy with short data sets. *Annals of biomedical engineering*, 41(2), 349-365.

Yurtkan, K., & Demirel, H. (2013). Entropy-based feature selection for improved 3D facial expression recognition. *Signal, Image and Video Processing*, 1-11.

Zuech, R., Khoshgoftaar, T. M., & Wald, R. (2015). Intrusion detection and Big Heterogeneous Data: a Survey. *Journal of Big Data*, 2(1), 1-41.

Zhai, J., Li, N., & Zhai, M. (2011). The condensed fuzzy k-nearest neighbor rule based on sample fuzzy entropy. *2011 International Conference on Machine Learning and Cybernetics*, 1, 282-286.