2017

Security Analytics: Using Deep Learning to Detect Cyber Attacks

Glenn M. Lambert II
University of North Florida

SECURITY ANALYTICS: USING DEEP LEARNING TO DETECT CYBER ATTACKS

by

Glenn Monroe Lambert II

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computing and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

Spring, 2017

This thesis titled “Security Analytics: Using Deep Learning to Detect Cyber Attacks” submitted by Glenn Monroe Lambert II in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

_____ _____

Dr. Sherif A. Elfayoumy
Thesis Advisor and Committee Chairperson

_____ _____

Dr. Ching-Hua Chuan

_____ _____

Dr. Swapnoneel Roy

Accepted for the School of Computing:

_____ _____

Dr. Sherif A. Elfayoumy
Director, School of Computing

Accepted for the College of Computing, Engineering, and Construction:

_____ _____

Dr. Mark Tumeo
Dean of the College of Computing, Engineering, and Construction

Accepted for the University:

_____ _____

Dr. John Kantner
Dean of the Graduate School

This work is dedicated to my wonderful wife, Charlotte Atkinson, for all her support and encouragement throughout this endeavor.

ACKNOWLEDGEMENT

I would first like to thank my thesis advisor Dr. Sherif Elfayoumy, Director of the School of Computing at the University of North Florida, for devoting a significant amount of his time to this research effort. His expert guidance was paramount to keeping me focused to the end.

I would also like to thank Clay Maddox, Assistant Director of Information Security at the University of North Florida, and Geoffrey Whittaker, IT Security Engineer, for their valuable security expertise and their time required for the enormous effort to obtain and cleanse the security logs for this research.

I would also like to thank Glenn Ford, Senior Application Security Analyst at Availity for lending his security expertise at a moment's notice.

CONTENTS

TABLES

FIGURES

EQUATIONS

ABSTRACT

Security attacks are becoming more prevalent as cyber attackers exploit system vulnerabilities for financial gain. The resulting loss of revenue and reputation can have deleterious effects on governments and businesses alike. Signature recognition and anomaly detection are the most common security detection techniques in use today. These techniques provide a strong defense. However, they fall short of detecting complicated or sophisticated attacks. Recent literature suggests using security analytics to differentiate between normal and malicious user activities.

The goal of this research is to develop a repeatable process to detect cyber attacks that is fast, accurate, comprehensive, and scalable. A model was developed and evaluated using several production log files provided by the University of North Florida Information Technology Security department. This model uses security analytics to complement existing security controls to detect suspicious user activity occurring in real time by applying machine learning algorithms to multiple heterogeneous server-side log files. The process is linearly scalable and comprehensive; as such it can be applied to any enterprise environment. The process is composed of three steps. The first step is data collection and transformation which involves identifying the source log files and selecting a feature set from those files. The resulting feature set is then transformed into a time series dataset using a sliding time window representation. Each instance of the dataset is labeled as *green*, *yellow*, or *red* using three different unsupervised learning

methods, one of which is Partitioning around Medoids (PAM). The final step uses Deep Learning to train and evaluate the model that will be used for detecting abnormal or suspicious activities. Experiments using datasets of varying sizes of time granularity resulted in a very high accuracy and performance. The time required to train and test the model was surprisingly fast even for large datasets. This is the first research paper that develops a model to detect cyber attacks using security analytics; hence this research builds a foundation on which to expand upon for future research in this subject area.

Chapter 1

INTRODUCTION

1.1 Overview

Security attacks are becoming more prevalent as cyber attackers exploit system vulnerabilities for financial gain. Theft of Intellectual Property and destruction of infrastructure are additional motives resulting from industrial espionage and Nation State actors, respectively [Sood13]. Nation State actors employ the most skilled attackers with the ability to launch targeted and coordinated attacks. Sony, Stuxnet, and Anthem are recent examples of targeted attacks.

The time from a security breach to detection is measured in days [Muncaster15]. Cyber attackers are aware of existing security controls and are continually improving their attacks. To make matters worse, cyber attackers have a wide range of tools available which allow them to bypass traditional security mechanisms. Zero day exploits, Malware Infection Frameworks (MIF), Rootkits, and Browser Exploit Packs (BEP) can be readily purchased on an underground market. Attackers can also purchase personal information and compromised domains in order to launch additional attacks [Sood13]. A security breach is inevitable. Early detection and mitigation are the best defense to surviving an attack.

Security professionals employ prevention and detection techniques to reduce the risk of a security breach. In "Applying Data Mining Techniques to Intrusion Detection," Ng. et al. define a security breach as "any action the system owner deems unauthorized" [Ng15]. Prevention techniques focus on making attacks more difficult. Some examples of prevention techniques include: establishing a good security policy, applying recent security updates, avoiding default configurations, and establishing an effective user security education program [Garcia12]. All information security policies should adhere to the three principles of the CIA triad which are Confidentiality, Integrity, and Availability. Confidentiality is a set of rules that limits access to information. Integrity is assurance that information is trustworthy and accurate. Availability refers to the ensuring that all authorized users are able to access information systems.

Detection techniques fall into two categories, attack recognition or signature-based detection, and anomaly-based detection. Traditional security solutions such as Firewalls, Intrusion Detection Systems (IDS), and virus scanners use a signature-based approach. The signature-based approach compares a hash of the payload to a database of known malicious signatures [Razzaq14]. Signature based detection techniques monitor network traffic for ongoing attacks but fall short of detecting zero-day attacks or a variant of an existing attack, also known as a mimicry attack [Garcia12]. These techniques provide a strong defense against known attacks. However, they are by no means a sufficient guard against skilled attackers who use the latest attack methods and exploits. Hence, they can easily bypass any security controls in place [Ye05, Sood13].

Anomaly detection detects abnormal events, including those that are not yet encountered. In other words, anything abnormal is considered an attack [Ng15]. Anomaly detection requires a model of normal system behavior. False positives can occur when normal activities are detected to be irregular [Garcia12].

The Cyber Research Alliance (CRA) identified the application of Big Data Analytics to cyber security as one of the top six priorities for future cyber security research and development [Kott14]. Big Data Analytics (BDA) is the aggregating and correlating of a broad range of heterogeneous data from multiple sources, and has the potential to detect cyber threats within actionable time frames with minimal or no human intervention [Kott14]. Security Analytics is the application of Big Data Analytics to cyber security. Security Analytics is a new trend in the industry, and interest is expected to gain momentum quickly. Finding appropriate algorithms required to locate hidden patterns in huge amounts of data is just one of the several challenges that must be overcome. Incomplete and noisy data are additional factors that must be considered. Finally, the massive scale of enterprise security data available poses the greatest challenge to a successful Security Analytics implementation [Kott14]. Security Analytics differs from traditional approaches by separating what is normal from what is abnormal. In other words, the focus is on the action or user activity instead of the payload content or signature [Mahmood13].

1.2 Problem Statement

The goal of this research is to develop a repeatable process to detect cyber attacks that is fast, accurate, and scalable. The process should evaluate multiple data sources in order to gain a comprehensive picture of user activity across multiple systems. User activity patterns undergo normal fluctuations throughout the day, and often those patterns differ from patterns that occur on weekends. The model is expected to differentiate between normal fluctuations and abnormal user activities. A deep learning algorithm is used to train a neural network to detect suspicious user activities.

This research is very closely related to one class of digital forensics which focuses on discovering evidence of criminal activity inadvertently left in log files on computer systems by hackers [Garfinkel16]. This research differs from digital forensics in that it focuses on finding malicious activity patterns and identifying criminal activity while it is occurring.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Background

Most computer systems record events in log files [Abad03]. The type and structure of log files vary widely by system and platform. For example, weblogs are produced by web servers running Apache or Internet Information Server (IIS) among others. Operating systems, firewalls, and Intrusion Detection Systems (IDS) record event information in log files. Applications also record user activities in log files [Abad03]. Any activities performed during a security breach will most likely result in log entries being recorded in one or more log files. These attacks cannot be identified by a single log entry occurrence, but instead, can be identified through a series of entries spanning several minutes [Abad03]. The amount of data logged per system can be more than several thousand events per minute. Additionally, these files are typically distributed across the network. In order to process and analyze the log data, they must be integrated and stored in a central location. Integrating highly heterogeneous data from multiple sources requires a massive centralized data repository [Kott13]. Such a data repository should meet the complexity requirements as defined by Big Data.

2.1.1 Machine Learning

Big Data is defined by three characteristics: volume, velocity, and variety. Volume is the size of the data stored and is measured in terabytes, petabytes, or Exabytes. Velocity is the rate at which data is generated. Variety refers to the types of data, such as structured, semi-structured, or non-structured [Mahmood13]. Structured data is data that typically reside in a database or data warehouse. Examples of unstructured data are documents, images, text messages, and tweets. Log data is considered semi-structured. In some cases, log data contains key-value pairs or is stored in CSV format. Adam Jacobs, in "The Pathologies of Big Data," defines Big Data as "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time" [Jacobs09]. Big Data presents new challenges to searching and processing of data. These new challenges require new techniques and methods, such as data mining or Big Data analytics.

Big data analytics employs data mining techniques for extracting actionable insights from data to make intelligent business decisions [Apte03]. Commonly, the first step in Big Data analytics is Extract Transform Load (ETL) [Mahmood13]. This is a pre-processing step that transforms data into a format that is compatible with data mining algorithms [Mahmood13]. The processing or analysis step applies an algorithm, such as clustering, to the transformed data. Finally, the results are displayed on a dashboard or in a report [Apte03]. Data mining is defined as the application of machine learning methods to large datasets [Alpaydin14].

Machine learning is a subfield of artificial intelligence that allows a computer to learn using sample data without being programmed to anticipate every possible situation [Alpaydin14]. The two most common types of machine learning are supervised and unsupervised learning. Supervised learning is used when a dataset of labeled instances is available. Supervised learning is used to solve classification problems. The goal of supervised learning is to train the computer to learn to predict a value or classify an input instance accurately. Unsupervised learning is used when a labeled dataset is not available. Clustering is an unsupervised learning technique which results in grouping similar instances in clusters. Clustering is used to discover patterns in data. In some cases, clustering is performed to classify an unlabeled dataset and using the resulting classified dataset for supervised learning [Alpaydin14].

Artificial Neural Network (ANN), proposed fifty years ago, is a collection of supervised learning models inspired by the human brain. A simple neural network or multi-layer perceptron is composed of three layers; an input layer, a hidden layer, and an output layer. Each layer is composed of neurons, which are interconnected to all the neurons in the next layer. The network is trained by adjusting the weights of the neurons to minimize the error between the output neuron and the desired result [Edwards15]. A neural network (Figure 1) using a large number of hidden layers is referred to as a deep neural network and training is referred to as deep learning.

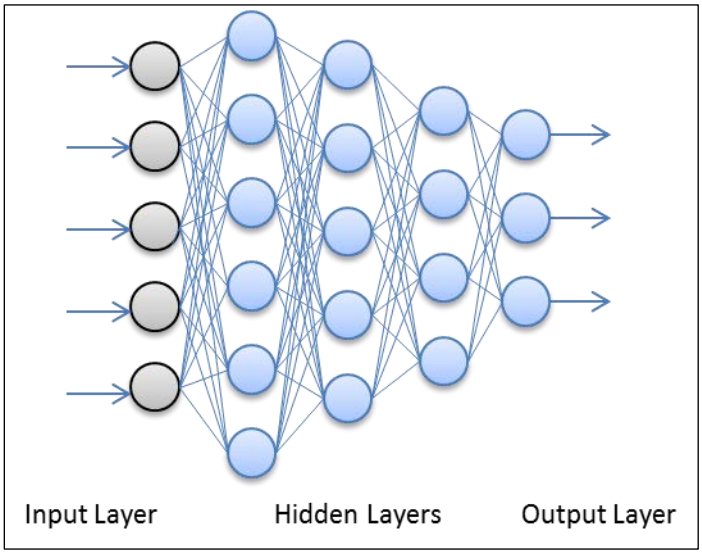

Figure 1: Neural Network Diagram

In 2006, Geoffrey Hinton and Ruslan Salakhutdinov developed techniques using multiple hidden layers. Pre-training was one such technique where the upper layers extract features with a higher level of abstraction which is used by the lower layers for more efficient classification. Unfortunately, since this technique requires billions of floating point operations, it was not computationally feasible until recently. The recent advent of technological advances in hardware caused a resurgence of interest due to the resulting improvements to performance. For example, a researcher at the Switzerland-based Dalle Molle Institute for Artificial Intelligence claims in one instance the training phase took only three days using graphic processing units (GPUs) where using CPU's would have taken five months [Edwards15]. Deep learning works well with large datasets of labeled data [Edwards15].

2.1.2 Time Series

A time series dataset consists of continuous sequences of values or events which are typically collected at fixed time intervals. Real-time surveillance systems, internet traffic, network sensors, and on-line data collection tools generate time series data which can be mined for valuable insights. Time series datasets have several applications, such as stock market analysis, sales forecasting, process and quality control, budgetary analysis, scientific experiments, and medical treatments [Han06].

Massive amounts of data can be generated in a constantly changing environment with a large number of data sources. This presents an additional challenge when working with time series data. In addition to a multitude of data formats, high change rate, and the large volumes of data collected, time may be reported inconsistently, or data may contain noise which obscures the "truth" within the data. Correlating events across multiple sources provides a comprehensive picture of the chain of events. Synchronizing or correlating the events from multiple sources introduces additional complexity [Han06].

There are three well-known window models: landmark windows, sliding windows, and decaying windows [Zhu03]. A widow can be time-based or count based. The exponentially decaying window (or damped window) is a variant of the sliding window where older events have a lower weight than more recent events [Zhu02]. Landmark windows contain aggregated values computed between a landmark point in time and the present. An example would be the average stock price of a company since its last acquisition [Zhu03].

Sliding windows are commonly used to facilitate effective event stream processing. Instead of sampling or performing computations on all of the data, only recent data is used for making decisions, thus reducing the memory required for processing. Aggregates are computed on the last **N** values and stored in the window (Figure 2). As time progresses, newer items are added, and older items are removed. The window is usually of a fixed size. Limiting the processing to recent data also prevents less relevant data from influencing statistical calculations [Zhu03].

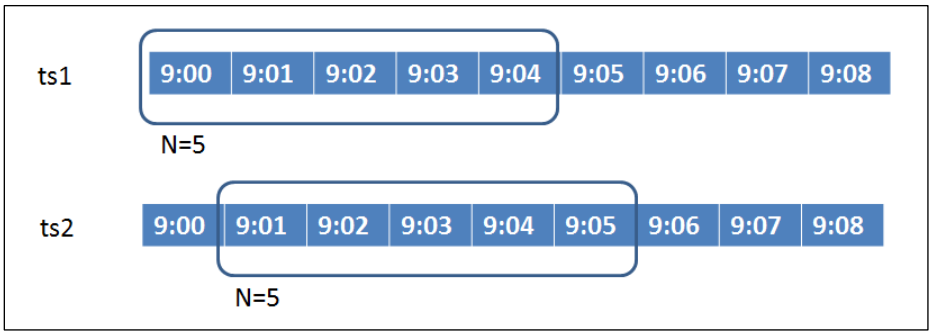

Figure 2: Sliding Window Model

The objectives of time series analysis are to forecast future values, explain how past events can impact future events, or how two time series can interact with each other. Trend analysis, similarity search, clustering, and classification are typical processes used to accomplish these objectives. Trend analysis involves identifying a trend, cyclic movement, seasonal variations, or irregular movements. Trends are depicted using a trend line over a long interval of time. Typical methods used for identifying long-term trends include the weighted average and least squares methods. Cyclic movements refer to the long term oscillations around a trend line. Seasonal variations are changes that are

calendar based and typically recur, such as holidays. Irregular movements are random chance events [Han06].

Similarity search finds sequences that differ slightly from a given sequence. Additionally, similarity search can match partial sequences or the whole sequence. An example would be to find a similar performing stock.

Clustering partitions time series data into groups based on similarity or a distance measure. Classification builds a model based on the time series in order to predict the label of an unlabeled time series.

2.2 Related Work

Many scholarly articles have been published on the topic of detecting intrusions using data mining techniques or machine intelligence [Buczak16]. The following sections are critical evaluations of recent research efforts on this topic.

2.2.1 Denial of Service and Brute force attacks

In “Applying Data Mining Techniques to Intrusion Detection,” Ng, et al. proposed an off-line solution to detect Denial of Service (DoS) and brute force password attacks [Ng15]. Their solution implements both anomaly detection and signature recognition methods. They maintain an attack signature database as well as a normal signature database. A Clustering algorithm is used on pre-processed log data to identify multiple occurrences of

similar log messages. Their tool searches the signature databases using log patterns detected while processing the log data. When the clustering algorithm detects an unusual number of event occurrences, the signature is compared to the normal log database and is ignored if found. If the signature is found in the existing attack signature database, then an alert is generated. However, if the signature is not found in either signature database, then it is presented to the user for manual classification. The initial log data was obtained from one host running the Ubuntu operating system. Attack log data was obtained by performing ICMP flood and brute force attacks against the host. A set of normal and attack patterns obtained from the initial data collection were stored in the signature database. They identified creating a real-time intrusion detection system as potential future work.

The primary shortcoming of the solution developed by Ng, et al. is that it depends on a single client log file source from one platform (Ubuntu). Additionally, it does not differentiate between events that have occurred recently or far in the past. Since their solution maintains a database of all normal activity patterns; it can only be implemented as an off-line solution. As such, it is not linearly scalable, and cannot detect suspicious user activity in real-time at an enterprise scale.

2.2.2 Web Application Attacks

Razzaq, et al. proposed a solution [Razzaq14] for detecting web application attacks by analyzing HTTP requests. The proposed solution was deployed as a web proxy that evaluates all network traffic before it is delivered to the web server. Even though the

solution only analyzes the HTTP protocol, they claim it could be expanded to other protocols. Additionally, their solution only examines portions of the headers and payload of user requests. They developed an ontology model (OWL) to build rules to analyze the user request to detect web application attacks, such as SQL Injection, DNS Cache poisoning attack, and HTTP response splitting attacks. These rules are applied to all user requests by analyzing portions of the HTTP traffic before being processed by the web server. Test attack vectors consisted of SQL Injection Cross Site Scripting (XSS) attacks using an open source tool called Web Goat to simulate the attack vectors. The solution detected web application attacks with an average detection rate of 86%. The detection rate (Figure 3) is calculated using the total number of attack records (TA) and the number of false negatives (FN). A false negative is an attack vector that is classified as normal. The performance results of the proposed system were a maximum throughput of 1400 requests per second with a maximum response time of 374 ms.

$$\text{Detection Rate} = \left[\frac{TA-FN}{TA}\right] * 100$$

Figure 3: Detection Rate Calculation

The most significant shortcoming with Razzaq's proposed solution [Razzaq14] is that all user traffic does not flow across a single web proxy. As a result, this solution is capable of evaluating only a small portion of user activity which would inevitably result in a security breach going unnoticed. Secondly, the solution only evaluates HTTP network traffic and is not linearly scalable due to the delay in evaluating every single user request before forwarding the request to its destination. Since most enterprise networks use

Secure Sockets Layer (SSL) to encrypt the network traffic in motion, the network packets will be unreadable unless the processing occurs at an SSL termination endpoint where the traffic is decrypted. These types of issues can be easily overcome by evaluating log files created by various computer systems.

2.2.3 Intrusion Detection Postmortem

Garcia, et al. proposed an off-line solution [Garcia12] to mine client log files to identify the source of a security breach. Given a security incident has already been detected, and a set of client log files, their system will attempt to locate the exploit in one of the log files. Postmortem intrusion detection is primarily used to discover how an intruder gained access to a system, what subsystems were accessed, and what information was compromised. The solution assumes that a security breach has already occurred and bypassed the Intrusion Detection System or any other security controls in place. This solution uses a combination of anomaly detection and a classification technique called KHMM which utilizes a Hidden Markov Model (HMM) and k-means clustering. The main idea around their work is that an attack would result in a sequence of system calls being logged that would not normally appear in normal activity. Normal log data is used to create a normal behavior profile. First, the log files are shrunk by replacing repetitive sequences with a meta-symbol. The log files are then pre-processed using a sliding window containing one hundred elements, stepping through the log file one hundred elements at a time. The last step builds the normal activity model from vector sequences in each window. The resulting model is used for detection. The KHMM process is composed of three steps. First, the preprocessed input is clustered using K-means. Then

the sliding window approach is used to create an HMM for each window. The last step uses an anomaly detection to compare each window with the average HMM from the previous step. If two or more consecutive abnormal windows are detected, they are marked for verification by a security analyst. The training and validation sets were composed of 32 log files from three Unix based systems (REL4, Fedora 8, and Ubuntu 9.04). The attack logs were synthetically generated using “buffer overflow” and “user to root” attacks. Experiments resulted in an average detection rate of 81.99% and false positive rate of 4.6%.

A major shortcoming of the solution proposed by Garcia et al. is that it does not detect intrusions; instead, it attempts to locate abnormal activity in a collection of client log files after a security breach has already been deemed to have occurred. Secondly, their solution can be only implemented in an off-line manner because it is not linearly scalable. This is primarily due to the fact that their solution evaluates every single user action. Scalability can be achieved by using aggregates over time of all user activity. Their solution implements a sliding window that is based on the number of events from an individual user and slides over the user session in increments equal to the size of the window. This method allows for a user sequence to cross window boundaries. Hence this presents a likely possibility that an attack sequence will be overlooked. This issue can be resolved by sliding the window using smaller increments.

Lastly, their solution is not effective because it only considers one log source type which records individual user commands. This solution may lend itself to a low false positive rate; however, if all user activity is not captured in the log, then it is highly probable that

a security breach will go unnoticed. In order to overcome this problem, multiple server source log files must be evaluated to get a complete picture of overall user activity.

2.2.4 Training a Neural Network to Mimic a Firewall

Valentan and Maly, in "Network firewall using artificial neural networks," train a multi-layer perceptron (MLP) artificial neural network to learn the rules of a firewall from the network traffic using the back propagation method [Valentan13]. The network consisted of 3 output neurons (ALLOW, REJECT, DENY), 49 input neurons, and 13 hidden neurons. The input neurons were mapped to the binary representation of IP (32 bit), port (16 bit), and protocol (1 bit). If the activation function (sigmoid) did not fire any of the output neurons, the network assumed the network packet was malicious and dropped it. The accuracy of the neural network on the testing set was 99.79%. A training dataset was generated before each epoch. The network used a cross-validation method for training. The generated dataset was split into two distinct sets (80% for training, and 20% for testing), the former for training, and the latter for testing. Network packets were created by randomly selecting a rule from the firewall table, and then randomly generating a network packet to match that rule. The training dataset consisted of a ratio of 4:1 DENY to ALLOW network packets. For testing, the dataset consisted of an equal ratio of DENY and ALLOW packets. The table of rules contains the associated action of ALLOW, REJECT, or DENY. The neural network is given the correct action during the training phase. The difference between the REJECT and DENY action is that DENY results in the packet being dropped with no response being sent to the source resulting in a "connection timed out" error. In the case of a REJECT action, the packet is prohibited from being sent

further. However, an ICMP destination unreachable response is communicated back to the source. Evaluation of the performance of the neural network was performed by comparing the total false positives and false negatives to the total number of packets evaluated. False positives were defined as malicious packets that were allowed. False negatives were normal packets that were blocked.

Training a neural network to learn the rules of a firewall is not an effective method of detecting or deterring intruders. The success of their solution is dependent on how effective the rules are at blocking malicious traffic. Commercial firewall and intrusion detection software is a better alternative for hardening the network security posture. A neural network can supplement a commercial intrusion detection system, but must be non-intrusive, and cannot impede normal operations.

2.3 Shortcomings of existing solutions

The most prevalent shortcoming of all the solutions reviewed is that they only detect and prevent individual attacks and not coordinated distributed attacks [Abad03]. Many attacks are not identified by a single log source but instead discovered when correlating information from multiple log files [Abad03]. If the attack does not result in an event being logged in the log file that is being monitored, then the attack cannot be detected using existing approaches.

Scalability is another major factor in evaluating the effectiveness of a solution. In the world of Big Data, the amount of information being stored and searched can easily grow

to several gigabytes very quickly [Garcia12]. Hence, a solution that does not scale linearly can result in slow detection response times or total system failure.

Additionally, a solution that evaluates raw network traffic to detect intrusions will result in overhead that will eventually inhibit the traffic being delivered to its destination promptly. Intrusion Detection Systems and Firewalls serve as protection controls to harden the security of the network. These systems should be complemented by implementing detection systems that are less intrusive.

Chapter 3

PROPOSED APPROACH

3.1 Overview

This research introduces the concept of a time slot. A time slot represents a small window in time which contains aggregate feature counts for that time interval. The time slot ***ts*** slides over a fixed window of time ***tw***.

The proposed approach consists of five major steps (Figure 4) with the output from each step serving as the input to the subsequent step in the process. The first step in the process, Data Collection, involves identifying and extracting log files from production systems.

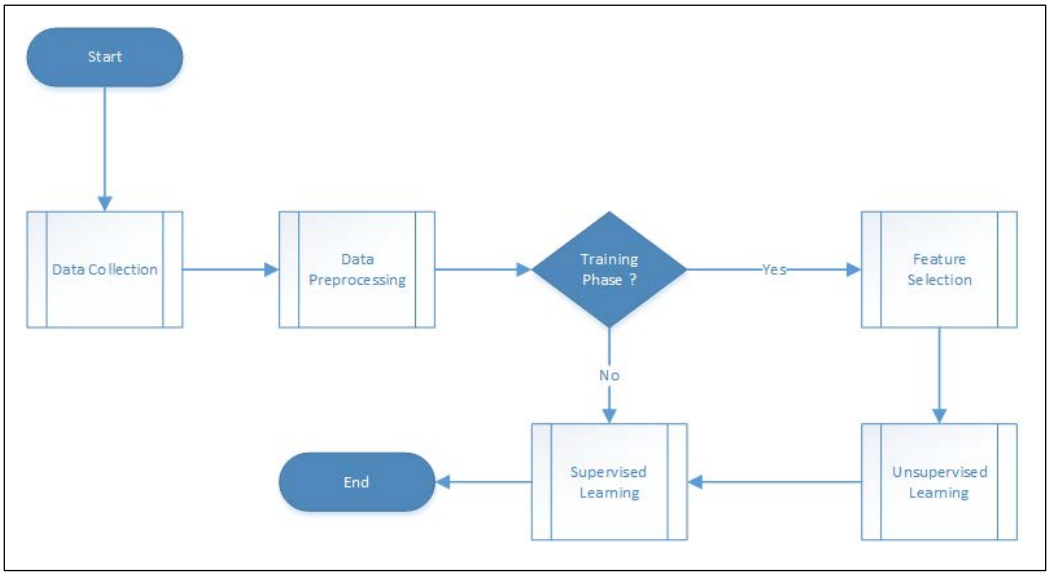

Figure 4: Process Flow Diagram

Data pre-processing is required to transform the data into a format usable by machine learning algorithms. Feature Selection is the process of identifying and selecting relevant features from the pre-processed dataset. Unsupervised learning is used to identify and learn patterns of user activity. This can be accomplished using clustering techniques. Feature selection and unsupervised learning only need to occur for training purposes. In the Supervised Learning step, the model is trained and evaluated using a classification technique using the labeled dataset from the previous step. After the model produces acceptable results, the model is trained and can be used in production phase to detect abnormal user activity.

In this research, a log entry (or instance) is referred to as an event. The term “source” is used to refer to an instance of a log file. The term “index” is used to refer to loading and parsing a log file using a search tool. The term “source type” is used to refer to a collection of log files of the same type. For example, the source type ***Neptune*** refers to the collection of log files from the Microsoft Internet Information servers used to service requests to the Microsoft Exchange servers. Microsoft Exchange is a Windows based email system.

3.2 Data Extraction and Transformation

This step is composed of three sub-tasks that collectively produce the required datasets for machine learning to occur. The data collection sub-task is the process of identifying, extracting, and integrating log data from the source systems into a single repository. Pre-processing is required to reduce the size of the dataset and transform it into a sliding

window representation. Feature selection, the process of identifying a set of features from the data to be used in machine learning, is only performed for initial training and evaluation of the model.

3.2.1. Data Collection

A familiarity with all available log source types is necessary for the purposes of detecting cyber attacks. Interviewing security professionals to identify a list of available source types is the first step in data collection. The available sources typically differ among organizations depending on their network architecture. However, possible source types may include email usage activity, firewall data, wireless access point (WAP) data, browser activity, physical facility access data, and Security Information and Event Management (SIEM) data [Mahmood13]. Web application log files are also prime candidates for consumption. Integrating these sources into a single repository allows us to build a comprehensive picture of user activity across multiple systems. Such a repository will allow us to gain insight into user activity that may be otherwise missed if examining the sources individually.

Understanding how any form of an attack could manifest itself in each of the source types is necessary for identifying potential attributes for feature extraction. The last step of data collection is identifying candidate features for extraction. The results of this step are needed in the pre-processing step where the feature extraction occurs.

3.2.2. Pre-Processing

Data transformation operations are used to convert the dataset into an appropriate structure to facilitate machine learning. Data aggregation and feature selection are common data transformation techniques used to obtain a reduced representation of the dataset without impacting its predictive accuracy [Han06].

The first step in pre-processing is to align the events in each of the source types by their respective time stamp and compute aggregate feature counts per unit time. The next step computes aggregate counts per time slot. A time slot has a fixed size and slides through time incrementally by one unit. For example, a time slot starting at time index ***t*** and size ***N*** will contain the count of feature occurrences starting at ***t*** and ending at **t**+**N**-1. Each row of the pre-processed dataset represents a collection of feature counts F_i for a single time slot ts_j. A conceptual representation of the resulting pre-processed dataset with the sliding time window is depicted in Figure 5.

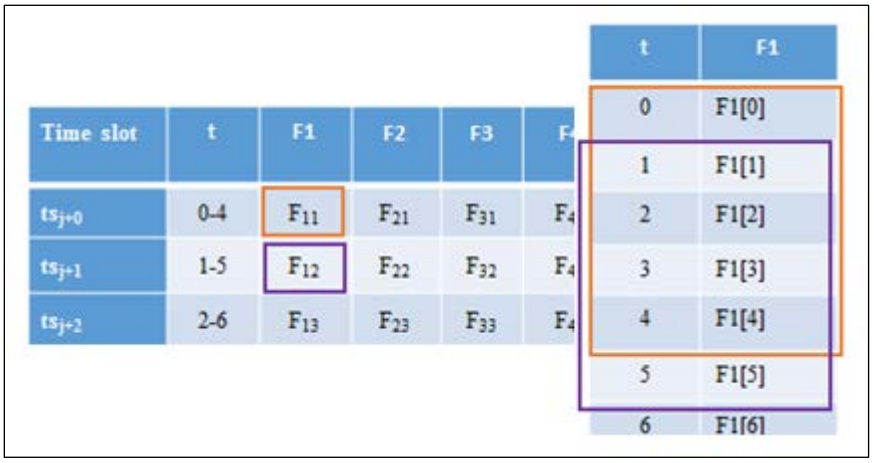

Figure 5: Pre-processed dataset with sliding time window

3.2.3. Feature Selection

A feature is an input variable or attribute that is binary, categorical or continuous in nature. The primary focus of feature selection is concerned with selecting relevant and informative features. However, other benefits exist, such as to limit storage requirements, increase calculation speed, increase predictive accuracy, and to gain an understanding of the process that generated the dataset [Guyon06].

Integrating data from multiple sources may result in a dataset containing hundreds of features some of which may be irrelevant or redundant. Redundancy can be detected by performing correlation analysis. Correlation analysis evaluates the correlation between two features. Chi-square is a common statistical method used to detect redundancy. There are other feature evaluation measures, such as Information Gain, Gain ratio, and the Gini index [Han06].

Selecting the best feature set often requires human expertise to convert raw data into a useful set of features. However, a variety of feature selection methods can be used in the absence of a subject matter expert (SME). Such methods are classified as either filters, wrappers, or embedded methods. Classical statistical methods which use correlation coefficients, such as the T-test, F-test, and chi-square, are types of filter methods used to assess variable independence. Filters calculate feature ranking based on classic statistical methods, where wrappers use the performance of a machine learning algorithm trained with the given feature subset. Embedded methods perform feature selection in the process of training, and are specific to a machine learning algorithm [Guyon06]. The hidden

layers generated during training in a neural network are an example of an embedded method.

3.3 Unsupervised Learning

Unsupervised learning techniques are typically used when the class label of each data element in a dataset is unknown. Clustering, a type of unsupervised learning is the process of grouping similar data elements into classes or clusters. Euclidean, Manhattan, and Minkowski are common similarity measures used by clustering algorithms. There are a variety of different types of clustering techniques, including but not limited to partitioning, hierarchical, density-based, and grid-based methods.

Outlier detection is a common application of clustering. Outliers are data elements that are far from all other elements and fall outside of any cluster. In some cases, the outlier may provide more insight into a problem than the normal items. Applications of outlier detection include credit card fraud detection and monitoring of electronic commerce for criminal activities. Clustering may be used in lieu of manual classification when working with very large datasets which could be very time-consuming and prone to human error.

Clustering is highly adaptable to change and can identify distinguishing features in the dataset. However, it also has some challenges. For example, clustering a large dataset may lead to biased results. Additionally, the results can be affected by noise, outliers, or missing elements. Mixed data types introduce additional complexity.

K-means is a common partitioning algorithm which calculates the center of each cluster using the mean value of all the objects in the cluster. K-medoids is similar, but instead of using the mean for the center of the cluster, it uses objects located near the center of the cluster. Partitioning based methods must be extended when working with very large datasets.

3.4 Supervised Learning

Supervised learning is the process of training a machine to accurately classify an instance or predict a value based on past examples. Data classification uses a labeled set of data called a training set to train a model for prediction, and a test set for evaluation purposes. There are several algorithms available used for classification. A renewed interest in neural networks has peaked with recent technological advances in computing power. Deep neural networks are especially known to perform well with large datasets [Edwards15].

3.5 Measurements and Evaluation

The following performance measures were used to evaluate the effectiveness of the proposed model. Accuracy is an overall measurement. However, Recall and f-score are equally important. For example, if an alert is raised when there is no security incident in progress, the cost is likely an inconvenience, however, if a security incident goes unnoticed, the cost could be devastating depending on the nature of the incident [Alpaydin14].

Accuracy (Equation 1) is defined as the ratio of correctly classified time slots to the total number of time slots [Alpaydin14].

$$\text{Accuracy} = \frac{\text{tp} + \text{tn}}{\text{N}}$$

Equation 1: Accuracy

Precision (Equation 2) is defined as the ratio of true positives to all time slots classified as positive. For example, time slots correctly classified as normal to the total number of time slots classified as normal [Alpaydin14].

$$\text{Precision} = (\text{tp}/\text{p}')$$

Equation 2: Precision

Recall (Equation 3) is defined as the ratio of true positives to the total number of actual positive time slots. In other words, the number of time slots classified correctly to the total actual time slots [Alpaydin14].

$$\text{Recall} = (\text{tp}/\text{p})$$

Equation 3: Recall

F-score is defined as the harmonic mean between precision and recall. This measure discourages models that sacrifice one measure over another [Han06].

In addition to measuring the detection performance, the training and test time was also evaluated. These measures were used to support the claim that this model is accurate, fast, and scalable.

This approach was assessed through experimentation using datasets of differing time granularity. An initial model and preliminary results using two distinct datasets are presented in the next chapter. Chapter 5 introduces additional enhancements to the model, a third dataset, and compares the results on each dataset.

Chapter 4

INITIAL MODEL AND PRELIMINARY RESULTS

4.1 System Architecture

The proposed system architecture, depicted in Figure 6, was implemented using Splunk Enterprise Edition 6.42 [Splunk17], R-Studio, and three sources which will be described in more detail in the next section. The source log files were manually loaded into Splunk using its web interface. However, a Splunk forwarder may be used to forward log files to the Splunk indexer for parsing and storing in real-time. A Splunk forwarder is also capable of receiving log data on a dedicated TCP port from high-speed appliances, such as a firewall. The Splunk search head hosts the web-based user interface and executes interactive searches and presents the results to the user.

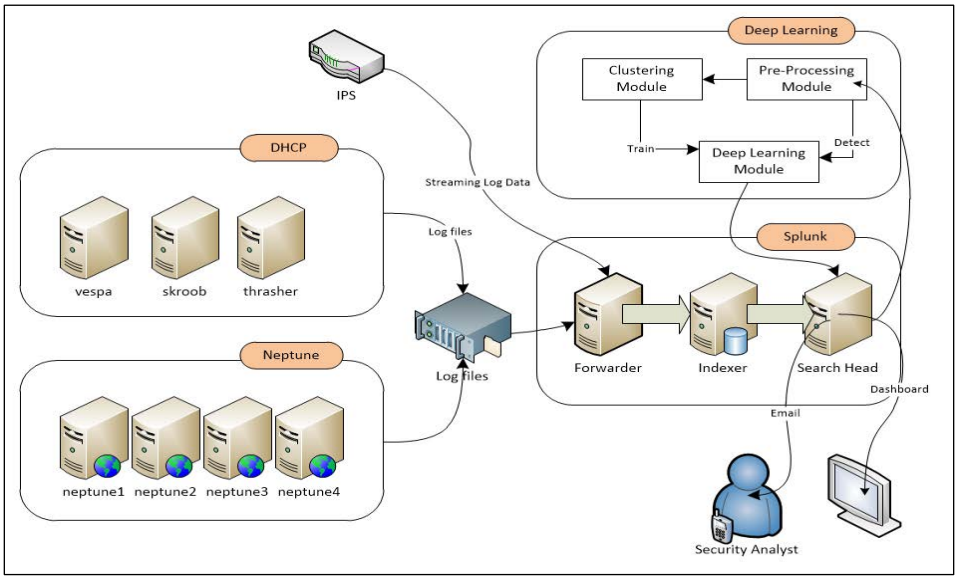

Figure 6: Proposed Solution Architecture

Splunk, a commercial log aggregation application, is used for indexing, searching, and transformation of log data. Splunk was chosen for its ease of use, fast performance, and advanced search language functionality. Loading a log file into Splunk can be initiated via drag and drop operation, and completed with just a few mouse clicks. Additionally, Splunk's architecture makes it a primary candidate for use in an online implementation. Since Splunk requires log files to be no larger than 500 MB in size, a log file splitter utility was used to load and index the log file. Due to the massive size of the logs, the import process spanned several days. The status of the import process can be determined anytime during or after the log import process by executing the Splunk command depicted in Figure 7. This command will display the source type, first event, last event, and a total number of events logged for each source type.

```
| metadata type=sourcetypes  | eval firstEvent = strftime(firstTime, "%m-%d-%Y %H:%M:%S") | eval
lastEvent=strftime(lastTime,"%m-%d-%Y %H:%M:%S") | table sourcetype, firstEvent, lastEvent,
totalCount | sort firstEvent
```

Figure 7: Verify Log File Import

A Splunk search command was executed to create a dataset of aggregate feature counts in one-minute intervals. This aggregated data was then exported to a CSV file, and fed into the Pre-Processing module. The Pre-Processing module converts the one-minute interval total counts to into a five-minute sliding window representation. For initial training, the data is fed into the Clustering Module where the dataset is classified and labeled. The resulting classified dataset is used by the Deep Learning module for training and testing. After the model is trained, Pre-Processed data is then fed directly into the Deep Learning

module for incident detection. The system will generate in real-time alerts and updates to dashboards when it detects abnormal activity.

4.2 Data Collection

The University of North Florida Information Technology Security Department provided a "sanitized" set of log files used for this experiment. These files were extracted from real production system logs and altered to obscure user information. The log files are listed in Table 1.

Source Type	File Type	Source	File Count	First Event	Last Event	Total Events	Total Size (GB)
Neptune	CSV	Microsoft IIS	60	04/10/15 03:59:29	04/25/15 03:59:52	61,600,331	21. 5
DHCP	Text	DHCP	9	04/10/15 00:00:00	04/24/15 23:59:59	26,988,670	4.6
IPS	CSV	Tipping Point	1	04/09/15 08:27:34	04/24/15 23:59:59	11,404,635	1.6

Table 1: Source Log Files

Two datasets were extracted from the integrated log files in Table 1 for the purposes of evaluating the model performance with varying parameters. These datasets are defined in Table 2. The main difference between the two datasets is the size of the dataset and its time window. Experimentation was performed using each dataset.

	Dataset 1		Dataset 2	
Instances	176	5-minute time slots	2876	5-minute time slots
Attributes	17	2 time fields + 15 features	17	2 time fields + 15 features
Time Window Size	180	Minutes	2880	Minutes
Start Time	21:00	04/19/2015	00:00	04/14/2015
End Time	00:00	04/20/2015	00:00	04/16/2015
Total Events	995,701		12,786,858	

Table 2: Dataset Definitions

The datasets depicted in Table 2 were created using the time slot concept to model the data. The time slot size selected for both datasets was five minutes. Each row in the dataset contains aggregate feature counts for five minutes. For example, in three hours of log data examined, one time slot represented aggregate counts of 26,807 events. This has the effect of reducing the number of resources needed to represent all the data for each dataset drastically allowing the system to scale linearly as new log files are introduced.

The log files for this research were extracted from the source systems, compressed, and transferred to DVD media. As a result, this research method is conducted in an off-line manner. A production deployment is not in the scope of this research. However, this research can be implemented in a near real-time manner. The training and test datasets needed for this research are created using the log files and contain aggregate count values in time series.

4.3 Feature Selection

The features selected for machine learning are derived counts based on specific attributes from one or more log files. Selecting the individual user names or IP values as features would result in a sparse matrix which would exponentially increase the memory requirement. By examining three hours of the data collected it becomes evident that such a solution would not be linearly scalable. In one particular case, there were no more than 316 active users out of a total 2,436 possible users. Figure 8 depicts the distribution of active users for this timeframe. Similarly, approximately 50% of the possible IP addresses were active at any point during the same timeframe. Consequently, these attributes were not selected as features.

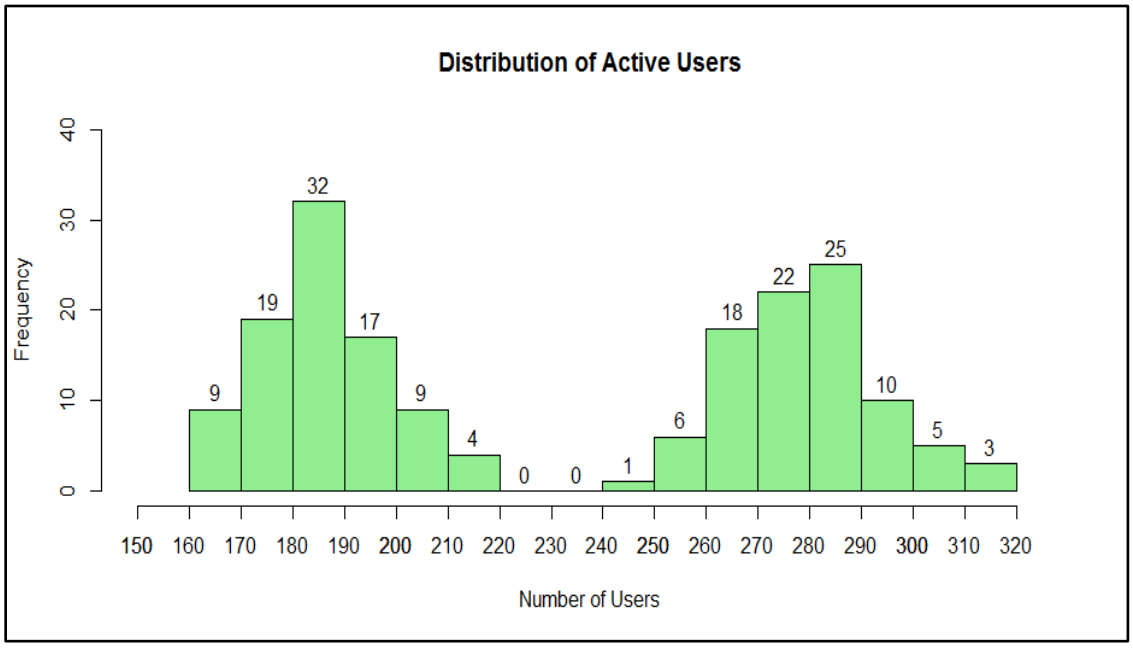

Figure 8: Active User Distribution

The features selected for this research (Table 3) were derived from aggregate values using the Neptune, DHCP, and IPS source types.

Ord #	Log	Feature	Description
1	Neptune	postCount	Count of HTTP POST requests
2	Neptune	getCount	Count of HTTP GET requests
3	Neptune	uniqueUserCount	Distinct count of Users
4	Neptune	HTTP2XX	Count of HTTP 2XX status codes
5	Neptune	HTTP4XX	Count of HTTP 4XX status codes
6	Neptune	HTTP5XX	Count of HTTP 5XX status codes
7	Neptune	owaUserCount	Count of Office Web Access requests
8	Neptune	activeSyncUserCount	Count of ActiveSync requests
9	Neptune	macUserCount	Count of MAC Outlook requests
10	IPS	foreignIPCount	Count of requests outside of the USA
11	IPS	facultyCount	Count of requests from faculty or staff
12	IPS	studentCount	Count of requests from students
13	IPS	blockCount	Count of requests blocked by IPS
14	IPS	permitCount	Count of requests permitted by IPS
15	DHCP	DHCPDiscover	Count of DHCP Discover requests

Table 3: Features used for Machine Learning

The “Neptune” source type contains event data from four Windows servers running Microsoft Internet Information Server (IIS). The structure of this source type adheres to the W3C Extended Log File standard [Hallam-Baker96]. The events contained in this source type are the result of user email activity. The features derived from this source type include the total number of HTTP POST and GET requests, the total number of successful and unsuccessful requests, the distinct count of users, and the number of Active Sync, Web Access, and MAC users. The sample event in Figure 9 depicts in bold

print the portions used to derive the *postCount*, *activeSyncUserCount*, *uniqueUserCount*, and *HTTP2XX* features. The features *uniqueIPCount* and *uniqueUserCount* appear to have a strong correlation as shown in Table 4.

D:\Elfa_Data\Neptune\Raw\4\u_ex150419_x.log,293972,2015-04-19,23:59:59,139.62.192.204,***POST***, /***Microsoft-Server-ActiveSync***/default.eas,User=User951&DeviceId=ApplDKVLK09WDVGF&DeviceType=iPad &Cmd=Ping&CorrelationID=<empty>;&ClientId=EPYTCILETMFIVQOYCFG &cafeReqId=f0cf56aa-c4b7-4474-8f5e-4ec2b0e4d895;,443,***UNFCSD\User951***,139.62.193.253, Apple-iPad3C2/1206.69,,***200***,0,0,24625,76.122.20.229

Figure 9: IIS Log Entry Sample

Attribute	Attribute	p-value	cor
postCount	getCount	0.182	-0.1010718
uniqueUserCount	uniqueIPCount	2.2e-16	0.9819344
blockCount	permitCount	0.2986	0.07878555
uniqueUserCount	owaUserCount	0.9263	0.007020625
owaUserCount	activeSyncUserCount	0.9028	0.009272762
owaUserCount	macUserCount	1.064e-08	-0.4145732
uniqueIPCount	foreignIPCount	0.09486	-0.1262986

Table 4: Correlation Results for Features

The DHCP source type contains event data from three UNIX servers which process requests for the network (IP) address for hosts connecting to the network using Dynamic Host Configuration Protocol [Droms97]. The sample event depicted in Figure 10 is used to derive the feature *DHCPDiscover*.

Apr 19 23:59:58 thrasher dhcpd: ***DHCPDISCOVER*** from 40:25:c2:7b:d3:14 via eth0

Figure 10: DHCP Log Entry Sample

The IPS source type contains event data from the Tipping Point Intrusion Prevention System (IPS), an industry standard Intrusion Prevention System. The IPS system logs events when any network traffic matching a rule is detected. The sample event depicted in Figure 11 is used to derive the following features: *blockCount*, *facultyCount* , and *foreignIPCount*.

2015-04-19 23:59:34",Low,"7611: DNS Reputation",Reputation,***Block***,1,**Faculty-Staff**,139.62.200.212,34847,***199.249.119.1,53***,192,download.newnext.me

Figure 11: IPS Log Entry Sample

4.4 Pre-Processing

The Splunk search in Figure 12 was used to create the datasets for this research by varying *earliest* and *latest* date-time values. The results were exported into a CSV format.

```
index=main (sourcetype=neptune OR sourcetype=tpsms OR sourcetype=dhcp) earliest=04/19/2015:21:00:0
latest=04/20/2015:0:0:0 | eval statusCd=substr(sc_status,1,1) | iplocation DEST_IP | bucket _time span=1m | eval
dhcpCMD=if(match(_raw,"DHCPDISCOVER"),"DISCOVER","") | eval userType=if(like(cs_uri_stem,"%owa%"),"OWA",
if(like(cs_uri_stem,"%Microsoft-Server-ActiveSync%"),"ASYNC", if(like(cs_User_Agent,"MacOutlook%"),
"MACOUTLOOK", "OTHER"))) | stats count(eval(cs_method="POST")) as postCount, count(eval(cs_method="GET"))
as getCount, dc(cs_username) as uniqueUserCount, dc(OriginalIP) as uniqueIPCount, count(eval(statusCd="2")) as
HTTP2XX, count(eval(statusCd="4")) as HTTP4XX, count(eval(statusCd="5")) as HTTP5XX, mode(FILTER) as
primaryReason, count(eval(userType="OWA")) as owaUserCount, count(eval(userType="ASYNC")) as
activeSyncUserCount, count(eval(userType="MACOUTLOOK")) as macUserCount,
count(eval(dhcpCMD="DISCOVER")) as DHCPDiscover, count(eval(Country!="United States")) as foreignIPCount,
count(eval(PROFILE="Faculty-Staff")) as facultyCount, count(eval(PROFILE="Dorms-Guest")) as studentCount,
count(eval(ACTION="Block")) as blockCount, count(eval(ACTION="Permit")) as permitCount, mode(VLAN_NUM) as
primaryVLAN by _time
```

Figure 12: Splunk Transformation Query

The exported CSV data is converted into a sliding window representation using an R-Script. The purpose of this step is to preserve a continuous set of temporal values as the system advances through each row in the dataset which contains the aggregate feature counts for one time slot. For example, given a time slot size of five minutes and a sixty minute time window starting at 21:00, the first row in the dataset contains aggregate feature counts for the time slot from 21:00 through 21:04. The second row contains aggregate feature counts from 21:01 through 21:05, and so forth. The start time for each subsequent time slot starts one-minute later than the previous time slot began. The time slot start and end times are included as the first two fields of each dataset as shown in Figure 13. These time fields were not used for machine learning, instead, are included in order to provide the actual time frame to a security analyst for investigation purposes.

ts.start	ts.stop	postCount	getCount	uniqueUserCount	uniqueIPCount
2015-04-19T21:00:00.000-0400	2015-04-19T21:04:00.000-0400	5177	1165	1222	1132
2015-04-19T21:01:00.000-0400	2015-04-19T21:05:00.000-0400	5143	1166	1157	1062
2015-04-19T21:02:00.000-0400	2015-04-19T21:06:00.000-0400	5189	1019	1220	1121
2015-04-19T21:03:00.000-0400	2015-04-19T21:07:00.000-0400	5388	1000	1125	1030
2015-04-19T21:04:00.000-0400	2015-04-19T21:08:00.000-0400	5454	1089	1234	1139
2015-04-19T21:05:00.000-0400	2015-04-19T21:09:00.000-0400	5361	1186	1125	1023

Figure 13: Partial Dataset Image

4.5 Unsupervised Learning Results

A classified dataset consisting of normal and abnormal activity is needed for supervised learning to occur. Classification would be extremely labor intensive due to the massive size of the log files. For example, if activity in one-time slot warranted investigation, a security analyst could potentially need to review over 30,000 log entries, thus making visual identification and classification impossible.

Generating synthetic data for abnormal activity was considered because there were no known security incidents during the timeframe the log data was collected. However, there is an inherent risk when assuming that the log data contains only normal activity. If anomalies exist in the data, the model may inaccurately classify instances, or worse ignore real security incidents. Consequently, clustering was used to identify anomalous activity within the training dataset.

The Partitioning Around Medoids (PAM) algorithm was chosen to classify the dataset into three clusters of activity. PAM was chosen because it is resistant to outliers and allows clustering of categorical values. Each cluster is classified as normal, critical, or

warning, and is labeled *green*, *red*, or *yellow*, respectively. The cluster score is calculated from the median value of the sum of all features and is used to determine the label assigned to each cluster. R code for calculating the cluster score is depicted in Figure 14. The cluster with the lowest score was labeled *green*. The cluster with the highest score was labeled *red*, and the remaining cluster was labeled *yellow*.

```
l<-which(wbpam$clustering %in% c(1))
cluster.scores<-
c(median(rowSums(tw[l,])))
l<-which(wbpam$clustering %in% c(2))
cluster.scores<-c(cluster.scores,
median(rowSums(tw[l,])))
l<-which(wbpam$clustering %in% c(3))
cluster.scores<-c(cluster.scores,
median(rowSums(tw[l,])))
print(cluster.scores)
```

Figure 14: R Code to Calculate Cluster Scores

The classification results for each dataset are shown in Figure 15. It is worth noting that all of the cluster scores resulting from Dataset 2 are lower than those from Dataset 1. The green cluster score from Dataset 2 is fifty-seven percent lower its counterpart.

Dataset 1			
Cluster Number	Cluster Size	Label	Score
1	74	green	29,646
2	39	yellow	49,054
3	63	red	72,774

Dataset 2			
Cluster Number	Cluster Size	Label	Score
1	1199	red	65,966
2	588	yellow	38,627
3	1089	green	16,981

Figure 15: Clustering Confusion Matrixes

Figure 16 contains box plots depicting the difference in the scale of activity for each dataset. The Y-axis represents the sum of all features for each instance in a cluster. The *normal* and *warning* clusters in Dataset 2 overlap. Further analysis will reveal that the skewed results from the clustering Dataset 2 were due to clustering on such a large time window.

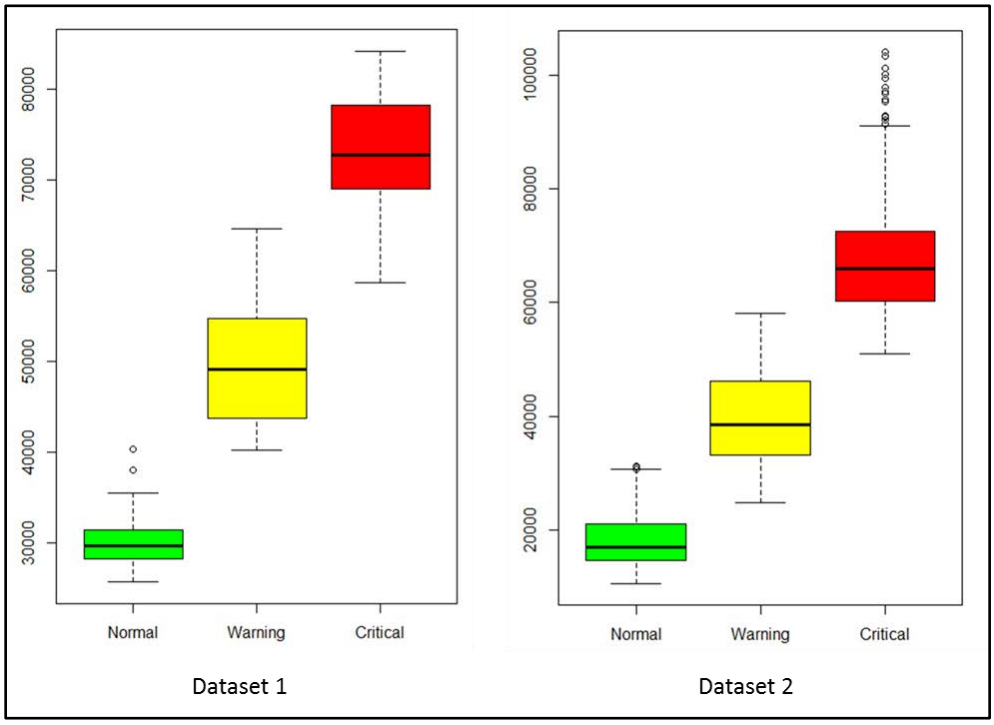

Figure 16: Cluster Scores

Typical user activity patterns appear to follow a Gaussian distribution throughout a normal business day. This is illustrated by the data from Dataset 2 in Figure 17. As a result, the peak activity times in Dataset 2 were classified as red, non-peak as *green*, and the transition period as *yellow*.

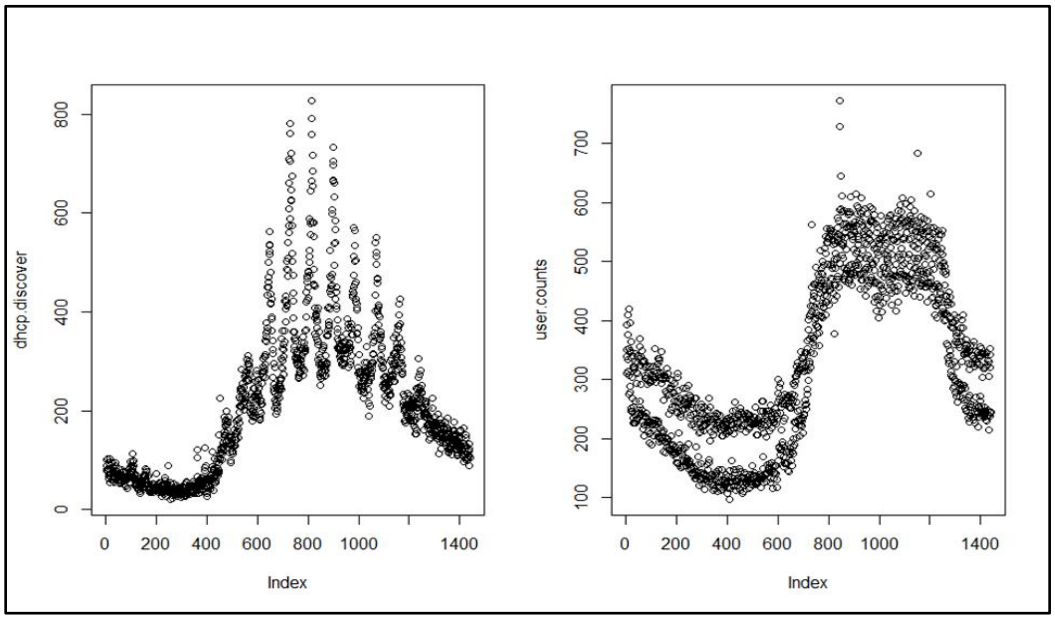

Figure 17: User Activity Distribution

Table 5 depicts the time slots color-coded according to each cluster in Dataset 1 and includes the total events, average number of events per minute (EPM), start and end times, and classification duration in minutes.

Cluster	Beginning Time Slot	Ending Time Slot	Start Time	End Time	Duration (min)	AVG EPM	Total Events
Green	1	16	21:00	21:20	20	1,324	26,474
Yellow	17	18	21:16	21:22	6	2,934	17,603
Red	19	34	21:18	21:38	20	5,456	109,111
Yellow	35	37	21:34	21:41	7	3,623	25,363
Green	38	83	21:37	22:27	50	1,250	62,501
Yellow	84	105	22:23	22:49	26	3,054	79,391
Red	106	108	22:45	22:52	7	4,018	28,123
Yellow	109	110	22:48	22:54	6	3,384	20,303
Green	111	115	22:50	22:59	9	2,110	18,991
Yellow	116	117	22:55	23:01	6	3,219	19,315
Red	118	152	22:57	23:36	39	5,361	209,096
Yellow	153	157	23:32	23:41	9	4,938	44,442
Red	158	166	23:37	23:50	13	5,297	68,858
Yellow	167	169	23:46	23:53	7	3,349	23,4438
Green	170	176	23:49	00:00	11	1,614	17,750

Table 5: Time Slot Classification Results

Plotting the feature *postCount* confirms anomalous user activity occurred during the three-hour time window, shown in the top half of Figure 18. The red line is the average of events per minute of the red clusters in Table 5. The activity above this line indicates abnormal activity. The area between the yellow and red lines is indicative of a border state between normal and abnormal activity.

The bottom chart in Figure 18 is a time chart of the feature *postCount* from Dataset 2 using the same boundaries as the top graph. The amount of time above the red line is notably smaller than that from Dataset 1.

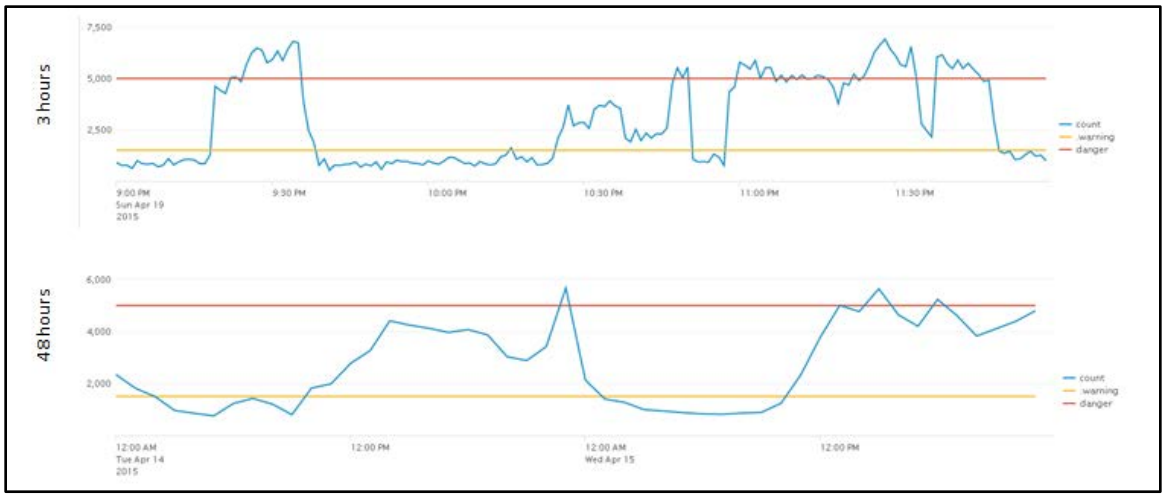

Figure 18: HTTP POST Requests

Approximately 38 percent of the user activity in Dataset 2 was classified as abnormal. If we assume user activity remains constant throughout the day, the thresholds should remain constant. However, the chart of Dataset 2 (48 hours) in Figure 18 using the same threshold for abnormal activity as Dataset 1, shows most of the activity is below the control boundary. It is apparent that the threshold for abnormal activity changes throughout the day based on user activity and the size of the time window chosen impacts the accuracy of the clustering results. In this case, a larger time window produced biased results.

Future experiments using a smaller time window and a larger period of activity are expected to result in more accurate clustering and facilitate learning routine activity patterns specific to any hour of any day of the week.

4.6 Supervised Learning Results

The R package “h2o” was used to train and test a neural network using the deep learning algorithm. The dataset was split into 70/30 % for training and testing, respectively, maintaining an equal proportion of each class in both the training and test sets.

The experiments conducted used one hundred epochs and the hyperbolic tangent for the activation function. Determining the optimal network topology is not a trivial task. Therefore these experiments used a simple network topology of one hidden layer with two neurons. Table 6 depicts the overall results of the deep learning algorithm on both datasets. The larger dataset (Dataset 2) resulted in greater accuracy. The confusion matrixes for both datasets are depicted in Table 7. The accuracy of the Deep Learning algorithm was slightly less than that of the Weka Multi-Level Perceptron (MLP). The h2o deep learning algorithm was significantly faster than the Weka MLP.

Deep Learning (h2o)	Dataset 1	Dataset 2
Time Window	3 hours	48 hours
Time slot	5 min	5 min
Number of raw events	995,701	12,786,858
Number of Instances	176	2876
Test Instances	51	858
Number of Attributes	17	17
Ignored Attributes	2	2
Time to train model	1.41s	7.69 s
Accuracy	96.08%	98.14%
Precision	91.67%	99.07%
Recall	100%	98.15%
F-score	95.65%	98.61%

Table 6: Deep Learning Results

Actual	Predicted					
	Dataset 1			Dataset 2		
	Green	Red	Yellow	Green	Red	Yellow
Green	22	0	0	320	0	6
Red	0	18	0	0	356	3
Yellow	2	0	9	3	4	169

Table 7: Deep Learning Confusion Matrixes

Chapter 5

EXPERIMENTS AND RESULTS

5.1 Overview

In the previous section, it was shown that user activity typically follows a normal distribution and can vary with the time of day. In order to account for the dynamic nature of user activity and preserve the prediction accuracy of the model, the experiments described in this section will introduce two new features and several new methods, such as normalization, rule-based clustering, split-level clustering, and topology analysis. Finally, the model was trained and evaluated using the original datasets used in the previous section, in addition to a newly created dataset.

5.2 Data Collection

A third and final dataset that spans approximately two calendar weeks was created for the purposes of evaluating the model performance on a larger sample of log data. This dataset was used to train the model to learn normal activity patterns that occur at various times during the day and evaluate its performance at detecting those user activities that fall outside of the normal range. It is worth noting that the new dataset is a superset of the other two datasets (Table 8).

	Dataset 1		Dataset 2		Dataset 3	
Instances	176	5-minute time slots	2876	5-minute time slots	18,896	5-minute time slots
Attributes	20	2 time fields + 18 features	20	2 time fields + 18 features	20	2 time fields + 18 features
Time Window	180	Minutes	2880	Minutes	18900	Minutes
Start Time	21:00	04/19/2015	00:00	04/14/2015	21:00	04/10/2015
End Time	00:00	04/20/2015	00:00	04/16/2015	23:59	04/23/2015
Total Events	995,701		12,786,858		102,993,636	

Table 8: Dataset Definitions

Each dataset is composed of one-minute feature aggregates derived from the original log files. The features used for machine learning are depicted in Table 9. The source log file of each feature is listed with its description. This is the same feature set used in the previous section, with the addition of the two new calculated fields: *dhour* and *wday*. The purpose of introducing the new features is to model the dynamic nature of user activity over time. For example, a normally occurring pattern during the afternoon may not normally occur in the middle of the night, and hence is suspicious in nature or could be an attack.

Ord #	Log	Feature	Description
1	Calculated	wday	Day of week (0-6)
2	Calculated	dhour	Hour of day (0-23)
3	Neptune	postCount	Count of HTTP POST requests
4	Neptune	getCount	Count of HTTP GET requests
5	Neptune	uniqueUserCount	Distinct count of Users
6	Neptune	uniqueIPCount	Distinct count of IP values
7	Neptune	HTTP2XX	Count of HTTP 2XX status codes
8	Neptune	HTTP4XX	Count of HTTP 4XX status codes
9	Neptune	HTTP5XX	Count of HTTP 5XX status codes
10	Neptune	owaUserCount	Count of Office Web Access requests
11	Neptune	activeSyncUserCount	Count of ActiveSync requests
12	Neptune	macUserCount	Count of MAC Outlook requests
13	DHCP	DHCPDiscover	Count of DHCP Discover requests
14	IPS	foreignIPCount	Count of requests outside of the USA
15	IPS	facultyCount	Count of requests from faculty or staff
16	IPS	studentCount	Count of requests from students
17	IPS	blockCount	Count of requests blocked by IPS
18	IPS	permitCount	Count of requests permitted by IPS

Table 9: Features used for Machine Learning

5.3 Pre-processing

The pre-processing module converts the datasets listed in Table 8 into a five-minute sliding window representation by summing the feature aggregates. The reason for using the sum instead of the median or mean is that the mean or median could mask a subtle fluctuation in an activity that would otherwise go unnoticed. Additionally, the pre-processing module introduces two new features which allow the neural network to accurately differentiate abnormal activity from fluctuations that may normally occur throughout the day. The new features are *wday* and *dhour*. The *wday* feature is the

ordinal number of the calendar day of the week (0-6). The *dhour* feature represents the hour of the timeslot (0-23). The time required for preprocessing each dataset is listed in Table 10.

Dataset Size	Preprocessing Time (s)
3 hour	0.570
2 day	18.53
2 week	187.29

Table 10: Pre-processing Times

5.3.1 Normalization

Normalization is performed by the pre-processing module to prepare the data for machine learning. The purpose of normalization is to bring all features into a common range so that one feature does not have higher precedence than any other feature. Normalization was performed on each feature column using Min-Max normalization [Figure 19].

$$v' = \frac{v - minA}{maxA - \min A}$$

Figure 19: MinMax Normalization

Normalization allows for easier comparison when charting features with a different scale. Additionally, normalization can speed up the time required to train the neural network [Han06]. Normalizing the dataset preserves the shape of the feature plots as can be seen in Figure 20.

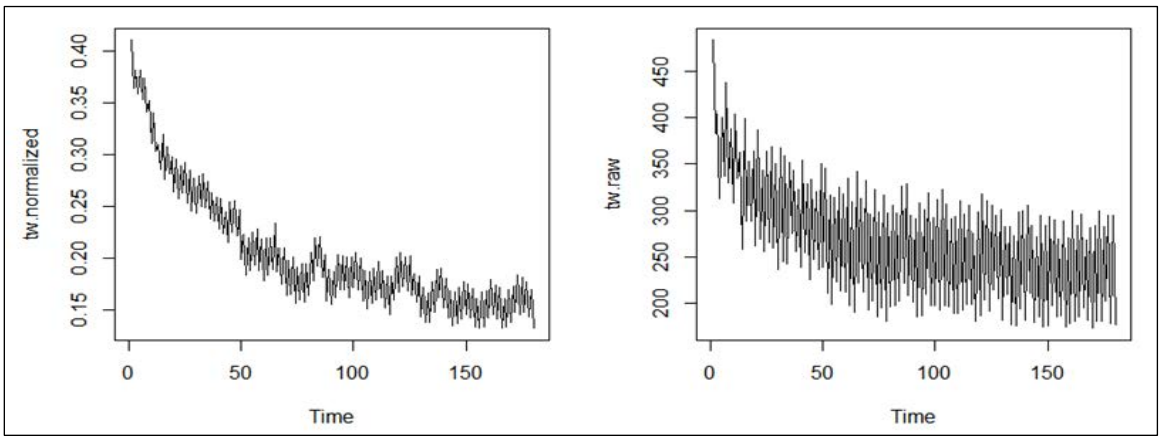

Figure 20: Effect of Normalization

5.4 Unsupervised Learning Results

The source log files used for this research were not known to have any intrusions at the time they were collected, and as a consequence, the datasets were not labeled. Abnormal activity patterns were discovered to exist within the data. However, there lacked a sufficient sample to train a neural network effectively. Due to the size of the log files, manual labeling of a dataset would require intensive effort. Hence, the Partitioning around Medoids (PAM) algorithm was used to create a labeled dataset with a proportional number of examples for each class. The PAM clustering results are shown in Table 11.

PAM Cluster Label	3 hour dataset		2 day dataset		2 week dataset	
	Size	Score	Size	Score	Size	Score
Green	104	4.644	1405	2.157	6375	1.642
Yellow	22	6.851	756	6.799	6142	2.393
Red	50	6.944	715	7.705	6379	5.663
Time (s)		0.03		3.64		7080.567

Table 11: PAM Clustering Results

Three classifications were chosen to model a common business view of user activity. The classifications *green*, *yellow*, and *red* were used. These classifications also reflect the criticality or urgency of activity. Normal user activity patterns are labeled *green*. Known attack patterns or activities that have a high sense of urgency are labeled *red*. Patterns that are suspicious, unknown or are a precursor to a cyber attack are labeled *yellow*.

Each of the datasets was partitioned into three clusters and labeled using a cluster scoring function. The cluster score was calculated by summing of the features of the cluster's medoid. The cluster with the lowest score was labeled *green*. The cluster with the largest score was labeled *red*, and the remaining cluster was labeled *yellow*. The medoids for each of the datasets are shown in Tables 12, 13, and 14.

PAM Medoids – 3 hour	Green	Yellow	Red
wday	0	0	0
dhour	0.333333	0	0.666667
postCount	0.082636	0.902882	0.85225
getCount	0.405852	0.360051	0.227735
uniqueUserCount	0.517241	0.366379	0.383621
uniqueIPCount	0.468	0.348	0.4
HTTP2XX	0.249519	0.677511	0.555995
HTTP4XX	0.009185	0.944362	0.926377
HTTP5XX	0.142857	0.142857	0.571429
owaUserCount	0.466337	0.410891	0.30099
activeSyncUserCount	0.230435	0.165217	0.407453
macUserCount	0.514644	0.158996	0.426778
DHCPDiscover	0.366071	0.370536	0.549107
foreignIPCount	0.129208	0.677889	0.097361
facultyCount	0.278486	0.623087	0.120961
studentCount	0.102689	0.01467	0.242054
blockCount	0.277226	0.619922	0.127701
permitCount	0.070362	0.06823	0.08742
Score	4.644084	6.85148	6.943898

Table 12: Medoids for Dataset 1

PAM Medoids – 2 day	Green	Yellow	Red
wday	0.5	0	0.5
dhour	0.125	0.666667	0.75
postCount	0.065534	0.37466	0.48549
getCount	0.089142	0.513308	0.51542
uniqueUserCount	0.119048	0.725369	0.719622
uniqueIPCount	0.107387	0.689009	0.707027
HTTP2XX	0.130583	0.432902	0.619718
HTTP4XX	0.019258	0.290196	0.317213
HTTP5XX	0.130435	0.173913	0.347826
owaUserCount	0.085845	0.54551	0.51172
activeSyncUserCount	0.210561	0.383993	0.532013
macUserCount	0.110621	0.589032	0.692405
DHCPDiscover	0	0.320021	0
foreignIPCount	0.06921	0.098309	0.1337
facultyCount	0.176685	0.432465	0.41543
studentCount	0.008959	0.035835	0.012833
blockCount	0.160679	0.40313	0.376901
permitCount	0.048089	0.124538	0.067818
Score	2.157035	6.798858	7.705136

Table 13: Medoids for Dataset 2

PAM Medoids – 2 week	Green	Yellow	Red
wday	0.142857	0.714286	0.428571
dhour	0.291667	0.375	0.666667
postCount	0.052542	0.105571	0.32932
getCount	0.108127	0.062672	0.474174
uniqueUserCount	0.06545	0.116425	0.537445
uniqueIPCount	0.058354	0.105147	0.542527
HTTP2XX	0.045769	0.188832	0.485073
HTTP4XX	0.050401	0.040254	0.182346
HTTP5XX	0.003175	0.022222	0.069841
owaUserCount	0.09505	0.086846	0.527581
activeSyncUserCount	0.290696	0.264154	0.449103
macUserCount	0.027767	0.042585	0.202964
DHCPDiscover	0.118068	0	0.061643
foreignIPCount	0.016903	0.079403	0.162343
facultyCount	0.110815	0.05842	0.192988
studentCount	0.009683	0.009441	0.029049
blockCount	0.109422	0.056035	0.189332
permitCount	0.045732	0.066311	0.132622
Score	1.642476	2.393603	5.663589

Table 14: Medoids for Dataset 3

5.4.1 Rule-based Clustering

Rule-based clustering was introduced to provide a different method of labeling data since clustering resulted in a near linear split of the data. This method attempts to fit the data to a more complex, non-linear equation which would be more representative of an attack. Additionally, a Subject Matter Expert (SME) may classify some events in the logs differently from another SME. The rule set chosen does not impact the validity of this

approach, as such the rules used in this experiment could be replaced with an entirely different set and achieve similar results.

This method utilized four rules that explicitly reference features from three different log sources. The rules were derived from an interview with a security analyst from a discussion on what events could represent attacks in the logs. Using the same classifications introduced earlier, the classes were defined as follows. Instances that matched one of the rules were labeled *yellow*, while instances that matched more than one rule were labeled *red*. Instances that did not match any of the rule patterns were labeled *green*. The results of the rule-based classification are depicted in Table 15.

Rule Cluster Label	3 hour dataset		2 day dataset		2 week dataset	
	Size	Score	Size	Score	Size	Score
Green	79	5.326	1593	1.961	9894	1.421
Yellow	72	5.902	909	5.598	5548	2.602
Red	25	7.148	374	6.966	3454	4.5
Time (s)		0.024		0.322		2.03

Table 15: Rule-based Clustering Results

The rules used in this method are listed below.

- Rule 1: High rates of DHCP discover requests are representative of a DHCP starvation attack.
- Rule 2: High connection counts to foreign IP's with a high rate of HTTP POST requests could be a malware attack.

- Rule 3: High rate of HTTP GET requests with low unique user counts could be representative of a denial of service attack.
- Rule 4: High number of unauthorized attempts for access is likely to be reconnaissance for an attack.

In order to provide a proportional number of examples for each class, the quantile function was used on the feature values to establish a dynamic threshold. For example, all instances where the DHCP discover value exceeds the 75% quantile were considered an attack. This method was faster than using PAM clustering. Clustering the two-week dataset using PAM took just under two hours compared to the rule-based method which took just over two minutes. The rule-based method also resulted in a smaller proportion of non-normal examples than the PAM method. For example, using the PAM method on Dataset 3 resulted in approximately 33% of activity in each cluster. The rule-based method classified 18% of the activity as critical or *red*.

5.4.2 Feature Ranking

After the datasets had been labeled, the features were ranked using an Information Gain attribute evaluator using Weka. The feature ranking for the PAM clustered data is shown in Table 16. The *wday* feature is a constant value in the three-hour dataset. Hence it was ranked zero. Any of the features ranked zero could be dropped without impacting the accuracy of the model, however, all of the features were retained for the experiments in this research. The new features have a higher ranking in the other two datasets. The

features targeted by the rule-based clustering were ranked higher than the other features as can be seen in Table 17.

Dataset 1 (3 hours)			Dataset 2 (2 days)			Dataset 3 (2 weeks)		
Rank	Ordinal #	Feature	Rank	Ordinal #	Feature	Rank	Ordinal #	Feature
0.828	3	postCount	0.947	2	dhour	0.774	5	uniqueUserCount
0.811	8	HTTP4XX	0.903	3	postCount	0.77	1	wday
0.675	2	dhour	0.902	7	HTTP2XX	0.77	6	uniqueIPCount
0.517	17	blockCount	0.894	8	HTTP4XX	0.72	12	macUserCount
0.446	15	facultyCount	0.846	5	uniqueUserCount	0.634	4	getCount
0.396	14	foreignIPCount	0.838	6	uniqueIPCount	0.63	7	HTTP2XX
0.353	16	studentCount	0.769	11	activeSyncUserCount	0.581	10	owaUserCount
0.261	7	HTTP2XX	0.748	12	macUserCount	0.499	3	postCount
0.257	11	activeSyncUserCount	0.73	13	DHCPDiscover	0.438	2	dhour
0.172	18	permitCount	0.694	4	getCount	0.417	11	activeSyncUserCount
0.156	4	getCount	0.594	10	owaUserCount	0.381	8	HTTP4XX
0.126	10	owaUserCount	0.516	15	facultyCount	0.297	13	DHCPDiscover
0.094	13	DHCPDiscover	0.512	1	wday	0.293	9	HTTP5XX
0.088	12	macUserCount	0.492	9	HTTP5XX	0.212	18	permitCount
0.058	9	HTTP5XX	0.484	17	blockCount	0.192	14	foreignIPCount
0	5	uniqueUserCount	0.354	14	foreignIPCount	0.177	15	facultyCount
0	6	uniqueIPCount	0.287	18	permitCount	0.174	17	blockCount
0	1	wday	0.215	16	studentCount	0.141	16	studentCount

Table 16: PAM Feature Ranking

Dataset 1 (3 hours)			Dataset 2 (2 days)			Dataset 3 (2 weeks)		
Rank	Ordinal #	Feature	Rank	Ordinal #	Feature	Rank	Ordinal #	Feature
0.3892	3	postCount	0.6652	8	HTTP4XX	0.4557	8	HTTP4XX
0.3728	8	HTTP4XX	0.5706	2	dhour	0.4199	3	postCount
0.2637	13	DHCPDiscover	0.4653	13	DHCPDiscover	0.384	13	DHCPDiscover
0.123	14	foreignIPCount	0.4584	3	postCount	0.2668	14	foreignIPCount
0.1082	2	dhour	0.3496	17	blockCount	0.2365	2	dhour
0.1039	6	uniqueIPCount	0.3215	5	uniqueUserCount	0.2337	6	uniqueIPCount
0.1029	17	blockCount	0.3193	6	uniqueIPCount	0.2319	5	uniqueUserCount
0.1021	15	facultyCount	0.3165	14	foreignIPCount	0.2231	7	HTTP2XX
0.0664	16	studentCount	0.3156	7	HTTP2XX	0.1956	11	activeSyncUserCount
0	18	permitCount	0.314	15	facultyCount	0.1864	12	macUserCount
0	4	getCount	0.2766	12	macUserCount	0.1793	4	getCount
0	5	uniqueUserCount	0.2718	4	getCount	0.1789	10	owaUserCount
0	9	HTTP5XX	0.2387	11	activeSyncUserCount	0.1137	18	permitCount
0	7	HTTP2XX	0.2358	10	owaUserCount	0.1079	17	blockCount
0	12	macUserCount	0.2344	18	permitCount	0.1062	15	facultyCount
0	11	activeSyncUserCount	0.1632	16	studentCount	0.1032	16	studentCount
0	10	owaUserCount	0.1079	9	HTTP5XX	0.0878	9	HTTP5XX
0	1	wday	0.0542	1	wday	0.0592	1	wday

Table 17: Rule-based Feature Ranking

5.4.3 Split-level Clustering

Split-level clustering was introduced to simulate a non-linear method of classifying the dataset. PAM is used to partition the dataset into three clusters. Each of the resulting clusters is then partitioned using PAM to create three clusters which are labeled *green*, *yellow*, or *red* according to their respective cluster score. The resulting nine clusters are combined according to their labeled color and used to create a dataset which is then used for evaluation purposes of the deep learning algorithm using multiple hidden layers. Figure 21 depicts the process used by the split-level clustering method.

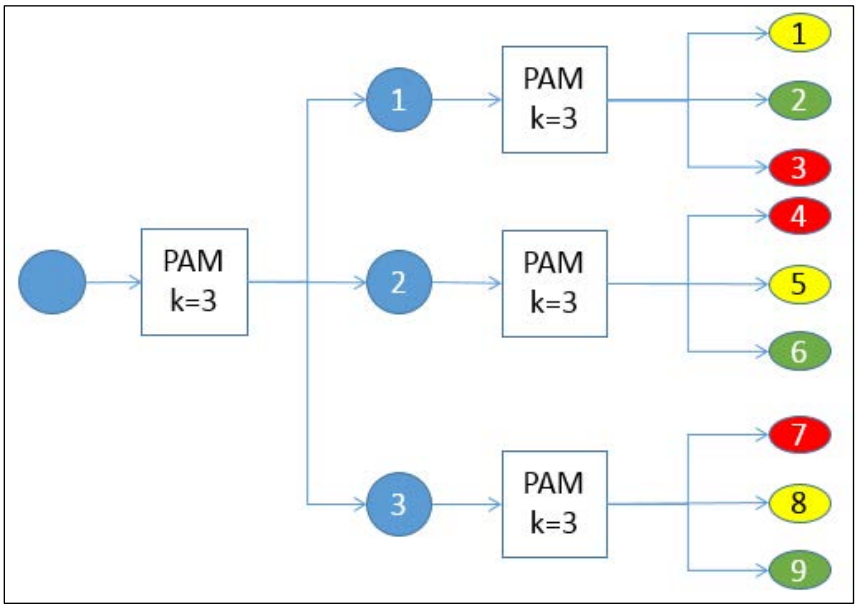

Figure 21: Split-Level Clustering Process

The split-level concept seems similar to hierarchical clustering; however it is not really for several reasons. First, the algorithm used is Partitioning among Medoids (PAM) which is a partitioning algorithm. Second, the number of clusters in hierarchical clustering is determined by the height in the tree, whereas the number of clusters is

specified for PAM. There are two types of hierarchical clustering methods. Agglomerative is a bottom-up technique which starts with every instance in its own cluster, and then merges the clusters until they are all in a single cluster. Divisive, a top-down strategy, starts with all the instances in one cluster and then subdivides the cluster until each instance is in its own cluster. In the split-level method, the height is constant, and the final number of clusters is controlled by *k* used in the second level which should match the levels of user activity used for classification.

5.5 Supervised Learning Results

Supervised learning was performed using the h2o deep learning algorithm [h2o17] to train and test the model using each of labeled datasets created during unsupervised learning. The datasets were split into training and test sets comprising 70% and 30% of the data respectively. The training set was used solely to train the neural network, and the test set was reserved for testing and evaluation purposes. The parameters for the h2o deep learning algorithm are the number of epochs, the activation function, and the hidden layer topology. The hidden layer parameter is a vector containing the number of neurons for each hidden layer. The activation function used was the Hyperbolic Tangent, and the number of epochs used for this research was 1000. The optimal number of epochs was determined through experimentation using 100, 1,000, and 10,000 epochs taking into account the accuracy and time to train the model.

Deep learning tests were conducted using the PAM labeled datasets varying the number of hidden neurons from 2 to 20 in a single hidden layer. The results shown in Table 18

are from a single test on each dataset. The deep learning algorithm automatically dropped the *wday* feature in the three-hour dataset because the value was constant.

Deep Learning (h2o) - PAM	Dataset 1	Dataset 2	Dataset 3
Time Window	3 hours	48 hours	2 weeks
Time slot size	5 min	5 min	5 min
Raw events	995,701	12,786,858	102,993,636
Instances	176	2876	18896
Train / Test Split	70% / 30%	70% / 30%	70% / 30%
Training Instances	124	2015	13229
Test Instances	52	861	5667
Features	18	18	18
Epochs	1000	1000	1000
Hidden Layer topology	1 layer 2 neurons	1 layer 2 neurons	1 layer 6 neurons
Accuracy	98.077%	99.768%	99.012%
Precision	100%	99.762%	98.544%
Recall	96.774%	99.762%	99.112%
F-score	98.361%	99.762%	98.827%
Time to train (s)	1.64	5.672	53.686
Time to test (s)	0.094	0.093	1.145

Table 18: Deep Learning Results using PAM Labeled Data

The resulting confusion matrices for each of the tests are shown in Table 19. There were no false negatives for Datasets 1 and 2. There were ten false negatives for the larger dataset where only two were classified as normal. There was only one false positive for

Datasets 1 and 2. The larger dataset resulted in seventeen false positives where only four were classified as critical.

PAM		Predicted								
		Dataset 1			Dataset 2			Dataset 3		
		Green	Red	Yellow	Green	Red	Yellow	Green	Red	Yellow
Actual	Green	30	1	0	420	0	1	1895	4	13
	Red	0	15	0	0	214	0	2	1907	8
	Yellow	0	0	6	1	0	225	26	7	1809

Table 19: Deep Learning Confusion Matrices for PAM Labeled Data

The single layer topology analysis in Table 20 shows the deep learning results for Dataset 1 of the various neuron configurations while holding all other parameters constant. There is no difference in performance with two, three, or four neurons. Adding a fifth neuron allowed the model to achieve 100% accuracy, precision, and recall.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	98.07692	100	96.77419	98.36066	1.62549	1000
3	98.07692	100	96.77419	98.36066	1.595507	1000
4	98.07692	100	96.77419	98.36066	1.627252	1000
5	100	100	100	100	1.622287	1000
6	100	100	100	100	1.626863	1000
7	100	100	100	100	2.590651	1000
8	98.07692	100	96.77419	98.36066	2.622711	1000
9	100	100	100	100	2.637034	1000
10	100	100	100	100	2.654797	1000
11	98.07692	100	96.77419	98.36066	2.630324	1000
12	98.07692	100	96.77419	98.36066	2.650092	1000
13	100	100	100	100	2.628364	1000
14	100	100	100	100	2.587347	1000
15	100	100	100	100	2.760165	1000
16	100	100	100	100	3.669263	1000
17	100	100	100	100	3.664567	1000
18	100	100	100	100	3.704171	1000
19	100	100	100	100	3.675554	1000
20	100	100	100	100	3.693978	1000

Table 20: Single Layer Topology Analysis PAM Labeling Using Dataset 1

The single layer topology analysis for Dataset 2 is shown in Table 21. Two hidden neurons produced the best accuracy for this dataset. Adding more neurons had no effect and in some cases reduced the accuracy slightly. The total time to train the model was only 5.69 seconds.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	99.76771	99.76247	99.76247	99.76247	5.689466	1000
3	99.76771	99.76247	99.76247	99.76247	6.691822	1000
4	99.76771	99.76247	99.76247	99.76247	5.658723	1000
5	99.76771	99.76247	99.76247	99.76247	6.726224	1000
6	99.76771	99.76247	99.76247	99.76247	6.728325	1000
7	99.65157	99.7619	99.52494	99.64328	6.728092	1000
8	99.76771	99.76247	99.76247	99.76247	6.720923	1000
9	99.76771	99.76247	99.76247	99.76247	6.732945	1000
10	99.76771	99.76247	99.76247	99.76247	6.709315	1000
11	99.76771	99.76247	99.76247	99.76247	6.734241	1000
12	99.65157	99.7619	99.52494	99.64328	6.750747	1000
13	99.76771	99.76247	99.76247	99.76247	6.738861	1000
14	99.76771	99.76247	99.76247	99.76247	6.741006	1000
15	99.76771	99.76247	99.76247	99.76247	6.750326	1000
16	99.76771	99.76247	99.76247	99.76247	6.759914	1000
17	99.76771	99.76247	99.76247	99.76247	11.77244	1000
18	99.76771	99.76247	99.76247	99.76247	6.731941	1000
19	99.76771	99.76247	99.76247	99.76247	11.79676	1000
20	99.65157	99.7619	99.52494	99.64328	11.905	1000

Table 21: Single Layer Topology Analysis PAM Labeling Using Dataset 2

The single layer topology analysis results for the largest dataset are shown in Table 22. Ten hidden neurons produced the highest accuracy (99.33%) and took 170 seconds to train the model. A single layer of six hidden neurons yielded an accuracy of 99.01% while only taking 54.5 seconds for training.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	98.4648	97.82496	98.79707	98.30861	45.49428	1000
3	98.39421	97.31544	98.58787	97.94752	61.85791	1000
4	98.69419	97.48718	99.42469	98.4464	76.9848	1000
5	98.85301	97.63739	99.42469	98.52293	91.35115	1000
6	99.01182	98.54394	99.11088	98.8266	53.50667	1000
7	99.15299	98.85297	99.16318	99.00783	83.10124	1000
8	99.04711	98.64795	99.21548	98.9309	123.0332	1000
9	99.22358	99.21301	98.90167	99.0571	148.0723	1000
10	99.32945	99.11504	99.58159	99.34777	170.2406	1000
11	99.15299	98.49585	99.32008	98.90625	126.1565	1000
12	99.27651	99.11227	99.26778	99.18997	134.5883	1000
13	99.02947	98.69995	99.26778	98.98305	120.8118	1000
14	98.94124	98.54318	99.05858	98.80021	155.6615	1000
15	99.15299	98.95725	99.26778	99.11227	152.4336	1000
16	98.62361	97.43458	99.32008	98.3683	69.93184	1000
17	99.13534	98.29809	99.68619	98.98728	108.6168	1000
18	99.15299	98.90568	99.26778	99.0864	245.4264	1000
19	99.20593	99.21301	98.90167	99.0571	273.0635	1000
20	99.06476	98.24652	99.63389	98.93534	117.8956	1000

Table 22: Single Layer Topology Analysis PAM Labeling Using Dataset 3

Deep learning tests were conducted using the Rule-based labeled datasets varying the number of hidden neurons from 2 to 20 in a single hidden layer. The results shown in Table 23 are from a single test on each dataset. The time to train the model using the Rule-based labeled datasets was significantly longer than the PAM labeled datasets. For example, the largest rule-based dataset took 90.5 seconds to train compared to the comparable PAM labeled dataset which took 53.7 seconds. The accuracy of the Rule-based datasets was also lower than the accuracy with the PAM labeled datasets.

Deep Learning (h2o) – Rule Based	Dataset 1	Dataset 2	Dataset 3
Time Window	3 hours	48 hours	2 weeks
Time slot size	5 min	5 min	5 min
Raw events	995,701	12,786,858	102,993,636
Instances	176	2876	18896
Train / Test Split	70% / 30%	70% / 30%	70% / 30%
Training Instances	125	2015	13228
Test Instances	51	861	5668
Features	18	18	18
Epochs	1000	1000	1000
Hidden Layer topology	1 layer 3 neurons	1 layer 4 neurons	1 layer 5 neurons
Accuracy	82.353%	92.567%	97.971%
Precision	82.609%	98.129%	98.791%
Recall	82.609%	98.951%	99.09%
F-score	82.609%	98.539%	98.94%
Time to train (s)	1.62	12.89	90.54
Time to test (s)	0.1	0.102	1.15

Table 23: Deep Learning Results Using Rule-based Labeled Data

The resulting confusion matrices for each of the tests are shown in Table 24. Looking at the red cluster, we can see there were no false negatives predicted for Dataset 1; thirty-nine false negatives occurred while classifying Dataset 2, and only fourteen false negatives were encountered classifying the test set of Dataset 3.

Rule-based		Predicted								
		Dataset 1			Dataset 2			Dataset 3		
		Green	Red	Yellow	Green	Red	Yellow	Green	Red	Yellow
Actual	Green	19	0	4	472	0	5	2941	0	27
	Red	0	7	0	1	73	38	0	1022	14
	Yellow	4	1	16	8	12	252	36	38	1590

Table 24: Confusion Matrices for Rule-based Labeled Data

The single layer topology analysis in Table 25 shows the deep learning results for Dataset 1 of the various neuron configurations while holding all other parameters constant. A single hidden layer with five neurons yielded an accuracy of 84.3% while classifying the test set of Dataset 1.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	78.43137	85	73.91304	79.06977	1.618708	1000
3	82.35294	82.6087	82.6087	82.6087	1.636856	1000
4	68.62745	64.28571	78.26087	70.58824	1.607999	1000
5	84.31373	84	91.30435	87.5	1.589796	1000
6	74.5098	74.07407	86.95652	80	1.569426	1000
7	74.5098	73.91304	73.91304	73.91304	2.61486	1000
8	76.47059	77.77778	91.30435	84	2.628524	1000
9	74.5098	76.92308	86.95652	81.63265	2.627736	1000
10	80.39216	77.77778	91.30435	84	2.611627	1000
11	84.31373	84.61538	95.65217	89.79592	2.613605	1000
12	78.43137	84	91.30435	87.5	2.577139	1000
13	74.5098	80.76923	91.30435	85.71429	2.658558	1000
14	68.62745	67.85714	82.6087	74.5098	2.614377	1000
15	70.58824	80	69.56522	74.4186	3.637561	1000
16	80.39216	80	86.95652	83.33333	3.626899	1000
17	78.43137	84	91.30435	87.5	3.623393	1000
18	74.5098	70	91.30435	79.24528	3.64017	1000
19	78.43137	79.16667	82.6087	80.85106	3.638308	1000
20	78.43137	76.66667	100	86.79245	3.655533	1000

Table 25: Single Layer Topology Analysis Rule-based Labeling Using Dataset 1

The single layer topology analysis in Table 26 shows the deep learning results using Dataset 2 for the different hidden neuron configurations. The configuration using eleven neurons in the single hidden layer yielded an accuracy of 95.47% with a training time of 28.1 seconds.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	83.15912	90.83821	97.69392	94.14141	7.732877	1000
3	88.96632	90.63098	99.37107	94.8	9.752759	1000
4	92.56678	98.1289	98.95178	98.53862	12.80058	1000
5	92.45064	95.73171	98.74214	97.21362	14.86402	1000
6	94.07666	97.10145	98.32285	97.70833	16.86723	1000
7	92.21835	95.52846	98.53249	97.00722	18.90082	1000
8	92.91521	96.88797	97.90356	97.39312	20.90594	1000
9	92.21835	97.3029	98.32285	97.81022	23.98935	1000
10	93.84437	97.1134	98.74214	97.921	26.00425	1000
11	95.47038	98.94515	98.32285	98.63302	28.06791	1000
12	92.68293	97.49478	97.90356	97.69874	31.15811	1000
13	92.91521	98.52008	97.69392	98.10526	33.16938	1000
14	93.37979	97.91667	98.53249	98.22362	35.20016	1000
15	90.47619	96.24217	96.6457	96.44351	37.2912	1000
16	92.79907	97.28033	97.48428	97.3822	40.37036	1000
17	94.88966	99.1453	97.27463	98.20106	42.32297	1000
18	92.91521	97.90356	97.90356	97.90356	44.38725	1000
19	92.21835	98.31579	97.90356	98.10924	46.56228	1000
20	91.86992	96.26556	97.27463	96.76747	48.50463	1000

Table 26: Single Layer Topology Analysis Rule-based Labeling Using Dataset 2

The single layer topology analysis in Table 27 shows the deep learning results using Dataset 3 for the different hidden neuron configurations. The configuration using five neurons in the single hidden layer yielded an accuracy of 97.97% with a training time of 90.5 seconds.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	82.56881	93.58842	91.47574	92.52002	44.46491	1000
3	80.5928	95.54876	94.74394	95.14465	58.71984	1000
4	97.93578	98.98854	98.92183	98.95517	75.91339	1000
5	97.97107	98.79073	99.0903	98.94029	90.54283	1000
6	97.81228	98.6237	98.98922	98.80612	106.6526	1000
7	97.65349	98.78419	98.55121	98.66757	118.7537	1000
8	95.92449	98.00203	97.50674	97.75376	134.1439	1000
9	97.67114	99.08132	98.11321	98.59489	149.4514	1000
10	97.52999	98.81757	98.55121	98.68421	165.662	1000
11	97.72406	98.6541	98.78706	98.72054	177.9252	1000
12	97.31828	98.6811	98.31536	98.49789	195.1906	1000
13	96.82428	98.41216	98.1469	98.27935	152.7024	1000
14	97.63585	98.48739	98.71968	98.6034	226.6637	1000
15	95.67749	97.89617	97.2035	97.54861	133.1717	1000
16	95.48342	97.36842	97.2372	97.30276	104.6774	1000
17	95.78335	98.05924	97.03504	97.54445	104.4585	1000
18	96.94778	98.01747	98.28167	98.14939	175.1375	1000
19	97.23006	98.38111	98.28167	98.33137	171.8929	1000
20	96.64785	98.56899	97.47305	98.01796	128.1003	1000

Table 27: Single Layer Topology Analysis Rule-based Labeling Using Dataset 3

5.5.1 Neural Network Topology

Defining the neural network topology must be completed prior to training. Defining the input and output layers are relatively straightforward. For the experiments conducted in this research, eighteen neurons were used for the input layer, one neuron for each feature. Three neurons were used for the output layer, one neuron for each possible classification. Generally, there is no best practice for selecting the number of hidden layers or neurons, but these values should not be arbitrarily selected [Han06]. As the number of neurons increases, the neural network's hypothesis function becomes more complex. Using more

than one hidden layer allows for implementing a more complex function on the data. An overly complex hypothesis function will learn the function of the underlying data including any noise resulting in poor generalization. This is known as overfitting. Finding the hypothesis with the minimum training error will result in the best fit. Conversely, if the hypothesis function is less complex than the data, the generalization error will be high. This is known as under-fitting. Selecting the number of hidden layers and neurons for each layer was accomplished by varying the number of hidden neurons in each layer and examining the results.

As the patterns and relationships in the data become more complex, the required number of hidden layers needed to learn a nonlinear relationship increase. In order to simulate such a nonlinear equation, testing of multiple hidden layer configurations was accomplished using the two split-level labeled datasets.

The optimal number of layers was determined by running tests on a single layer with 2 to 20 neurons. The number of neurons that produced the greatest accuracy or f-score with the least amount of training time was then held constant while varying the second layer of neurons from 2 to 20. Finally, a third hidden layer was added using the optimal number of neurons identified in the previous two runs. The layer that produced the greatest accuracy or f-score was selected as the most optimum hidden layer configuration.
The topology analysis for the first hidden layer using Dataset 2 is shown in Table 28. The configuration with 16 neurons produced an accuracy of 97.2% with a training time of 39.5 seconds.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	68.21346	86.09865	67.60563	75.73964	7.806764	1000
3	82.36659	87.67606	87.67606	87.67606	9.742069	1000
4	90.37123	94.61538	86.61972	90.44118	12.78492	1000
5	85.61485	93.10345	85.56338	89.17431	14.84607	1000
6	92.4594	95.65217	92.95775	94.28571	16.87575	1000
7	91.41531	97.70115	89.78873	93.57798	18.91523	1000
8	92.69142	96.71533	93.30986	94.98208	21.95703	1000
9	90.95128	95.20295	90.84507	92.97297	24.0757	1000
10	95.0116	97.21254	98.23944	97.72329	26.15671	1000
11	95.35963	97.5	96.12676	96.80851	28.05099	1000
12	94.89559	97.84946	96.12676	96.98046	30.07342	1000
13	96.2877	98.21429	96.83099	97.51773	32.17703	1000
14	95.12761	98.57651	97.53521	98.0531	35.15127	1000
15	96.40371	99.27536	96.47887	97.85714	37.26569	1000
16	97.21578	97.87986	97.53521	97.70723	39.5149	1000
17	94.77958	98.21429	96.83099	97.51773	41.33255	1000
18	96.51972	97.87234	97.1831	97.5265	44.33592	1000
19	95.93968	97.90941	98.94366	98.42382	46.46972	1000
20	96.2877	99.28315	97.53521	98.40142	49.58842	1000

Table 28: Layer 1 Topology Analysis Split Level Using Dataset 2

The results from the next step using two hidden layers with the first layer having 16 neurons while varying the number of neurons in the second layer from 2 to 20 are shown in Table 29. The hidden layer topology of 16, 15 neurons yielded an accuracy of 97.8%. The two layer hidden layer topology is optimal because it yielded a greater accuracy than the single layer topology. The gain was 0.6% accuracy at the cost of 20 seconds of additional training time.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
16,2	95.24362	96.86411	97.88732	97.37303	39.38839	1000
16,3	94.77958	98.88476	93.66197	96.20253	40.43865	1000
16,4	94.19954	98.52399	94.01408	96.21622	41.38389	1000
16,5	94.77958	96.55172	98.59155	97.56098	44.58092	1000
16,6	94.43155	98.92473	97.1831	98.04618	45.52426	1000
16,7	94.66357	97.79412	93.66197	95.68345	47.56215	1000
16,8	94.19954	98.50746	92.95775	95.65217	48.66051	1000
16,9	94.08353	98.9011	95.07042	96.94794	50.62686	1000
16,10	95.59165	97.53521	97.53521	97.53521	51.65276	1000
16,11	94.19954	96.40288	94.3662	95.37367	53.69832	1000
16,12	96.63573	98.57651	97.53521	98.0531	54.66463	1000
16,13	94.43155	98.56631	96.83099	97.69094	56.81233	1000
16,14	94.89559	98.20789	96.47887	97.3357	58.80139	1000
16,15	97.79582	98.93617	98.23944	98.58657	59.87349	1000
16,16	95.70766	98.92473	97.1831	98.04618	60.78947	1000
16,17	95.35963	98.20789	96.47887	97.3357	62.87584	1000
16,18	96.17169	98.25175	98.94366	98.59649	64.77319	1000
16,19	96.2877	97.51773	96.83099	97.17314	66.00017	1000
16,20	95.35963	98.1685	94.3662	96.2298	67.91216	1000

Table 29: Layer 2 Topology Analysis Split Level Using Dataset 2

The topology analysis for the first hidden layer using Dataset 3 is shown in Table 30. The configuration with 17 neurons produced an accuracy of 94.2% with a training time of 200.8 seconds. The configuration with 15 neurons produced a lower accuracy of 91.5%, but with a training time of 66 seconds.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
2	75.28229	80.64153	65.67369	72.39202	40.3704	1000
3	81.82781	88.12241	77.86434	82.6764	50.46101	1000
4	82.16302	88.85487	66.49863	76.06815	73.05516	1000
5	84.26253	91.33017	70.48579	79.56544	86.20817	1000
6	85.33874	89.45704	77.77269	83.20667	54.54736	1000
7	89.64361	96.15385	80.20165	87.45627	112.7395	1000
8	91.46083	90.66729	89.04675	89.84971	104.4871	1000
9	92.87227	94.46849	90.00917	92.18493	112.6339	1000
10	88.58504	95.50309	77.86434	85.78642	55.6617	1000
11	91.0374	93.83768	85.83868	89.66012	132.2834	1000
12	93.6662	94.52902	91.06324	92.76377	168.9416	1000
13	92.02541	95.82689	85.2429	90.22556	91.40125	1000
14	93.08398	95.37263	89.73419	92.46753	110.8291	1000
15	91.51376	97.53351	83.36389	89.89375	66.049	1000
16	93.38391	95.13956	90.60495	92.8169	108.8855	1000
17	94.21313	95.02605	91.93401	93.45446	208.7715	1000
18	93.52505	95.01677	90.87993	92.90232	81.25883	1000
19	93.08398	97.10579	89.18423	92.97659	95.57244	1000
20	91.42555	96.59898	87.21357	91.66667	60.82192	1000

Table 30: Layer 1 Topology Analysis Split Level Using Dataset 3

The first test conducted selected the neuron configuration that yielded the most accurate results with the best time to train. The results from the next step using two hidden layers with the first layer having 15 neurons while varying the number of neurons in the second layer from 2 to 20 are shown in Table 31. The hidden layer topology of 15, 6 neurons yielded an accuracy of 95%.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
15,2	89.94354	90.35748	88.0385	89.18292	55.71038	1000
15,3	92.83698	95.5611	87.80935	91.52138	121.9721	1000
15,4	92.16655	96.35711	86.06783	90.92229	127.911	1000
15,5	92.74877	95.74988	89.82585	92.69331	79.05914	1000
15,6	95.04234	96.14469	92.57562	94.32641	127.92	1000
15,7	93.6662	96.35468	89.64253	92.87749	93.42123	1000
15,8	90.73747	95.1952	87.16774	91.00478	54.55815	1000
15,9	92.87227	97.81818	86.29698	91.6971	102.499	1000
15,10	94.54834	95.76996	92.34647	94.02706	118.8778	1000
15,11	93.03105	96.18022	90.00917	92.99242	80.15039	1000
15,12	93.27805	93.36406	92.8506	93.10662	90.35095	1000
15,13	93.70148	96.15946	90.65078	93.3239	91.3145	1000
15,14	93.47212	94.37618	91.52154	92.92694	81.17394	1000
15,15	94.38956	94.62366	92.75894	93.68202	78.08645	1000
15,16	93.20748	98.02083	86.25115	91.76012	134.048	1000
15,17	92.58998	96.4018	88.40513	92.23046	98.4739	1000
15,18	90.4199	94.677	83.95967	88.99684	82.44917	1000
15,19	91.72548	97.19189	85.65536	91.05968	132.3058	1000
15,20	93.22512	96.0333	89.87168	92.85038	121.7948	1000

Table 31: Layer 2 Topology Analysis Split Level Using Dataset 3

The results of the third layer topology analysis with the first and second layer containing 15 and 6 neurons are displayed in Table 32. The best three layer configuration consists of 15, 6, and 12 neurons, yielding an accuracy of 93.1% and f-score of 91.8% with a training time of 199.3 seconds. The two layer hidden layer topology is optimal because it yielded a greater accuracy than both the single layer and third layer topology.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
15,6,2	89.09668	93.79347	80.33914	86.54653	61.84188	1000
15,6,3	91.2844	92.05896	88.72594	90.36173	192.185	1000
15,6,4	90.98447	96.35417	84.7846	90.1999	135.1145	1000
15,6,5	92.37826	95.91022	88.13016	91.85574	71.93221	1000
15,6,6	91.09033	92.68409	89.41338	91.01936	61.87233	1000
15,6,7	90.73747	94.76598	86.29698	90.33341	62.75709	1000
15,6,8	89.27311	94.66595	80.52246	87.02328	90.29544	1000
15,6,9	91.60198	95.11947	85.74702	90.19041	78.2987	1000
15,6,10	92.13126	95.47966	87.12191	91.10951	85.21178	1000
15,6,11	92.44884	95.34884	88.31347	91.69641	204.4548	1000
15,6,12	93.08398	95.17717	88.63428	91.78927	199.3481	1000
15,6,13	90.98447	96.06918	84.0055	89.63325	113.7216	1000
15,6,14	92.41355	96.36364	87.44271	91.68669	114.802	1000
15,6,15	91.84898	94.79219	86.75527	90.59584	73.98066	1000
15,6,16	91.91955	94.10906	88.58845	91.26534	111.7607	1000
15,6,17	87.35004	94.16058	76.8561	84.63285	78.048	1000
15,6,18	88.3204	95.09583	77.31439	85.28817	91.31433	1000
15,6,19	91.72548	97.71739	82.40147	89.40825	168.7535	1000
15,6,20	88.79675	89.33398	84.83043	87.02398	58.71918	1000

Table 32: Layer 3 Topology Analysis Split Level PAM Dataset 3

The second test used the 17 neuron configuration which yielded the most accurate results in the single layer test. Examining the results of the second layer topology analysis in Table 33, we can see a network topology configuration of two hidden layers with 17 neurons in each layer is the optimal choice yielding an accuracy of 96.3% and f-score of 96.2%. The best one layer configuration with 17 neurons was 94.2% accuracy and f-score of 93.5%. The best three layer configuration with 17, 17, and 4 neurons yielded an accuracy of 94.4% and f-score of 94.0%.

Hidden	Accuracy	Precision	Recall	F-score	Time (s)	Epochs
17,2	93.27805	93.32088	91.56737	92.43581	126.6879	1000
17,3	93.89555	96.6325	90.74244	93.59489	117.1308	1000
17,4	93.04869	96.52393	87.80935	91.96064	199.4389	1000
17,5	91.88426	92.71081	89.18423	90.91334	68.58157	1000
17,6	93.80734	94.70949	91.88818	93.27751	120.3014	1000
17,7	93.80734	95.21277	90.23831	92.65882	76.17656	1000
17,8	93.13691	96.35214	90.78827	93.48749	89.70245	1000
17,9	93.34862	97.4026	89.36755	93.21224	182.6643	1000
17,10	93.10162	95.19417	89.87168	92.45639	123.041	1000
17,11	93.61327	97.08981	88.68011	92.69461	89.70555	1000
17,12	94.68948	96.36015	92.20898	94.23888	99.12012	1000
17,13	92.73112	97.59959	87.5802	92.31884	142.6929	1000
17,14	92.29005	93.25527	91.24656	92.23998	77.43766	1000
17,15	93.82498	97.13439	90.10082	93.4855	76.25334	1000
17,16	93.33098	97.76536	88.22181	92.74874	113.4737	1000
17,17	96.2597	97.95627	94.45463	96.17359	202.8807	1000
17,18	92.16655	94.21129	89.50504	91.79788	84.26322	1000
17,19	95.2717	98.04209	91.79652	94.81657	174.5046	1000
17,20	93.36627	97.39929	87.53437	92.20372	112.0047	1000

Table 33: Layer 2 Topology Analysis Split Level PAM Dataset 3

5.5.2 Additional observations

Scalability is achieved using the time slot to model the data. For example, Dataset 1 represented a total of 995,701 events in 176 instances. Time to test was 0.094 seconds using 52 instances. Dataset 2 was created from 12,786,858 events and was reduced to 2,876 instances. Time to test was 0.093 seconds using 861 instances. The number of instances increased by a factor of 16, but the time to test was faster by 0.001 seconds. Dataset 3 was comprised of 18,896 instances and represented 102,993,636 raw events. Time to test was 1.145 seconds. The time to test Dataset 3 was 12 times that of Dataset 1

where Dataset 3 was 363 times larger than Dataset 1. It is evident that increasing the amount of data increases the time to test linearly.

Including additional log files will not increase the number of instances in the dataset, but instead will only add columns equal to the number of features extracted from each log file added.

5.5.3 Implementation considerations

There are several factors that should be considered before training the model whether it is the initial training or subsequent feedback sessions. First, the security analyst will need a tool for examining or discovering suspicious patterns in the log data. The PAM clustering method used in this research does not serve as such a tool.

Additionally, each training session should use current data that contains a proportionate number of examples for each class. There are a number of methods that can be used to obtain attack training data. The easiest method is to use data gathered during a real breach. Another method is to use Honey Pots, systems which are designed to ferret out hackers and learn new methods. Logs gleaned from penetration or vulnerability scans can also be a valuable source of log attack data. Lastly, existing data can be programmatically modified to represent potential incidents or attacks.

Over time user activity patterns change, and new patterns may ensue. Also, existing features may have been overlooked, initially deemed not relevant, or introduced through

the procurement of new computer system. As a result, the performance of the model will eventually degrade and become unacceptable. In this event, features should be re-evaluated for relevance prior to retraining the model with a fresh set of log data.

For subsequent training sessions, the security analyst can use logs that were manually marked as suspicious or attack through normal daily investigations. When there are a sufficient number of examples, they can be added to the initial dataset and used to retain the model.

Chapter 6

CONCLUSION AND FUTURE WORK

The results of the experiments conducted in this thesis demonstrate that a classified dataset with a proportional set of examples trained with the Deep Learning algorithm can accurately detect abnormal activity. This method allows for multiple log source types to be aligned using a sliding time window and provides a scalable solution which is a much-needed feature.

In a typical enterprise environment, the amount of log data processed could vary from several hundred gigabytes to a terabyte daily. The prototype developed in this research was relatively small consisting of a set of eighteen features from three different log source types totaling approximately twenty-five gigabytes in size. This research demonstrated the prototype could very accurately model low complexity data with a shallow network. However, the complexity of the data increases as more log sources and features are introduced. This research demonstrated that highly complex data could be accurately modeled using a deep neural network.

Detecting a cyber attack is just the beginning of a long, complicated investigative process. The security analyst may need to perform risk mitigation actions, such as blacklisting originating source IP's and locking accounts. Logs files need to be examined to identify any compromised accounts, originating IP's, and all resources accessed by the attacker. All related activities should be collected and examined several weeks or even months before the detected event. Potential areas of future work are automatic correlation

and analysis of the log data from cyber attacks. Additional machine learning algorithms and analysis required for automatic correlation can put a strain on computing resources depending on the volume of data to be searched and velocity of the log data being collected. Additional areas of future work include building a distributed computing implementation such as Hadoop with terabytes of log data.

REFERENCES

Print Publications:

[Abad03]
Abad, C., Taylor, J., Sengul, C., Yurcik, W., Yuanyuan Zhou, & Rowe, K. “Log correlation for intrusion detection: A proof of concept.” ASCAC ’03 Proceedings of the 19th Annual Computer Security Applications Conference. December, 8, 2003, Las Vegas, NV, USA, pp. 255-264.

[Alpaydin14]
Alpaydın, Ethem. Introduction to machine learning. Cambridge, MA: MIT Press, 2014.

[Apte03]
Chid Apte. “The big (data) dig” OR/MS Today. 30.1 (Feb. 2003) pp. 24.

[Buczak16]
A. L. Buczak, and E. Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection." IEEE Communications Surveys & Tutorials 18.2 (2016): 1153-76. Print.

[Edwards15]
Edwards, Chris. "Growing Pains for Deep Learning." Communications of the ACM, vol. 58, no. 7, July 2015, pp. 14-16.

[Garcia12]
K. A. Garcia, et al. "Analyzing Log Files for Postmortem Intrusion Detection." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.6 (2012): 1690-704.

[Guyon06]
Guyon, Isabelle. “Feature Extraction: Foundations and Applications.” Springer-Verlag, 2006.

[Han06]
Han, Jiawei, and Micheline Kamber. Data mining concepts and techniques. San Francisco: Morgan Kaufmann, 2006.

[Jacobs09]
Jacobs, Adam. "The Pathologies of Big Data." Queue 7.6 (2009): 10:10,10:19.

[Kott13]
Kott, A., and C. Arnold. "The Promises and Challenges of Continuous Monitoring and Risk Scoring." Security & Privacy, IEEE 11.1 (2013): pp. 90-3.

[Kott14]
Kott, Alexander, Ananthram Swami, and Patrick Mcdaniel. "Security Outlook: Six Cyber Game Changers for the Next 15 Years." Computer 47.12 (2014): 104-06.

[Mahmood13]
Mahmood, T Mahmood, T., and U. Afzal. "Security Analytics: Big Data Analytics for Cybersecurity: A Review of Trends, Techniques and Tools". 2013 2nd National Conference on Information Assurance (NCIA). December, 11 2013, Rawalpindi, Pakistan. pp. 129-134.

[Ng15]
J. Ng, D. Joshi, and S. M. Banik. "Applying Data Mining Techniques to Intrusion Detection". Information Technology: New Generations (ITNG) 2015 Proceedings of the 12th International Conference on Information Technology. April 13, 2015, Las Vegas, NV, USA. pp. 800-801.

[Razzaq14]
Razzaq, Abdul, et al. Semantic Security Against Web Application Attacks. 254 Vol. Elsevier Inc, 2014.

[Sood13]
A. K. Sood, and R. J. Enbody. "Targeted Cyberattacks: A Superset of Advanced Persistent Threats." IEEE Security & Privacy 11.1 (2013): pp. 54-61.

[Valentan13]
Valentín, Kristián, and Michal MALY. "Network Firewall using Artificial Neural Networks." Computing & Informatics 32.6 (2013): 1312-27.

[Ye05]
Nong Ye, and T. Farley. "A Scientific Approach to Cyberattack Detection." Computer 38.11 (2005): pp. 55-61.

[Zhu02]
Y. Zhu and D. Shasha. "StatStream: Statistical monitoring of thousands of data streams in real-time." In VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases. August 20, 2002, Hong Kong SAR, China, pp. 358-369.

[Zhu03]
Zhu, Yunyue, and Dennis Shasha. "Efficient elastic burst detection in data streams." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03. August 24, 2003, Washington, DC, USA, pp. 336-345.

Electronic Sources:

[Droms97]
Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, http://www.rfc-editor.org/info/rfc2131, last revision March 1997, last accessed March 26, 2017.

[Garfinkel16]
Simson L. Garfinkel. "Digital Forensics", American Scientist, http://www.americanscientist.org/issues/id.16080,y.0,no.,content.true,page.1,css.print/issue.aspx, last accessed April 19, 2017.

[h2o17]
Deep Learning - H2O 3.10.4.4 documentation, http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html?highlight=deep%20learning#, last accessed March 18, 2017.

[Hallam-Baker96]
Phillip M Hallam-Baker, Extended Log File Format, http://www.w3.org/TR/WD-logfile.html, last revised March 23, 1996, last accessed April 13, 2016.

[Muncaster15]
Phil Muncaster. "Hackers Spend Over 200 Days Inside Systems Before Discovery.", Infosecurity Magazine. N.p., 24 Feb. 2015, https://www.infosecurity-magazine.com/news/hackers-spend-over-200-days-inside/. Last accessed April 18, 2017.

[Splunk17]
Download Splunk Enterprise for free, https://www.splunk.com/en_us/download/splunk-enterprise.html, last accessed March 26, 2017.

VITA

Glenn Lambert is currently an Operations Center Engineer with Availity, LLC and holds a Certified Associate in Project Management (CAPM)® Certification and a Bachelor of Science in Computer and Information Science from the University of North Florida. He has over 20 years of experience in the Information Technology industry and is currently responsible for monitoring critical production systems using a variety of tools including Splunk for log file analysis. He is a military veteran with 15 years of service in the Florida Army National Guard. He is also a member of the University of North Florida chapter of Upsilon Pi Epsilon.