Autocorrel II: Unsupervised Network Event Correlation Using Neural Networks

Nathalie Japkowicz

Reuben Smith

School of Information Technology and Engineering
University of Ottawa
Ottawa Ontario

Project Manager:

Contract Number: W7714-04-09128

Contract Scientific Authority: Maxwell Dondo and Peter Mason

The scientific or technical validity of this Contract Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

DEFENCE R&D CANADA - OTTAWA

Contract Report

Defence R&D Canada – Ottawa CR 2005-155

March 2005

Author

M. Dondo

Approved by

Dr. J. Lefebvre
Head/IO Section

Approved for release by

Dr. A. Ashley
Chief Scientist

Abstract

Network event correlation is the process where relationships between intrusion detection system alerts or other network events are discovered and reported. An accurate network event correlation system can enable the intrusion detection analyst to find important events more easily. We present a system that uses unsupervised machine learning algorithms to create an effective and maintenance-free way to do network event correlation. The system uses the autoassociator, a type of neural network architecture, to find the relationships between related network events hidden in a collection of unrelated data. We demonstrate our system using intrusion alerts generated by a Snort intrusion detection system and discuss the overall performance of our system.

This page intentionally left blank.

Executive summary

Autocorrel II: Unsupervised Network Event Correlation Using Neural Networks

Nathalie Japkowicz, Reuben Smith; Defence R&D Canada – Ottawa CR 2005-155;
DEFENCE R&D CANADA - OTTAWA; March 2005.

The research area of network event correlation aims to give context to events reported by different systems on the network. The context that the correlation system tries to discover can be any one of many different types of contextual information. For instance, the correlation system could report an alert from an intrusion detection system that says a particular service was accessed, because the service on other computers has recently suffered a number of buffer-overflow attacks across the internet. Or, the correlation system could report that an incorrectly formed IP packet flagged by the router is very similar to 500 other incorrectly formed packets flagged by the same router. These are both examples of network event correlation, since they try to make sense of events reported by systems on the network.

In our research we're concerned primarily with the task of finding correlations between different intrusion detection system alerts. We want to find relationships between alerts that indicate the motives and methods of a particular attack attempt against the network. Intrusion detection systems can be configured to report all of the steps of an attack, but they are often not capable of finding all of the alerts related to the attack or separating them from the massive amounts of unrelated alerts reported by the systems.

In our research we try to separate all of the alerts reported by an intrusion detection system into discrete clusters of alerts so that each cluster represents all — and exactly all — of the alerts for a particular attack attempt. For false alarms generated by the intrusion detection system, we want to cluster together related false alarms so that either the root cause of the false alarms can be understood and dealt with, or so that they can be collectively analyzed or ignored quickly.

To find alert correlations that would help discover a particular attack attempt, we use a two step correlation process. In the first step we model each alert as a number of distinguishing features of the alert then use a trained neural network called the autoassociator to place each alert into a cluster with its related alerts. In this first step we cluster together alerts that were produced by very similar network traffic, which often leads to a number of clusters that are only composed of one type of alert.

In the second correlation step we compute a number of statistics on the clusters found in the first step that help to highlight patterns in the clusters that might not

otherwise be obvious. Using these statistical measures of the clusters from the first step we're able to find correlations between related clusters. After this step each cluster produced by our system should represent one attack attempt.

The advantage of our system over other correlation systems is that our system is low maintenance or maintenance-free. An organization must maintain most other correlation systems by inputting knowledge such as the relationships between different types of alerts, the changing distribution of alerts reported on the network, or the rules that find patterns in new, predictable attacks. Our system is based on unsupervised machine learning, a type of machine learning algorithm that infers relationships from data without the need to train the algorithm with expertly labelled data.

In this document we report on a number of experiments that we've run to explain and test the behaviour of our system. We also have created an optimal set of clusters for a set of data to which we compare the results of our system. By performing a comparison between the clusters we'd like to see and the clusters we actually see, we're able to gauge our performance.

Table of contents

This page intentionally left blank.

1 Introduction

Correlating network events means finding context for a particular network event. This can mean finding events that are very similar to other events, it can mean finding an event that could only have happened because of another event, and it can mean relating outside information such as vulnerability information or network architecture to an event.

All of these types of correlation are useful. By finding other similar intrusion detection alerts an analyst can deal with alerts more quickly than if dealing with the alerts separately. If a router reports that it received a malformed packet that caused a system fault this information could be correlated to a vulnerability report for a network router, which might give an analyst the clue that the router has been compromised.

In our research we're concerned primarily with the task of finding correlations between different intrusion detection system alerts. We want to find relationships between alerts that indicate the motives and methods of a particular attack attempt against the network. Intrusion detection systems can be configured to report all of the steps of an attack, but they are often not capable of finding all of the alerts related to the attack or separating them from the massive amounts of unrelated alerts reported by the systems.

In our research we try to separate the alerts reported by an intrusion detection system into discrete clusters of alerts so that each cluster represents all the alerts for a particular attack attempt. For false alarms generated by the intrusion detection system, we want to cluster together related false alarms so that either the root cause of the false alarms can be understood and dealt with, or so that they can be collectively analyzed or ignored quickly.

Neural networks are a type of machine learning algorithm useful in storing an abstracted form of large amounts of data in a novel way. Instead of storing all of the original data unfiltered, neural networks are designed to store the trends of the data. The trends represented in a neural network are used to classify new data or to analyze new data.

The autoassociator is a specialized piece of neural network architecture which can be used to process data in an unsupervised way. The autoassociator's ability to learn from unstructured, unlabelled data makes it ideal to use for the intrusion detection alert data. Maintaining a supervised machine learning system can be costly because training data periodically must be replaced and updated, but the autoassociator doesn't require labelled data for training, so there is less need for maintenance.

The system we have produced uses the autoassociator to find correlations in the alerts we examine. For the autoassociator to understand the intrusion detection data we must encode each alert as a string of numbers from which the autoassociator can learn numerical trends. We give the details of the system we've produced in the section on our model, Section 3.

The system we've produced has two levels of correlation. We show how we encode alerts for the first level in Section 3.2.1, and we give the details of how the autoassociator learns from this data in Section 3.2.2. We interpret the autoassociator output as clusters using an algorithm we created to do this in Section 3.2.3. For the second correlation step we feed the clusters formed from the first step into a clustering algorithm. To do this we must summarize each of the clusters as a finite set of features. We discuss these features in depth in Section 3.3.1.

We perform a number of experiments on the Autocorrel II system in Section 4.2. We present the details and results of experiments such as the effect of using different data scaling algorithms, the effect of using a training dataset with the autoassociator, and the best combinations of parameters to choose for the two steps of our system. We also discuss the overall performance of our system, and how we calculated it, in Section 4.3.

Please note that the bulk of this final report is summary of the work we've completed in the progress reports for this research project. In fact much of the text for this report was pasted together from the experiments we completed and reported on in the progress reports. These progress reports are available as appendices, starting with the project proposal at Annex B.

2 Background

(Please note that parts of this section were taken verbatim from the Autocorrel I report [1] for DRDC.)

The IP protocol is described in sufficient detail by Stevens [2]. The Stevens book is necessary reading before advancing to the weightier network intrusion detection system material by Northcutt [3], relating directly to our task. We talk about the current state of intrusion detection system (IDS) research in Section 2.1. We discuss the general state of alert correlation in Section 2.1.3. We have a special report on machine learning-based alert correlation methods in Section 2.1.3.1.

We give detailed background information on artificial neural networks in Section 2.2, and we talk about a specific, relevant type of neural network, the autoassociator, in Section 2.2.6. We give some information on two clustering algorithm we use, the EM algorithm and self-organizing maps, in Section 2.3.

2.1 Intrusion Detection Research

Intrusion detection systems (IDSs) are systems which report packets or connections that are considered suspicious by some set of rules, or by statistical analysis against some pre-defined set of normal traffic. They exist to solve the problem of attackers penetrating a target system of an unknowing system administrator. IDSs can be deployed inside or outside the network's firewalls depending on the type and level of detail of an attack that an administrator wishes to see. IDSs are generally considered to be low-level tools in analyzing network traffic because of the vast number of possible attacks they report, and because they do not indicate to the analyst the context of the supposed attack.

Richard Steven's TCP Illustrated [2] volumes are a good starting point for those interested in the raw workings of the TCP protocol. Richard Bejtlich [4] wrote an introduction on what is expected of an intrusion detection analyst when analyzing traffic, and on how intrusion detection systems can give a false sense of security to those who rely too heavily on them. Bejtlich's paper advocates the use of TCPDump [5] for the intrusion detection analyst to obtain a layer lower than the output presented at the IDS console.

Steven Northcutt wrote a book on intrusion detection called Network Intrusion Detection: An Analyst's Handbook [3], where he presents the basic problems motivating intrusion detection, and the most applicable solutions to the problems. He draws on his experience of creating the Shadow IDS [6] and imparts his knowledge in the handbook. He also provides information on how most IDS vendors deal with the problem of intrusion alert correlation. It is important to note that Northcutt does not treat the problem of alert correlation in the light of machine learning.

We use the terms *intrusion detection* and *network intrusion detection* interchangeably in this report, although they are not always synonymous in research by other authors. Historically, *intrusion detection* implies host-based or network-based intrusion detection, and *network intrusion detection* implies only the latter. The former tends to refer to detection methods that use characteristics of the host, and the latter tends to refer to detection methods that examine network traffic.

2.1.1 Deployed Intrusion Detection Systems

EMERALD [7] is a distributed, scalable, hierarchial, customizable network intrusion monitor. It can compose other intrusion detect systems to correlate threat warnings between different systems. At the time of writing this paper, the updaters of this system had not released their correlation unit to the public. EMERALD explicitly divides statistical analysis (anomaly detection) from signature-based analysis (misuse detection), and recombines the results at a higher level.

Snort [8] is a lightweight, open source intrusion detection system, which doesn't do TCP or IP defragmentation. We don't consider this defragmentation feature important for our research, because if IP packets are fragmented to such granularity that their malicious intent is hidden, then any anomaly-based system will see these packets as anomalous. (Snort flags these packets, even though it's ruled-based, since this rule is turned on by default.)

ACID [9] is a web-based intrusion correlation console that offers only very simplistic correlation abilities, but which can interoperate easily with Snort. In ACID, the IDS analyst can correlate alerts by source or destination TCP port, or source or destination IP address. This single-variable analysis leads to problems when trying to detect attacks such as distributed denial of service attacks.

Shadow [6] is an IDS created by Stephen Northcutt et al. In his book [3], Northcutt mentions alert correlation, but only in a capacity similar to that found in ACID. He mentions correlation by time, as well as by the other single-variable attributes listed for ACID.

NetSTAT [10] is an IDS which focuses on *network intrusion detection* rather than *host intrusion detection*. Similar to EMERALD, NetSTAT is scalable and composable.

QuidSCOR [11] is an open-source IDS, though it requires a subscription from its publisher, Qualys Inc, to work correctly. It works with IDSs such as Snort to correlate alerts against Common Vulnerability and Exposure (CVE) information published by Mitre.org.

Intellitactics/NSM [12] and Network Flight Recorder (NFR) [13] are other IDS systems that purport to have correlation abilities, but their software isn't free to download, so their claims were not investigated more thoroughly.

2.1.2 Anomaly-based Detection Research

Anomaly-based intrusion detection research typically trains its systems on labelled network traffic, which contains no intrusions or abnormal traffic that are not labelled as such for the learning algorithm which processes the traffic. These systems are then tested against traffic containing a small proportion of intrusions to normal traffic. These systems exist in contrast to misuse detection systems, which typically employ hand-coded rules that must be updated continually to reflect the most current attacks being perpetrated against the network.

In "A Multiple Model Cost-Sensitive Approach for Intrusion Detection" (Wei Fan, Wenke Lee, Salvatore J. Stolfo, and Matthew Miller [14]) Fan *et al* try to classify network intrusions by seeing them as anomalies to normal network traffic. Rather than detect intrusions by matching incoming packets to an administrator-created signature, Lee and Stolfo's system [15] trains a machine learning algorithm to recognize non-intrusion traffic so that it can be discerned from intrusion traffic. New intrusions will not be classified as such in signature-based IDSs, so anomaly-based systems remedy this problem by design. Lee and Stolfo [15] explore both signature-based detection (also known as misuse-detection) and anomaly detection, and compare the results. In Fan et al. [14], the authors expand on this work by showing the system can be optimized with respect to operational costs of the IDS. In [15] and [14], the authors use the RIPPER [16] algorithm for classification of data from the DARPA [17] and KDD99-Cup [18] datasets.

In Eskin et al. [19], the authors build on the work in [15] and [14] to extend the work to unlabelled data — this refers to data that doesn't have each packet labelled as *normal* or *anomaly*. This new method leads to an unsupervised anomaly detection algorithm which can be deployed with minimal training effort. Eskin et al. [19] also generalize their approach so that any unsupervised machine learning algorithm can be used; they compare the results of a cluster-based estimation algorithm, the K-nearest neighbour algorithm, and the one class SVM algorithm. They find that the one class SVM algorithm wins out if higher false alarm rates are allowed, and they use ROC curves [20] to present their results.

2.1.3 Alert Correlation Research

A recent paper that reviews and criticizes the various alert correlation efforts by different groups is by Hätälä *et al* [21]. They review a number of correlation systems, most of which aren't based on machine learning. They also offer some accurate criticisms of the correlation systems they report on. Hätälä *et al* also discuss the *Intrusion Detection Message Exchange Format (IDMEF)* standard by the *Intrusion Detection Working Group (IDWG)* [22] which hopes to encourage the development and interoperability of alert correlation in deployed intrusion detection systems.

Ning and Cui [23] use an ideal of how network events should be correlated as the basis for their system, which they implement using correlation rules. Ning and Cui recognize that many existing IDSs tend to detect low-level events without being able to relate these events to the broader plan of the attacker. Their goal is to create hierarchies of alerts using prerequisites and consequences for each type of alert. They attain this goal using a rule-based system (notably in contract with a learning system) that assigns prerequisites and consequences to each type of alert, which in turn allows the analyst to quickly see the possible consequences of the most mundane of alerts. Their system runs on the output of existing IDS systems. They demonstrate the usefulness of their system by showing how their system can significantly reduce the number of false alarms reported by an IDS while negligibly reducing the valid alarms reported by the IDS. Their tool is most logically used to run as an offline forensic tool for mining old stored alerts after a new vulnerability is found.

Debar and Wespi [24] present an aggregation and correlation component (ACC) for an IDS. They use this correlation component to flag alerts as an original, a consequence, or a double alert. In this research, they present rule-based methods for attaining this goal. They concern themselves with implementation issues such as integrating their ACC with existing IDSs — namely the Tivoli Enterprise Console, and they suppose the use of the IDWG [22] format for their work — and they discuss other issues such as raising or lowering the priority of IDS alert to reflect the inferences of their ACC.

Valdes and Skinner [25] present a paper for using statistical techniques in the alert correlation problem. They present a system to fuse alerts together from heterogeneous network sensors into single, more easily understood alerts. They control how loosely their fusing algorithm works with a single system parameter. Ning *et al* [23] criticize their system as not being an effective correlation effort because often the groups of alerts that a correlation system should find are not tied together by underlying numerical similarities.

The approach of Valdes and Skinner is based on a layered approach to correlation. They first advocate reducing the number of alerts in a particular stream of alerts from a single sensor by grouping similar alerts. (This process is known as *threading*, though they don't name it as such in their paper.) Then they give details on how to fuse the results from different sensors into a single stream. And finally they show how to use their system to find correlations between the remaining clusters, where each correlation found represents a link between steps of an attack by an opponent.

2.1.3.1 Machine Learning-Based Alert Correlation

Julisch *et al* [26] wrote a paper that describes a method for alert correlation that uses a conceptual clustering algorithm of their own creation. They resolve the problem of training their system by relying on clustering algorithms rather than classification algorithms. They do require some training in their system, though, to train the system which alerts can be handled automatically, since their system explicitly has the goal to reduce the number of alerts presented to the administrator. The training of their system fits into the general framework they've created for intrusion detection system operation. They advocate having the administrator hand-modify the correlation system once per month to reflect the changes in the network environment. As well, in their system they require that the correlation system has information specific to the local network before initial operation of the system. They require a seasoned analyst to perform this initial configuration.

Dain and Cunningham [27] have presented another machine learning-based correlation system in their paper "Fusing a Heterogeneous Alert Stream into Scenarios". In this paper, Dain *et al* train machine learning algorithms such as neural networks and decision trees to recognize attack scenarios based on a novel list of features. Hätälä *et al* [21] criticize the Dain *et al* work for it's use of a simplistic dataset. The Dain *et al* research is only tested with a DEFCON conference dataset [28]. The use of this dataset simplifies the problem of alert correlation because attackers represented in the dataset are motivated by points in the competition and points are not awarded for stealthy attacks. As such, many of the attacks originate from a single IP address (Dain *et al* [27]).

We have a methodological criticism of the Dain *et al* work. In the explanation of their preparation of the dataset, they offer the detail that, "a script was written to randomly split the data into [training, evaluation and testing] files." This statement implies that care was not taken to ensure that all training and evaluation instances occurred before any testing instances, and in fact that it is very likely that many testing instances were derived from events occurring before training instances. This is a methodological problem because it doesn't model a real world realtime system (which their system is purported to be), where the future is truly unknown. As well, because the authors split up individual attack scenarios over the training and testing datasets, the system was trained on one half of an attack scenario then tested on a different part of the same attack scenario, which is implausible for any realistic training set. Lastly, even if the above weren't true, Dain and Cunningham would still have the problem that they trained and tested on the same dataset, implying that in any situation where their system would be deployed, significant work would have to be undertaken to hand-label a large set of training data. (In other words, to mimic the paper's test configuration in the real world an analyst would have to label half of the alerts in an attack before the system could classify the other half.)

Their excellent performance in the domain occurs because their tests were unfairly biased towards success. (Their system attained a performance level of over 99%. Another reason this figure is so high is the way they formulated their experiment. The machine learning algorithm gets credit for many correct decisions not to place alerts into existing scenarios, even when it has placed the alert in question into an incorrect scenario.) In fact, because of these methodological errors, their testing results can be discounted, regardless of the novel ideas they present in their paper.

2.2 Artificial Neural Networks

(Please note that this section appears unchanged from the section by the same name in our report for Autocorrel I (Japkowicz and Smith [1]). We include it here because it contains sufficient background for the knowledge required in neural networks for this report, even though it's unchanged.)

The area of artificial neural networks (ANNs) have been a subject of intense research lately. As a result, the field of ANNs is now very broad. In this section, we present a brief overview of ANNs.

ANN models attempt to emulate the human brain through the dense interconnection of simple computational elements called neurons [30]. Each neuron is linked to some of its neighbours through synaptic connections of varying strengths. Learning is accomplished by continuously adjusting these connection strengths (weights) until the overall network outputs the desired results. These weight adjustments are based on mathematical algorithms used in solving nonlinear optimization functions.

2.2.1 The Neuron

Similar to the biological nervous system, the basic computational element of an ANN is called the neuron or processing node. The neuron model is based on highly simplified considerations of the biological neuron. A simple node is shown in Figure 1, where N inputs are summed at the node. Each input u_i is connected to the processing node through the synaptic connections, which are represented by connection strengths called weights w_i. A bias term θ is also used at each node. The sum is fed through a transfer function f, called the activation function, to generate the output o. The signal flow is considered unidirectional as indicated by the arrows.

Although ANNs are constructed using this fundamental building block, there are significant differences in the architectures and driving fundamentals behind each ANN model.

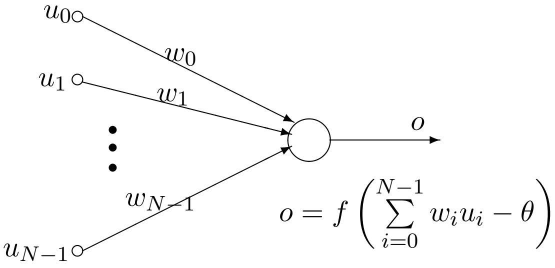

Figure 1: *The basic neuron*

2.2.2 The Activation Function

The activation function f plays a pivotal role in the functioning of the neuron. It determines the node output. As in Figure 1, the neuron output signal is given by:

$$o = f\left(\mathbf{w}^T\mathbf{u}\right) \tag{1}$$

where $\mathbf{w}$ is the weight vector defined as

$$\mathbf{w} \equiv \left[\begin{array}{cccc} w_1 & w_2 & \cdots & w_N \end{array}\right]^T$$

and the input vector $\mathbf{u}$ is defined as

$$\mathbf{u} \equiv \left[\begin{array}{cccc} u_1 & u_2 & \cdots & u_N \end{array}\right]^T$$

There are many different types of activation functions f to choose from, depending on the application [30, 31, 32]. Some of the commonly used activation functions are shown in Figure 2. These activation functions are the *hard-limiter*, the *threshold logic*, and the *sigmoid*. Since real applications are usually modeled as continuous functions, the most commonly used continuous activation function is the sigmoid.

Activation functions may be either unipolar, for positive output, or bipolar for output that may be positive or negative. For example, the bipolar sigmoidal activation function is defined as:

$$f(x) \equiv \frac{2}{1+\exp^{-\lambda x}} - 1 \tag{2}$$

and the unipolar sigmoidal activation function is defined as

$$f(x) \equiv \frac{1}{1+\exp^{-\lambda x}} \tag{3}$$

where λ is a constant.

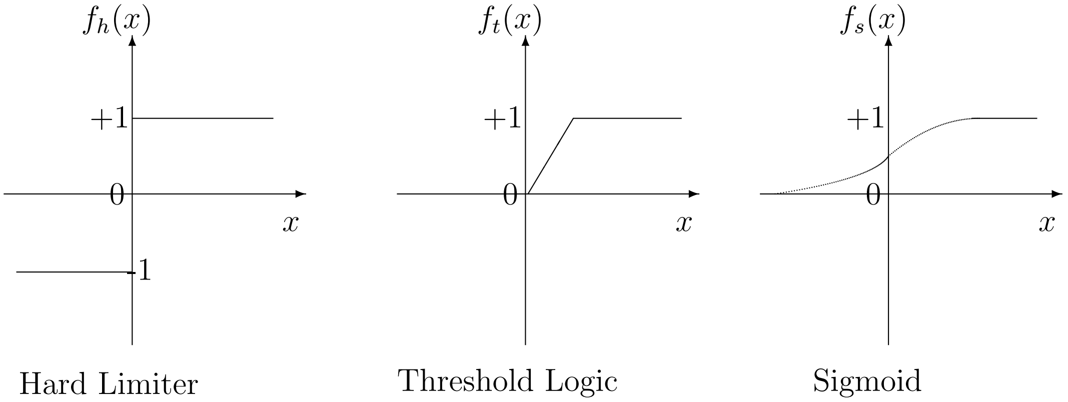

Figure 2: *Activation functions*

A special case of an ANN is a single node based on the neuron model shown in Figure 1 and is called a *perceptron* after the work of Rosenblatt [30]. A perceptron consists of one or more neurons. If a continuous activation function is used, the neuron model is known as a *continuous perceptron*. A continuous perceptron is capable of classifying *linearly separable* classes of data of the form $a\mathbf{x} + b$. Multiple nodes in this format form a single layer multi-node ANN capable of classifying linearly separable data patterns.

2.2.3 Multi-Layer ANNs

To emulate massively interconnected biological systems, ANNs have to be similarly interconnected. ANNs are the simple clustering of primitive artificial neurons. This clustering occurs by creating layers of neurons which are connected to one another. Figure 3 shows a multi-layer perceptron. An input layer interfaces with the outside world to receive inputs and an output layer provides the outside world with the network's outputs. The rest of the neurons are hidden from view, and are called *hidden layers*.

The objective of using a multi-layer perceptron is to be able to classify patterns that linear classifiers (single layer ANNs) are incapable of classifying. The most important attribute of multi-layer ANNs is that they can learn to classify a problem of any complexity. The biggest challenge is usually in deciding the number of hidden layers in an ANN.

Zurada [32] gives an extensive discussion on the design of the number and size of hidden layers in a given architecture; nevertheless, trial and error methods have been

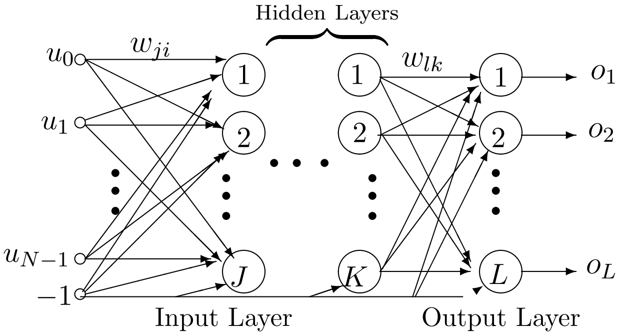

Figure 3: *Multi-layer perceptron*

widely used. If the number of hidden layers is too large, the ANN architecture will have problems generalizing; it will simply memorize the training set, making it useless for use with new datasets.

Inter-layer connections within an ANN architecture can take the following forms [32]:

- In a *fully-connected* ANN, each neuron on one layer is connected to every neuron on the next layer.
- In a *partially-connected* ANN, a neuron on one layer does not have to be connected to all neurons on the next layer.

If signal flow direction is taken into consideration, these two architectures can be further refined:

- In a *feedforward* ANN, the neurons on one layer send their output to the neurons in the next layer, but they do not receive any input back from the neurons in the next layer.
- In a *bi-directional* ANN, the neurons on one layer may send their output to the next layer or the preceding layer, and the subsequent layers may also do the same.
- In a *hierarchical* ANN connection, the neurons of a lower layer may only communicate with neurons on the next level of layers.
- In a *resonance-connected* ANN, the layers have bi-directional connections, and they can continue sending messages across the connections a number of times until previously defined conditions are achieved.

In more sophisticated ANN structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. These take the following two forms:

- In fully- or partially-connected *recurrent* networks, neurons within a layer communicate their outputs to neurons within the layer. This is done a number of times before they are allowed to send their outputs on to another layer.

- In *on-center/off-surround* ANNs, a neuron within a layer has excitatory connections to itself and its neighbors, and has inhibitory connections to other neurons. The neurons exchange their output signals a number of times until a winner is found. The winner is allowed to update its and its members' weights.

The overall architecture of an ANN depends on the mappings required, the type of input patterns, and the learning rules to be used.

2.2.4 ANN Training

Similar to the brain, ANNs learn from experience by changing the ANN's connection weights until a solution is found. The learning ability of an ANN is determined by its architecture and by the algorithm chosen for training. The training methods [30] fall into broad categories:

- In *unsupervised training*, hidden neurons find an optimum operating point by themselves, without external influence.

- *Supervised training* requires that the network be given sample input and output patterns to learn. It is guided through the learning process until a satisfactory optimum operating point or a predefined threshold is reached. The most common training termination criteria is by setting a training threshold.

Backpropagation training is a form of supervised learning that has proven highly successful in training multi-layered ANNs. Information about errors is filtered back through the system and is used to adjust the connections between the layers, thus improving performance.

ANNs can be trained *on-line* or *off-line*. In off-line training algorithms, its weights do not change after the successful completion of the initial training. This is the most common training approach; especially in supervised training. In on-line or real time learning, weights continuously change when the system is in operation [32].

2.2.5 Training Rules

There is a wide variety of learning rules that are used with ANNs. Error minimization algorithms are used to determine the convergence levels when updating weights. In general, all ANN learning involves the iterative updating of the connection weights until the desired convergence is achieved. Most training algorithms start by initializing the weights to 0 or very small random numbers. This weight update is given by:

$$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \Delta\mathbf{w}^{k} \tag{4}$$

Equation 4 is the ANN *general learning rule* [32]. The numerous learning rules, which are variations of this rule, only differ by the mathematical algorithms used to update the connection weights, or more specifically to calculate the value of $\Delta\mathbf{w}^k$ at each iteration k. Some of the common training rules are as follows:

- In the *Hebbian* rule [32, 31], the connection weight update $\Delta\mathbf{w}^k$ is proportional to the neuron's output. This was the first ANN learning rule [30, 33].

- The *perceptron* rule [30] updates the weights based on the difference between the desired output d and the actual neuron's response o.

- The *delta* learning rule [30, 33] is based on the minimization of the mean square error (MSE) as represented by the error function E, as shown in Equation 5.

  $$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla E\left(\mathbf{w}^{k}\right) \tag{5}$$

  where η is a learning constant, and ∇E is the gradient of the error function E, defined by:

  $$E_k = \frac{1}{2}\left(d^k - o^k\right)^2 \tag{6}$$

  The objective is to iterate Equation 5 until the error E approaches zero (or a preset threshold value).

  For an ANN with P training patterns, and K outputs, the *root-mean square error* (also known as the MSE [32]) is defined as:

  $$E_{rms} = \frac{1}{PK}\sqrt{\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{pk} - o_{pk})^2} \tag{7}$$

- The *Widrow-Hoff* [31, 32] learning rule (sometimes called the *Least Mean Square* learning rule) is considered a special case of the delta learning rule in that the neuron output o is independent of the activation function f.

- The most widely used supervised training approach which is derived from the Widrow-Hoff algorithm is the *error backpropagation training algorithm.* As the name implies, the error $\Delta\mathbf{w}^k$ is propagated back into the previous layers. This is done one layer at a time, until the first layer is reached.

Consider an ANN with one hidden layer, K outputs, J hidden nodes, I inputs, and P training patterns. The output layer weights are adjusted as follows:

$$w_{kj} = w_{kj} + \eta \delta_{ok} y_j, \quad \text{for k} = 1, \cdots, \text{K}, \ \text{j} = 1, \cdots, \text{J} \tag{8}$$

where η is a learning constant and the output error $\boldsymbol{\delta_{\mathrm{ok}}}$ is given by

$$\delta_{ok} = \frac{1}{2}(d_k - o_k)(1 - o_k^2), \quad \text{for k} = 1, 2, \cdots, \text{K} \tag{9}$$

The weight update for the hidden layer is as follows:

$$w_{ji} = w_{ji} + \eta \delta_{yj} u_i, \quad \text{for k} = 1, \cdots, \text{K}, \ \text{i} = 1, \cdots, \text{I} \tag{10}$$

where the output error $\boldsymbol{\delta_{\mathrm{yj}}}$ is given by

$$\delta_{yj} = \frac{1}{2}(1 - y_j^2) \sum_{k=1}^{K} \delta_{ok} w_{kj}, \quad \text{for j} = 1, 2, \cdots, \text{J} \tag{11}$$

The process is iteratively repeated until a preset threshold of the MSE (Equation 7) is achieved.

A good, longer treatment of neural networks in all their incarnations, see the book Neural Network Design by Hagan et al. [34].

2.2.6 The Autoassociator

Based on the work by Japkowicz [35], we extend the uses of the autoassociator to a new application, a clustering algorithm. We use the autoassociator to cluster alerts outputted from an intrusion detection system. We give the details of how this new clustering algorithm, and the motivation for it, in Section 3.2.2.

The autoassociator is a piece of feedforward, fully-connected, multi-layer neural network architecture whose design is specialized for recognizing one class of training data rather than discriminating between training input of different classes, as other neural network architectures are designed to do. The autoassociator is known as an unsupervised machine learning algorithm because the classifier is trained on one class of data, so that it can recognize that class of data. Although the autoassociator is trained on only one class of data, it's tested on both classes of data in a binary learning problem (ie- two class problem).

In Figure 4, d is the number of attributes of the data items in the dataset. $x_1, \ldots, x_d$ represent the input values of the attributes for a data item, $f_1(x), \ldots, f_d(x)$, represent the reconstructed output values of the autoassociator, and $h_1, \ldots, h_k$ represent one

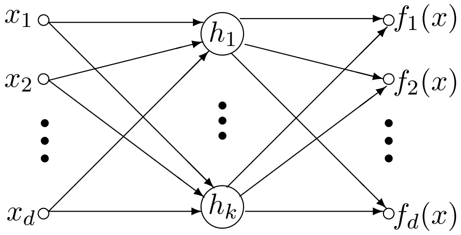

Figure 4: *The autoassociator*

layer of k hidden units in Figure 4. The potential of the autoassociator to internally represent data trends often lies in the number of hidden units and design of the hidden layer, so these parameters should be carefully tailored to the problem.

The most clear difference between the design of the autoassociator in Figure 4 and the multi-layer perceptron is that the autoassociator has d output neurons, where the multi-layer perceptron has only 1 output neuron. This difference is crucial.

The reconstruction error for a data item is computed using the $f_1(x)$, $f_2(x)$, ..., $f_d(x)$ autoassociator output values from Figure 4. The formula for computing the reconstruction error is:

$$E_x = \sum_{i=1}^{d} [x_i - f_i(x)]^2 \tag{12}$$

where x_i is the value of the i^{th} attribute of data item x and $f_i(x)$ is the value of the i^{th} output of the neural network with input x. After training, the autoassociator is able to recognize the class represented in the training data because data items from the training class will have low reconstruction error values when compared to that of data items from unseen classes. The theory holds that data items from classes without representative training examples will have higher reconstruction errors.

A good treatment of using the autoassociator for this type of classification is [35]. This paper explains more of the theory behind the operation of the autoassociator, as well as applying this style of neural network in a different way from what we present here.

2.3 Clustering Algorithms

We use the EM algorithm (Section 2.3.1) and self-organizing maps (Section 2.3.2) as clustering algorithms. We also use the autoassociator (Section 2.2.6) as a clustering algorithm, but this use is new, so we do not present it here in the background section. We should note that self-organizing maps are neural networks and could have been discussed in Section 2.2, but we employ them as a clustering algorithm here, so we describe them in this section on clustering algorithms.

2.3.1 The EM Algorithm

The EM algorithm is one of the most well-known clustering algorithms. Initially presented in Dempster *et al* [36], it consists of two repeated steps, expectation and maximization. According to Witten and Frank [37], the EM algorithm uses a statistical model called *finite mixtures* to achieve the goal of producing the most likely set of clusters given the number of clusters, k, and a set of data. The model consists of a set of k probability distributions, one to represent the data of each cluster. There are parameters that define each of the k distributions. The EM algorithm begins by making initial guesses for these parameters based on the input data, then determines the probability that a particular data instance belongs to a particular cluster for all data using these parameter guesses. The distribution parameters are revised again and this process is repeated until the resulting clusters have some level of overall cluster “goodness” or until a maximum number of algorithm iterations is reached. The expectation step of the algorithm estimates the clusters of each data instance given the parameters of the finite mixture; the maximization step of the algorithm tries to maximize the likelihood of the distributions that make up the finite mixture, given the data.

2.3.2 Self-Organizing Maps

Self-organizing maps (SOMs) were created by Kohonen and they are explained in detail in his book [38]. SOMs are an unsupervised method of mapping a high-dimensional input to a two-dimensional output such that similar inputs are mapped to close locations in the two-dimensional map output. (Actually, other dimensionalities are available for output as well, but two-dimensional output is the most common.) This mapping is primarily used for the visualization of high-dimensional data, but it can also be used for clustering. SOMs are a type of neural network — a single-layer neural network to be precise — and, accordingly, have similar training parameters.

Self-organizing maps are not natively intended for clustering, though they can easily be adapted to this application. There exist papers that analyze clustering with SOMs. Tang *et al* [39] succinctly discuss clustering the output of the SOM. The requirements of their experiments are identical to ours: (1) the need for hard-clustering, (2) automatic determination of the number of clusters k, and (3) robustness of the clusters formed. Given these requirements they chose to employ the *potential function*. This algorithm is described in detail in a paper by Chiu [40], so we do not describe it here. As well, we weren't able to find an existing implementation of this algorithm, so we decided to go with a different method of clustering the SOM, despite the obvious suitability of this method. In a paper by Vesanto and Alhoniemi [41] other methods of clustering the SOM are discussed.

We decided to use the MATLAB (Demuth and Beale [31]) implementation of SOMs, which allows a form of clustering. In this implementation the SOM neural network is trained, then, as with other neural networks, data can be simulated on the network to discover the output of the network for that input. For SOMs the network output of a data instance input is a point in the SOM. Any two vectors which share the same point in the SOM should have similar representation in the higher dimensional space, by the design of the SOM algorithm. For clustering the SOM output in MATLAB we can simply create one cluster for every point in the SOM output that has at least one data instance associated with it. In practice this produces acceptable clusters.

A good document on how to use SOMs is the SOM_PAK program tutorial by Kohonen *et al* [42]. This document explains in depth the possible parameters for SOMs, which we'll only discuss briefly here. We discuss the parameters available for tuning with the Matlab Neural Network Toolbox [31] in Section 2.3.2.1.

2.3.2.1 Self-Organizing Map Parameters

SOMs have a number of different parameters to tune. The *lattice type*, or *topology*, of the network is one such parameter. There are two options for this lattice type, rectangular (also known as grid) and hexagonal. According to Kohonen *et al* [42] the lattice type hexagonal is best suited for visual display. As well, this type is the default for both SOM_PAK and the MATLAB Neural Network Toolbox (Demuth and Beale [31]). Because of this we choose the hexagonal lattice type when experimenting with our SOMs. Another important parameter is the distance function, used to determine the distance between points in the map. We use the default distance function in the MATLAB Neural Network Toolbox, `LINKDIST`.

There are two phases to training an SOM in MATLAB, an ordering phase and a tuning phase. Kohonen *et al* [42] also recommends this two step process. During the ordering phase the training data are given their approximate correct position in the map, and during the tuning phase the positions of the data are fine-tuned. With SOM_PAK you're able to change all the parameters at both these steps, but in MATLAB you can only change the learning rate, `OLR`, and maximum number of steps, `OSTEPS`, at the ordering phase. (`OSTEPS`, the maximum number of training steps, is analogous to the *epochs* parameter for training the autoassociator.) At the tuning phase you only control the learning rate, TLR, and the neighbourhood distance function, TND. The learning rate parameter in SOMs is similar to that parameter in other neural networks. It controls how quickly the neural network converges to a desired state and it decays while doing so. The neighbourhood distance function defines the radius of the neighbourhood during training. A larger radius allows the training vectors to move greater distances in the map at each training step.

3 Our Model

The purpose of our research is to develop a system of alert correlation that can find correlation between different steps of an attack. The system we've created is based on unsupervised machine learning algorithms, which means, practically, that less setup and maintenance are required in an operational setting, when compared with a supervised system. Our system of finding correlation is implemented in two steps. The first step of the system clusters alerts such that each cluster represents a different stage of an attack. The second step clusters together different stages of a single attack.

The two steps in our system interact in an intuitive way. The first step takes a set of alerts, constructs the first step features for the alerts, and outputs clusters of those alerts such that each cluster represents a distinct step of an attack. The second step in our system takes these clusters from the first step, constructs the second step features from these clusters of alerts, then outputs the correlations found by our system. We discuss the details of the operation of the two steps, including the different feature constructions, in Section 3.2 for the first step and Section 3.3 for the second step.

We have a main justification for using unsupervised machine learning algorithms in our system rather than supervised machine learning algorithms. The primary reason for using unsupervised algorithms in our research is that supervised machine learning algorithms impose a strong constraints on how the system functions which makes the deployment of supervised correlation system impractical. For instance in the supervised machine learning-based research by Dain and Cunningham [27], they attain excellent performance with their system, but their reported performance is predicated on the presence of a significant amount of hand-clustered data. In other words their system works very well but requires that a large percentage of the alerts of an attack must be organized by hand beforehand.

Another example of a supervised machine learning-based alert correlation system is the system presented in Julisch *et al* [26]. Since their system is based on supervised machine learning algorithms — in particular, a type of algorithm known as a *conceptual clustering algorithm* — an adminstrator is required to tune the correlation system every few months. As well, when the system is first deployed it is necessary to encode properties of the network under inspection to help the clustering algorithm. This means that the setup and maintenance required for their system is significant.

3.1 The Data

We use the Snort IDS [8] to read the data, which comes from the incidents.org [43] website. The alerts generated by Snort are output as text to Perl [44] scripts that can understand the Snort alerts and that format the data for use with any machine learning algorithm.

3.1.1 Gathering and Selecting

We chose to use the incidents.org dataset for our evaluation and testing because it produced interesting results. There were many interesting correlations to be found in the incidents.org dataset, which made the data easier to work with and easier to demonstrate our system with. The dataset is a capture of real data from the internet — that is, the dataset wasn't generated automatically, but captured "in the wild" — which we believe accounts for the fact that it's better to work with. The 1999 DARPA dataset [17] was automatically generated to model real traffic (because of fear of privacy issues), and the DEFCON dataset [28] was captured from a hacker competition where there were very few legitimate uses on the network and where the attackers had no reason to hide their attacks.

In the set of alerts generated by Snort on the incidents.org data, we reserved the first 1000 alerts for evaluation and testing, and we reserved the next 10,000 alerts for training. We recognize that a proper experiment would be to train on the first 10,000 alerts, then test on the remaining alerts, since alerts appear earlier in the Snort output if they were encountered earlier. Theoretically, testing on alerts from before the training period could invalid the test since it's not a realistic scenario for a real-world deployment — that is, in the real world the classifier will always be trained on previously seen data then deployed for future data — but the practical concern that the sample of alerts at the beginning were more diverse and interesting took precidence over this theoretical concern.

The data we've selected offers the TCP/IP traffic as seen by only one sensor. If we'd had access to traffic as seen by multiple sensors then we could have tested our system at this type of correlation as well, since we believe that similar alerts represented by different sensors will be encoded in the same way, but this type of data isn't readily available, so were weren't able to test our system in this way.

3.2 First Correlation Step

In the first step of our system we use a set of features that remains almost unchanged from the set of features we reported on for our first DRDC report [1]. This set of features was created to fully represent an IP packet that has been flagged as an alert by an IDS, in particular the Snort IDS in our experiments. The feature set allows similar-looking IP packets to be clustered together. Together, this feature set combined with our use of the autoassociator neural network [35], they work to cluster

IP packets received in a given window into individual steps of greater attacks. We give the details of how alerts are encoded as a set of features in Section 3.2.1.

For the purpose of our research we define a *step* (or *stage*) of an attack to be one action in the greater attacker's plan. For instance, one step by an attacker would be running the security tool `nmap` once against a network to discover what services are available. This step wouldn't include the use of tools to attack particular services, or other reconnaisance tools used by the same attacker.

We use the autoassociator neural network for clustering in this first step of correlation. We use this neural network architecture because it is very flexible and because it performed well in the experiments of our previous report [1]. Another reason we use this algorithm is that the neural network can be trained with unlabelled training data, which makes the results between successive runs of the algorithm on different datasets comparable, due to the stability in the reconstruction error values. (We discuss this topic more in Section 3.2.2.2.) The final reason we choose the autoassociator at this step is because the autoassociator, used with the cluster barrier algorithm in Section 3.2.3, doesn't require the number of output clusters to be known ahead of time. The other clustering algorithms we evaluated for this first step, such as the *EM algorithm* [36] or *self-organizing maps (SOMs)* [38], either explicitly require the number of output clusters to be given or require architecture choices that dictate the number of clusters formed.

To train the autoassociator we select 10,000 unlabelled alerts and train the autoassociator (implemented in Matlab for our tests) for 500 epochs using 64 hidden units in the construction of the network. (See Section 3.2.2 for details on the selection of these parameters.) We use the back propagation algorithm and Matlab's implementation of the gradient decent method for training the autoassociator.

After the autoassociator is trained we use simulate our set of evaluation alerts on the network to obtain the reconstruction error values for the data. We cluster the evaluation data using these reconstruction error values and the cluster barrier algorithm from our previous DRDC report [1]. (The details of this algorithm are in Section 3.2.3.) The use of the reconstruction error formula creates the problem of a 40-to-1 mapping that causes some errors in clustering. We discussed this problem in our previous research, and we discuss the effects of this mapping further in Section 3.2.2.1. We propose an experiment to learn more about the problem in Section 4.2.1.

Our use of unlabelled data to train our unsupervised machine learning algorithm is supported by literature. Specifically in Duda, Hart and Stork [45] they write that it's possible to "train with large amounts of (less expensive) unlabelled data, and only then use supervision to label the groups found." This use of unlabelled data corresponds to our own use and thus gives us justification for our architectural decision.

3.2.1 Feature Construction

We have 40 features to characterize each alert generated by Snort [8], the IDS we use in our testing. The features we use are taken directly from the Snort alerts. We do not to use any of information specific to Snort, so we argue that our selection of features can represent the alerts generated by any IDS rather than only those from Snort. (We do not use the *Intrusion Detection Message Exchange Format (IDMEF)* standard by the *Intrusion Detection Working Group (IDWG)* [22] because the set of mandatory features in this standard isn't sufficient for correlation (Hätälä *et al* [21]), so vendor-specific alert processing is always necessary.) For instance, we do not use the Snort attack priority level or Snort's labelling of the attack because this information cannot be found in the raw IP packet. We wanted to make our prototype system as general as possible, so we showed how using only protocol features can still allow for very good clustering with the autoassociator.

We did not include the IP source address or IP destination address protocol fields in the list of classification features because we noticed that these features tended to impede correct clustering. We also reason that if we wish to cluster a number of packets together that make up a *distributed denial of service (DDOS)* or similar attack, the IP address fields could only impede classification since they are easily forged. We note, however, that we do use IP source and destination addresses in our second step of correlation.

We include the TCP source and destination port features because we find they increase performance for clustering types of attacks on particular TCP services. We do not use the TCP sequence and TCP acknowledgement numbers at this step because we find that they hinder performance.

We tested, evaluated and considered the other features individually, but we decided to keep them all because they all helped classification. In the end, we had 40 features that we passed to the autoassociator for classification. These features are listed in Table 1.

3.2.1.1 Formatting

All of the 40 features we selected were quite easy to convert to numerical data. For instance, if the TCP Ack flag was set in the TCP header (if there was a TCP header), the value of the tcpAckFlag feature was set to 1, and if there was no TCP header or the flag was clear, the tcpAckFlag feature set to 0.

Table 1: *First Step Features*

Feature	Feature	Feature	Feature
portSrc	portDest	ipIsIcmpProtocol	ipIsIgmpProtocol
ipIsTcpProtocol	ipIsUdpProtocol	ipLen	ipDgmLen
ipId	ipTos	ipTtl	ipOptLsrr
ipPacketDefrag	ipReserveBit	ipMiniFrag	ipFragOffset
ipFragSize	icmpCode	icmpId	icmpSeq
icmpType	tcpFlag1	tcpFlag2	tcpFlagUrg
tcpFlagAck	tcpFlagPsh	tcpFlagRst	tcpFlagSyn
tcpFlagFin	tcpLen	tcpWinNum	tcpUrgPtr
tcpOptMss	tcpOptNopCount	tcpOptSackOk	tcpOptTs1
tcpOptTs2	tcpOptWs	tcpHeaderTrunc	udpLen

Some features of note are the `ipIsIcmpProtocol`, `ipIsIgmpProtocol`, `ipIsTcpProtocol`, and `ipIsUdpProtocol` binary features. If a data item has the value one for the `ipIsTcpProtocol` feature, this indicates that the Snort alert signifies an IP packet containing a TCP payload. Since the web server in this alert listens on a TCP port, this feature has value 1 for this data item. A Snort alert for a non-TCP IP packet will have `ipIsTcpProtocol` set to zero. Only one of the `ipIs*Protocol` features will be set for a given data item. We created these features because we imagined that it would be possible for two packets of the same protocol to be essentially identical in all respects except for the IP protocol field.

3.2.1.2 Formatting Example

To illustrate precisely how the transformation is made from a textual alert to a data item, and what information is encoded, we present an example. (Although the data items we output can be scaled, we present a non-scaled data item here for readability.) The following Snort-generated alert was processed with our scripts to generate a row of numbers, where each number corresponds to a predetermined feature. The alert below was generated by Snort after a possible attacker tried to access the UNIX `telnet` service. It's possible that this alert could have been generated by a valid user, but it's also possible that this alert could have been generated by an attacker trying to penetrate the system. If this second hypothesis were true, it is likely that you would find many `TELNET login incorrect` errors close to this one targetting the same machine.

```
[**] [1:718:7] TELNET login incorrect [**]
04/05-09:38:26.936288 172.16.114.50:23 -> 172.16.114.168:10332
TCP TTL:64 TOS:0x10 ID:2014 IpLen:20 DgmLen:66 DF
**AP*** Seq:0x801D2FDC Ack:0x6EDB5C7F Win:0x7C00 TcpLen:20
```

The Perl scripts we've created encode this alert information as predetermined features. We've created Table 2 for the encoded features of the data item. As an example of how to read Figure 2, in the data below, the number 23 is the TCP source port (`portSrc`) given in the alert.

Table 2: *An Encoded Alert*

Feature	Value	Feature	Value
portSrc	23	portDest	10332
ipIsIcmpProtocol	0	ipIsIgmpProtocol	0
ipIsTcpProtocol	1	ipIsUdpProtocol	0
ipLen	20	ipDgmLen	66
ipId	2014	ipTos	16
ipTtl	64	ipOptLsrr	0
ipPacketDefrag	1	ipReserveBit	0
ipMiniFrag	0	ipFragOffset	0
ipFragSize	0	icmpCode	0
icmpId	0	icmpSeq	0
icmpType	0	tcpFlag1	0
tcpFlag2	0	tcpFlagUrg	0
tcpFlagAck	1	tcpFlagPsh	1
tcpFlagRst	0	tcpFlagSyn	0
tcpFlagFin	0	tcpLen	20
tcpWinNum	31744	tcpUrgPtr	0
tcpOptMss	0	tcpOptNopCount	0
tcpOptSackOk	0	tcpOptTs1	0
tcpOptTs2	0	tcpOptWs	0
tcpHeaderTrunc	0	udpLen	0

As you can see from the Snort alert, the TCP header in this packet has only the Ack and Push TCP flags set. (The string `***AP***` indicates this.) This information is encoded in the `tcpFlag*` group of features. `tcpFlagAck` and `tcpFlagPsh` are set to 1 and the rest of the flags are set to 0.

The `portSrc` and `portDest` features represent the TCP source and destination ports for a Snort alert. The values of the `portSrc` and `portDest` features are taken directly from the alert, as you can see above. The value for `portSrc` and `portDest` are 23 and 10332 respectively. The TCP source and destination port features are important in finding good correlations because they are often the only obvious indicator to which protocol the alert relates.

3.2.1.3 Scaling Algorithms

We scale the data using the common method (see Duda, Hart and Stork [45]) of determining a linear map from the training data then applying it to both the training data and the testing and evaluation data.

Specifically, for every feature in the dataset we determined the range of values for that set. We called the highest value for feature f *high* and the lowest value *low*. From these values we computed the linearly scaled value for f of a data item d_i as $(d_i - low)/(high - low)$. For the case when $high = low$, we computed f for d_i as $d_i - low + 1$. You can see that with this computation, as long as all the input values are in the interval $[low, high]$ then the linearly mapped values will be in the interval $[-1, 1]$. (Note that in our previous report [1] we used the same scaling algorithm, but scaled to the interval $[0, 1]$ instead of $[-1, 1]$.)

The necessity for scaling is explained in a paper on experimenting with support vector machines (SVMs) (Lin *et al* [46]). Lin gives an overview of why scaling is important — basically, so that one feature doesn't numerically overwhelm other ones — and why it's okay to have testing or evaluation data outside of the chosen scaling interval. We determine the $[low, high]$ range for every feature using the training data then use these values to scale the testing data. The precise range for the testing cannot be known ahead of time, by the definition of the testing data, and to allow for a realistic representation of the testing data, the model must acknowledge that training data rarely perfectly predicts testing data.

For our previously reported system [1] we meant to take these values for *high* and *low* from the training set and apply the scaling with these values to all of the evaluation set. But because of a bug in our software, we were effectively determining new values for *high* and *low* for the evaluation set, which happened to have *high* and *low* values close to those of the training set. Because of this bug we want to report how our system would perform without the bug.

We also want to try scaling our system to a set of predefined ranges, which was originally suggested by our DRDC liason, Dr. Dondo. We believe this type of scaling is intuitive, so we present results for that change as well. This type of change is a good idea because of the domain we're experimenting in. For IP addresses, as well as all of the other features we've selected for the first step of our Autocorrel system, we know that the addresses are 32 bits by definition in the IP header, so the values of the addresses are always in the range of encoded values $[0, 2^{32} - 1]$. We can assign $low = 0$ and $high = 2^{32} - 1$ and remove the step for determination of *high* and *low* altogether when running the scaling algorithm, leaving the system with more predictable sets of values produced. For each of the features we were able to determine a correct range of values from domain literature such as Stevens [2]. We do not need to reproduce the ranges here because the format of the IP, TCP, etc. headers are well known.

We also wanted to compare linearly scaling to the interval $[0,1]$ instead of $[-1,1]$ because this is the range we used in the previous report for DRDC. To do this we took our previously scaled values for feature f at data item d_i, say $scaled([-1,1], d_i^{(f)})$, and performed the following transformation to get $scaled([0,1], d_i^{(f)})$: $scaled([0,1], d_i^{(f)}) := scaled([-1,1], d_i^{(f)})/2 + 0.5$.

Lastly, we wanted to try a Gaussian method of scaling, as well, that relies on accurate (and Gaussian) distributions in the training data. To compute the Gaussian scaling for $d_i^{(f)}$, we first have to determine $\mu^{(f)}$ and $\sqrt{\omega}^{(f)}$. To do this, we used the mean and standard deviation calculations, respectively, of Matlab. Once these values are found, we could compute the new value for $d_i^{(f)}$ as $(d_i^{(f)} - \mu^{(f)})/\sqrt{\omega}^{(f)}$, where the formula includes the unscaled values for $d_i^{(f)}$. This gives the training data a new mean of 0 and a new standard deviation of 1. The values of $\mu^{(f)}$ and $\sqrt{\omega}^{(f)}$ from the training set are used to scale the evaluation and testing sets.

For completeness we also discuss our results when trying out our system using only unscaled data.

We consider these scaling methods and try to determine whether our selected method is superior. We present these experiments in Section 4.2.3.

3.2.2 Training the Autoassociator

Machine learning is used in this project to cluster numerically similar network events. We will show how the autoassociator, an elegant neural network design, can be used for exactly this task. There is a post-processing step required to interpret the neural network output, which we'll discuss in the next section, but the autoassociator is used in the same way as is described in Section 2.2.6. The difference between the way we use the autoassociator and how it is used in "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks" (Japkowicz [35]) is in what type of data we input to the neural network and how we process the output from the network.

In our task we use the autoassociator for clustering rather than for binary classification. Where Japkowicz [35] found a cutoff in the reconstruction error values to separate two classes of data, we group the reconstruction error values to form clusters. The difference in reconstruction error of two given data is positively correlated to the general numerical difference of those data. In our task, if the numerical representation of two network events is similar, the autoassociator will produce similar reconstruction errors. The converse is often true, but not always true: numerically dissimilar network events often produce dissimilar reconstruction errors, but not always. The reason this is the case is because the reconstruction error formula maps the 40 autoassociator outputs to the one-dimensional range of error reconstruction values. Because of this 40-to-1 mapping, there is a lot of information lost, so numerically different errors can be mapped to the same reconstruction error. We explore this problem further in Section 3.2.2.1.

As an example of the correlation of error reconstruction values, the last two Snort [8] alerts listed below are very similar and the first is dissimilar from the last two. Since the last two alerts are numerically similar, they have very similar reconstruction errors (which are the same in this particular case: 3.1920 for both alerts, after 500 training epochs), but the last two have substantially different reconstruction errors from the first Snort alert below (which had reconstruction error 3.3388 in the same experiment).

```
[**] [1:716:10] TELNET access [**]
04/05-09:35:53.754855 172.16.112.50:23 -> 172.16.114.148:7298
TCP TTL:255 TOS:0x0 ID:28873 IpLen:20 DgmLen:55 DF
**AP*** Seq:0x367323BE Ack:0xF384C1BE Win:0x2238 TcpLen:20

[**] [1:1244:13] WEB-IIS ISAPI .idq attempt [**]
04/05-09:37:05.489493 172.16.117.103:8307 -> 208.160.10.123:80
TCP TTL:64 TOS:0x0 ID:4996 IpLen:20 DgmLen:549 DF
**AP*** Seq:0x336579CE Ack:0x7B33D444 Win:0x7D78 TcpLen:20

[**] [1:1245:10] WEB-IIS ISAPI .idq access [**]
04/05-09:37:05.489493 172.16.117.103:8307 -> 208.160.10.123:80
TCP TTL:64 TOS:0x0 ID:4996 IpLen:20 DgmLen:549 DF
**AP*** Seq:0x336579CE Ack:0x7B33D444 Win:0x7D78 TcpLen:20
```

From the list of reconstruction errors of a set of Snort alerts, we can use a simple algorithm to form discrete clusters of Snort events whose numerical representations are similar. This step is necessary because the autoassociator doesn't explicitly form clusters; there is a post-processing step required to do this. We discuss how this algorithm works in Section 3.2.3.

3.2.2.1 Feature Mapping Problem

The reconstruction error formula (Equation 13) computes a single real value from the 40 autoassociator outputs, causing a 40-to-1 mapping. The use of a 40-to-1 mapping formula was necessary because it allows us to form discrete clusters from the otherwise non-interpretable 40-dimensional output of the autoassociator. We chose to use the reconstruction error formula as the formula for performing the 40-

to-1 mapping because it was used in Japkowicz [35], where the formula was used to generate ROC graphs (Fawcett [20]) in a binary classification problem.

$$E_x = \sum_{i=1}^{n} [x_i - f_i(x)]^2 \tag{13}$$

In Equation 13 x_i is the value of the i^{th} attribute of data item x and $f_i(x)$ is the value of the i^{th} output of the neural network simulated with input x.

The problem lies in the fact that different 40 autoassociator outputs can be mapped to the same real value using the formula. Our cluster barrier algorithm (Section 3.2.3) then sees the different autoassociator outputs as the same and clusters the items together.

We explore the extent of this problem in Section 4.2.1 and discuss possible solutions.

3.2.2.2 Reconstruction Error Stability

We have already suggested that stability is an important trait of the autoassociator trained with 10,000 alerts, but first we must more rigourously define stability in the context of reconstruction error values changing between experiments. Stability for a particular alert or data item means that for a given training set that the reconstruction error value of that alert doesn't change much if it is part of a very different greater evaluation set. What this means practically is that the reconstruction error should be mostly independent of the dataset which it's a part of.

We show the stability of the reconstruction error values produced by our system in an experiment in Section 4.2.6.

3.2.3 Clustering Autoassociator Output

The goal of the clustering algorithm is to input the reconstruction error values produced by the autoassociator, then turn the list of alerts into clusters with well-defined boundaries rather than a list of real numbers with no obvious boundaries (but with intuitive clusters). To achieve this goal, we found that a heuristic approach worked well. We decided that any cohesive gaps in the list of sorted reconstruction error values would indicate a boundary between clusters, so we used that idea.

We form discrete clusters such that every cluster obeys this rule: When you sort the reconstruction errors of the data items in the cluster, the difference between the reconstruction errors of any pair of items adjacent in the sorted list can be at most

0.0025. We developed this heuristic from the data, and we found that it worked well for our purposes.

This heuristic is formally known as a *single-link clustering algorithm* (Jain *et al* [47]) and it is applied to one-dimensional data — namely the reconstruction errors of the data. The clustering algorithm is known as a single-link algorithm because if the distance between any two data in a set is less than the threshold then those two data will appear in the same cluster. To compare this against a related heuristic, a *complete-link clustering algorithm* (Jain *et al* [47]) requires that the distance between every pair of data in a cluster is less than the threshold. This heuristic produces a very different sort of cluster.

We experimented with the size of the barrier between clusters before settling on 0.0025. We show this experimentation in Section 4.2.3.

3.3 Second Correlation Step

By having a second step in our correlation system we hope to be able to cluster each of the clusters produced by the first step to form *super-clusters*, or clusters of clusters of alerts. We do this because we hope to link clusters from the previous step that are related, but that do not form one discrete cluster. Effectively, we hope to be able to link different steps of an attack together by representing each step such that it can be correctly clustered with other steps from the same attack.

The motivation for this second step comes from Valdes and Skinner in their paper "Probablistic Alert Correlation" [25]. In their approach they use statistical methods primarily based on the type of the alert and the source and destination IP addresses of the alert. They correlate alerts using three distinct steps. The first step threads the alerts of a given sensor so that each logical step of an attack at a given sensor isn't represented multiple times. The second step of their system combines the output of the multiple sensors to fuse the data into one stream. After this second step each step of an attack should be represented only once in their system. For the final step of their system, they try to correlate their already-formed groups of alerts based on which groups are part of the same greater attack. At this step they use source IP addresses to determine attackers and destination IP addresses to determine targets. They also use predefined matrices of values that represent how likely it is that two types of alerts are related, if seen in succession.

In our system we combine the first two steps of Valdes and Skinner's system into one step, namely the first step in our system. Our system is fairly successful at threading a set of alerts, and we believe — without proof, but with strong indication, since our methods are sensor independent — that our system could handle multi-sensor threading. (We haven't proven this latter claim because multi-sensor data isn't available to us.)

Adding a second level of clustering effectively mimics Valdes and Skinner's intuitive approach of finding attack step correlation after the threading and multi-sensor correlation have been done. We should note, though, that both of the other systems we model our approach on, the systems of Dain and Cunningham [27] and Julisch and Dacier [26], do not use a multi-step correlation approach. The difference is that their systems are both supervised data mining systems, whereas ours is unsupervised.

3.3.1 Feature Construction

For this second correlation step, we take the output clusters of the previous stage and represent them each as new data items, each encoded using a new set of features. Feature construction for the data ultimately determines what is learned in machine learning problems. So, we intentionally construct our features such that a cluster of alerts representing a single step of an attack will be similar to another cluster of alerts from the same attack. We construct our features such that the converse is also true, by necessity.

We take our motivation for feature construction at this level from the paper "Fusing a Heterogeneous Alert Stream into Scenarios" (Dain and Cunningham [27]). In this paper the authors describe a number of features to be used in their supervised binary learning system.

Before describing the features in detail, we present our methods on determining the similarity of IP addresses, since it differs slightly with Dain and Cunningham's method. A measure of closeness between two IP addresses is used in the Dain and Cunningham system that is applied to source and destination IP addresses of alerts. Their system closeness is defined as the maximal number of most-significant bits shared between two encoded IP addresses (called r), when an IP address is encoded as a 32-bit integer in the same way we have done for Autocorrel I. (Namely, IP address $w.x.y.z$ is turned into the 32-bit integer $256^3w + 256^2x + 256y + z$.) So, for instance, in their system if they saw one IP address 192.168.2.17 and another IP address 192.168.2.16, these would be deemed very similar ($r = 31$), whereas a source address 192.168.1.16 would be considered less similar to the previous two ($r = 22$). (See their paper [27] for further details.)

Because our system is a clustering system where the features of all data are compared at once for similarity, we cannot implement a measure to compare just two IP addresses. As such, we implement a feature that makes two IP addresses similar if and only if they would be similar in the Dain and Cunningham system. In our system, we represent the similarity of a group of IP addresses by representing them as the IP addresses of the group with the unshared least significant bits masked out. So, using the previous examples, the group of IP addresses (192.168.2.16, 192.168.2.17) would be represented with the single IP address 192.168.2.16, and the group of IP addresses (192.168.2.16, 192.168.2.17, 192.168.1.16) would be represented as 192.168.0.0.

We now discuss the features that Dain and Cunningham use, and the analogous features in our system, in the following list. (Each of the items of text in the following list is preceeded by the name of the feature in our system.)

1. `ipSrcAddrCommonPart`: Attackers often prepetrate an attack from a single host, or from a single IP subnet. So Dain and Cunningham include a feature to indicate similarity between the source IP addresses of two alerts. In our system, we have a feature to indicate the shared part of the most-significant bits of a group of IP addresses from a cluster.

2. `ipDestAddrCommonPart`: Dain and Cunningham asserted that attackers often target a single host or subnet in their attacks. So they include a feature indicating the similarity of two destination IP addresses. In our system we constructed a feature for destination IP addresses to be analogous of the one for source IP addresses.

3. `ipSrcAddrCommonBits`: Dain and Cunningham say, "An attack scenario may contain components with spoofed source IP addresses while other components of the attack may use the attackers real source IP." As such, they include a feature to indicate the minimal r value found between the source IP of an alert and the source IPs of a group of alerts. We include the exact same feature in our system, but computed such that the feature indicates the minimal value of r between any two source IPs in a cluster.

4. `avgTimeSig` and `varTimeSig`: Dain and Cunningham have features to indicate the similarity in time between when two alerts were generated. In our system we have features to indicate the average time when a group of alerts was generated, and the standard deviation (named `varTimeSig`) for the times when a group of alerts were generated.

5. `avgReconsErr`: Dain and Cunningham have features to indicate whether a new alert is of the same type as the most recent alerts of already established groups. In our system, we have a feature to indicate the average reconstruction error of a group of alerts. As we've established in the previous reports, the reconstruction errors of alerts tend to correspond directly with their type. (Note: this feature isn't available if the clustering algorithm from our previous step wasn't based on the autoassociator.)

There are a few features from the Dain and Cunningham paper that we weren't able to represent in our system. In their system (and in other correlation systems) they include a feature to indicate the similarity of a source IP address and a destination IP address. They suggest that this might be useful in classification because the source IP address of a new alert can be the destination of an old one, if an IP address in your network has been compromised in a previous part of an attack. Because of the structure of their data (the packet captures of a DEFCON hacking contest [28]) it is very unlikely that this feature was used for what they designed it. Attackers didn't use the victim machines as *stepping stones* — a computer used to obscure the true source of an attack — in the contest because the contest wasn't structured that way. In our data we haven't seen any evidence of the use of stepping stones, so while we were able to design a feature to encode this information, we chose not to implement it.

Another set of features they included in their system related to a technical problem of the RealSecure IDS that they used in their experimenting. Apparently this IDS sometimes switches the source and destination IP address in alerts, so they had features to account for that. In our system, using only Snort to experiment with, we didn't see any evidence of this, so we didn't feel there was a problem to overcome by creating new features.

We also experimented with some other features that Dain and Cunningham didn't consider. We list them below in the same format as the previous list.

1. `modePortSrc` and `modePortDest`: We found that using the TCP source and destination ports was sometimes valuable in determining the grouping of a set of alerts. We also found that when the ports were useful, there was almost always a particular source or destination port that was much more common than the rest. Accordingly, we created two features: one to encode the most common TCP source port in a cluster of alerts, and one to encode the most common destination port.

2. `avgSeqNumDiff`: We also found that TCP sequence numbers and TCP acknowledgement numbers are sometimes good predictors of the use of a hacker tool. Often there are obvious patterns in a group of sequence or acknowledgement numbers, and this pattern can usually be encoded by taking the difference of the two numbers. Doing this also helps in the specific situation where there is an explicit relationship between these two sets of numbers within a cluster. We encoded the average difference between these two numbers for each cluster.

3.3.1.1 Formatting Example

We now present an example of how a cluster produced by the first step of our system can be formatted as a data item in our second step. Table 3 contains the following two alerts encoded as described in Section 3.2.1.2.

```
[**] [1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:21.366507 206.48.61.139:4006 -> 170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:24710 IpLen:20 DgmLen:48 DF
*****S* Seq:  0x13D84F7 Ack:  0x0 Win:  0x2000 TcpLen:  28
TCP Options (4) => MSS: 536 NOP NOP SackOK

[**] [1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:24.316507 206.48.61.139:4006 -> 170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:26758 IpLen:20 DgmLen:48 DF
*****S* Seq:  0x13D84F7 Ack:  0x0 Win:  0x2000 TcpLen:  28
TCP Options (4) => MSS: 536 NOP NOP SackOK
```

Table 3: *Two Encoded Alerts for the First Step*

Feature	Alert 1	Alert 2	Feature	Alert 1	Alert 2
portSrc	4006	4006	portDest	3128	3128
ipIsIcmpProtocol	0	0	ipIsIgmpProtocol	0	0
ipIsTcpProtocol	1	1	ipIsUdpProtocol	0	0
ipLen	20	20	ipDgmLen	48	48
ipId	24710	26758	ipTos	0	0
ipTtl	114	114	ipOptLsrr	0	0
ipPacketDefrag	1	1	ipReserveBit	0	0
ipMiniFrag	0	0	ipFragOffset	0	0
ipFragSize	0	0	icmpCode	0	0
icmpId	0	0	icmpSeq	0	0
icmpType	0	0	tcpFlag1	0	0
tcpFlag2	0	0	tcpFlagUrg	0	0
tcpFlagAck	0	0	tcpFlagPsh	0	0
tcpFlagRst	0	0	tcpFlagSyn	1	1
tcpFlagFin	0	0	tcpLen	28	28
tcpWinNum	8192	8192	tcpUrgPtr	0	0
tcpOptMss	536	536	tcpOptNopCount	2	2
tcpOptSackOk	0	0	tcpOptTs1	0	0
tcpOptTs2	0	0	tcpOptWs	0	0
tcpHeaderTrunc	0	0	udpLen	0	0

Then, Table 4 contains the features constructed using the results of the first step and the values in Table 3.

Table 4: *A Cluster Encoded for the Second Step*

Feature	Value	Feature	Value
numAlerts	2	ipSrcAddrCommonPart	3459267979
ipSrcAddrCommonBits	32	ipDestAddrCommonPart	2860586991
ipDestAddrCommonBits	32	modePortSrc	4006
modePortDest	3128	avgIpHdrLen	20.0
avgPayloadLen	48.0	avgReconsErr	6.7834
avgSeqNumDiff	20808951.0	avgTimeSig	974228782.5
varTimeSig	2.12132034356	avgTcpFlagsSet	1.0

4 Results Analysis

We present a number of experiments here, most of which are motivated by details of our model in Section 3. The actual experiments and results are in Section 4.2. How we evaluate the experiments is in Section 4.1, and the type of errors that we encounter are discussed in Section 4.1.3. We give the overall performance of the system in Section 4.3 and we discuss the errors we've made in our experiment in Section 4.4.

4.1 Evaluation

To evaluate the performance of our system we created a *gold standard* to dictate the type of results we want to see produced automatically by our system. When creating the gold standard we tried to be faithful to the notion that one *super-cluster* — a cluster produced by the second correlation step of our system: a cluster of clusters — should represent one attack attempt against the network. We give more details on the gold standard in Section 4.1.1 and we talk about how we use the gold standard in Section 4.1.2.

We draw the gold standard dataset from the incidents.org dataset [43]. The characteristics of this dataset are discussed in Section 3.1. The 500 alerts used in the gold standard are the first half of the subset of 1000 alerts reserved for evaluation and testing from the start of the dataset. These 500 alerts are organized into clusters, as we discuss in Section 4.1.1.

We note that we use the word *evaluate* in this report to mean "to gauge the performance of, with the intent to attain better performance under the same conditions by modifying the system in question". We evaluate our system with an experimental set of parameters or components with the knowledge that we're trying to maximize the performance for a particular *evaluation dataset*. (Evaluation datasets are also called *validation datasets*.) We contrast this against *testing datasets*, whose purpose it is to indendently report on performance after optimization on the *evaluation dataset* has been completed. So, in the section on our experiments, Section 4.2, we use evaluation datasets, but in the section on overall system performance, Section 4.3, we use a testing dataset.

4.1.1 Gold Standard for Performance

We use a gold standard to shape the output of our system to what we want to see. We've created an optimal set of clusters from a set of five consecutive 100 alert windows of the incidents.org dataset [43]. In creating this optimal set of clusters that we'll evaluate our system against, we're forcing our system to become like the gold standard, since if it does so, it performs better. Another group of researchers in the field of alert correlation who have created optimal groupings of alerts to which they compare their the performance of their system is Dain and Cunningham [27].

We do not include more than just a small part of the gold standard, because it is so large. There are 25 super-clusters in the gold standard. We include part of the gold standard in Section E.6. We've included the full gold standard with the package of software for this research given to DRDC as part of this contract.

We created this complete gold standard, for the complete two-step correlation system, based on the gold standards we'd created for the first correlation step for each of the five 100-alert windows. Since each of the 100-alert windows had their own gold standard, the gold standard for whole system represents each of the 100-alert window gold standard clusters as a single number. For instance, super-cluster 1 of the complete gold standard has six members: 1.1, 1.2, 1.3, 2.5, 4.12, and 4.13. In this notation, 1.2 stands for cluster 2 of 100-alert window 1, 4.12 stands for cluster 12 of 100-alert window 4, *et cetera*. This was a convenient way to encode the relationships between clusters within super-clusters.

4.1.2 Results Comparison Against Gold Standard

We also need to explain a bit about the evaluation of our performance in the two-step correlation system. In our correlation system we had the option of scoring how well the second step of our system does given the first step. We could have matched clusters produced by our first step to their optimal gold standard clusters, then evaluated how well our system did with respect to the gold standard super-clusters. We chose not to use this system of error-counting, though, because it would have been fraught with problems. There is no perfect match between our gold standard clusters and the clusters produced by our old system. Inevitably, smaller clusters produced by our old system would not have been counted, so we wouldn't have had a very realistic evaluation of our system.

We decided to evaluate the performance of our complete system by translating the super-clusters in the set of alerts represented by the sub-clusters in our super-clusters. For instance, using the example given previously, if super-cluster 1 has sub-clusters 1.1, 1.2, 1.3, 2.5, 4.12, and 4.13, and sub-cluster 1.1 has alerts 1, 2, 3, 4, 5, 6, 8, and 10, sub-cluster 1.2 has alert 9, sub-cluster 2.5 has alert 146, *et cetera*, then we translate super-cluster 1 into the set of alerts 1, 2, 3, 4, 5, 6, 8, 10, 9, 7, 146, 303, and 324. Once this translation is done, we can compare all of the alerts of our gold standard of super-clusters to the actual results by simply counting the errors.

By reporting our results in this way, we get a good sense of the overall performance of our system, since two alerts are counted as successfully being clustered together if and only if they appear in the same, correct super-cluster.

4.1.3 Types of Errors

We counted the number of errors we made by counting the number of alerts which were correctly clustered versus the number of alerts which were incorrectly clustered for a sample of test data. (We use the gold standard from Section 4.1.1 to determine which alerts are errors.) There are other ways of counting errors — such as counting the number of completely correct clusters — but we felt our method was the most accurate gauge of our performance.

We differentiate between two types of errors in our analysis because they are of different levels of importance, and because they have different causes. We call the first type of error *separation error*, which occurs when a distinct group of alerts found by the candidate algorithm should belong to a larger group of related alerts, but isn't clustered with the larger group. For instance, if we have a cluster A of 10 nmap scan alerts and a separate cluster B of 3 nmap alerts that are clearly related to cluster A, then we count 3 separation errors because the 3 alerts in B should have been grouped with the 10 alerts in A. We consider this type of error less damaging because, if this error occurs, the intrusion analyst must look at more clusters, but he will not miss important alerts hidden in a larger group.

The second type of error we find is *clustering error*, which occurs when two unrelated alerts are clustered together. For instance, if we have a mixed cluster of 10 nmap scans and 5 Squid scans where the alerts of the two types in the cluster are obviously not related, we count 5 clustering errors. This type of error indicates the problem of the system finding correlation where there is none. We consider this type of error more serious because it may cause an intrusion detection analyst to miss important information, if the analyst is relying on the results of the alert correlation system. Any alert correlation system should only make the intrusion detection analyst's job easier, but clustering errors could make the analyst's job harder by hiding important alerts. More dangerously, an attacker could possibly craft an attack to avoid detection by somehow forcing important alerts to not be seen by the IDS analysts.

4.2 Experiments

Here we present the results of a number of experiments we've performed. In Section 4.2.1 we explore the problem of the 40-to-1 mapping that we discussed in our previous report for Autocorrel I [1]. In Section 4.2.2 we discover the role of the 10,000 unlabelled training alerts in Autocorrel I. In Section 4.2.3 we experiment with data different scaling methods and try to determine which one is optimal. In Section 4.2.4 and Section 4.2.5 we try to find the best algorithms and parameters for the first and second correlation steps of our system. In Section 4.2.6 we present an experiment that shows the role of the 10,000 unlabelled training alerts in making the reconstruction error values produced by the autoassociator more numerically stable.

To clarify, the experiments in Section 4.2.5 and in the performance testing Section 4.3 are the only experiments that deal with the entire system. All of the other experiments in this section deal with only the first stage of our system, which is a refined Autocorrel I. This set of experiments is designed to discover and prove different aspects about the Autocorrel I system.

4.2.1 Feature Mapping Problem

To conclusively prove that the 40-to-1 mapping is a significant source of errors, we devised and ran an experiment. To perform the experiment we analyzed the errors produced by the Autocorrel I system to determine whether they were caused by the 40-to-1 feature mapping. We used an evaluation dataset of 100 alerts for this experiment.

We are most concerned with the errors where intuitively dissimilar alerts are grouped together. We call these types of errors clustering errors (see Section 4.1.3) because they indicate the problem of the system finding correlation where there is none.

We hypothesize that some of the clustering errors we've seen in our tests of the Autocorrel I system are caused the 40-to-1 mapping, but we do not know the percentage of clustering errors. To calculate the percentage of clustering errors caused by the mapping, we need to determine whether each individual clustering error was caused by the mapping problem, then simply count the number of clustering errors caused by the mapping problem and compare that to the total number of clustering errors.

We note that the reconstruction error formula can't cause any separation errors, the other type of error encountered in our clustering system. Separation errors are errors seen when an optimal (gold standard) cluster is split into two or more separate clusters, so our deterministic 40-to-1 mapping, defined by the reconstruction error formula, would always map numerically similar data items to similar domain values. The problem with the reconstruction error formula is that it can map dissimilar data items to similar domain values, and the effects of this are only clustering errors.

To determine whether a particular clustering error is caused by the 40-to-1 mapping we need to look at the values which compose the computed reconstruction error values. Since E_x, from Equation 13, is the output of the 40-to-1 mapping, we look the 40 inputs to this formula, $d_{x_i} = x_i - f_i(x)$. The clusters in the Autocorrel I system are formed by examining the differences in behaviour between mappings of the components of $d_x = [d_{x_1}, \ldots, d_{x_n}]$ values for different data items x. That is, two data items x and y should only be clustered together if the values d_{x_i} and d_{y_i} are reasonably similar for all i. If we have a suitable test of similarity for high-dimensional vectors, we can test if two data items were correctly clustered together.

There exist a number of vector similarity measures to quantify this similarity (Manning and Schütze [48]), for example *cosine similarity* and *Euclidean distance*, also known as *geometric distance*. We chose to use Euclidean distance because the measure is well-known and the results are easily interpretable. Manning and Schütze define Euclidean distance between vectors x and y as follows:

$$|x - y| = \sqrt{\sum_{i=1}^{n} [x_i - y_i]^2} \qquad (14)$$

We use the Euclidean distance measure to measure the distances between all the combinations of two vectors in clusters that contain clustering errors. We computed the distance $|d_x - d_y|$ for all such pairs and found that the distances are significant between intuitively anomalous alerts from a cluster and the rest of the cluster.

When running our experiment we found four clusters with clustering error, meaning that there were four system-produced clusters that contained alerts from two or more gold standard clusters. Clusters 4, 6, 8 and 22 were the four system-produced clusters containing clustering errors. We discuss these clusters below.

Table 5: *Euclidean distances between alerts in cluster 4*

	26	27	82	81	80	**7**	79	78	77	76
26	0.0000	0.0113	0.0299	0.0357	0.0242	**2.8747**	0.0438	0.0542	0.0639	0.0739
27	0.0113	0.0000	0.0188	0.0258	0.0262	**2.8737**	0.0436	0.0534	0.0627	0.0725
82	0.0299	0.0188	0.0000	0.0108	0.0348	**2.8714**	0.0454	0.0531	0.0611	0.0698
81	0.0357	0.0258	0.0108	0.0000	0.0328	**2.8689**	0.0387	0.0451	0.0523	0.0604
80	0.0242	0.0262	0.0348	0.0328	0.0000	**2.8693**	0.0200	0.0304	0.0401	0.0502
7	**2.8747**	**2.8737**	**2.8714**	**2.8689**	**2.8693**	**0.0000**	**2.8647**	**2.8625**	**2.8603**	**2.8581**
79	0.0438	0.0436	0.0454	0.0387	0.0200	**2.8647**	0.0000	0.0104	0.0202	0.0302
78	0.0542	0.0534	0.0531	0.0451	0.0304	**2.8625**	0.0104	0.0000	0.0098	0.0198
77	0.0639	0.0627	0.0611	0.0523	0.0401	**2.8603**	0.0202	0.0098	0.0000	0.0101
76	0.0739	0.0725	0.0698	0.0604	0.0502	**2.8581**	0.0302	0.0198	0.0101	0.0000

Table 5 shows a matrix of distances between the alerts of cluster 4. Each of the alerts in the dataset is assigned a unique index number for easy reference, and the numbers of the alerts in this cluster are 26, 27, 82, 81, 80, 7, 79, 78, 77, and 76. The anomalous alert is alert 7, which is an alert of type “Short UDP Packet”. (The distances in Table 5 that involve the anomalous alert, alert 7, are typeset in **bold**.) Here's the Snort representation of this alert:

```
[**][116:97:1] Short UDP packet, length field > payload length [**]
11/11-13:29:54.796507 211.194.68.39:0->207.166.72.218:0
UDP TTL:109 TOS:0x0 ID:2062 IpLen:20 DgmLen:78
Len:129
```

The rest of the alerts in this cluster are "Scan nmap TCP" alerts, which are correctly clustered together. Here's the Snort representation of alert 26, a sample of this type of alert:

```
[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:45.976507 167.79.91.3:80->170.129.50.122:53
TCP TTL:49 TOS:0x0 ID:11661 IpLen:20 DgmLen:40
**A**** Seq:0x2A5 Ack:0x0 Win:0x578 TcpLen:20
```

From Table 5 you can see that the Euclidean distance between alert 7 and the rest of the alerts in the cluster ranges between 2.85 and 2.88, whereas the distance between any two of the rest is at most 0.07.

Alert 7 is clearly numerically anomalous in this cluster, yet Autocorrel I has clustered it with the other alerts because the reconstruction error value of alert 7 is similar to the reconstruction error values of the other alerts in the cluster. This shows that the reconstruction error formula is responsible for the clustering error in cluster 4.

Table 6: *Euclidean distances between alerts in cluster 6*

	28	29	30	31	**59**	32	33	**58**
28	0.0000	0.0114	0.0224	0.0342	**3.4449**	0.0655	0.0762	**3.4049**
29	0.0114	0.0000	0.0110	0.0228	**3.4431**	0.0543	0.0649	**3.4027**
30	0.0224	0.0110	0.0000	0.0118	**3.4415**	0.0436	0.0542	**3.4006**
31	0.0342	0.0228	0.0118	0.0000	**3.4398**	0.0324	0.0427	**3.3984**
59	**3.4449**	**3.4431**	**3.4415**	**3.4398**	**0.0000**	**3.4353**	**3.4339**	**0.1864**
32	0.0655	0.0543	0.0436	0.0324	**3.4353**	0.0000	0.0108	**3.3927**
33	0.0762	0.0649	0.0542	0.0427	**3.4339**	0.0108	0.0000	**3.3909**
58	**3.4049**	**3.4027**	**3.4006**	**3.3984**	**0.1864**	**3.3927**	**3.3909**	**0.0000**

The matrix in Table 6 shows the distances between the alerts of cluster 6. (Again, the distances involving anomalous alerts in this table are typeset in **bold**.) In this cluster there are two anomalous alerts that contribute to the clustering errors. The alerts 58 and 59 are anomalous; alert 58 is of type "SCAN Proxy Port 8080 attempt" and alert 59 is of type "SCAN Squid Proxy attempt". The other alerts in this cluster are “Scan nmap TCP” alerts, similar to alert 26 that was previously shown. Here are the Snort representations of alerts 58 and 59:

```
[**][1:620:6] SCAN Proxy Port 8080 attempt [**]
11/14-16:00:56.996507 66.159.18.66:43517->170.129.50.120:8080
TCP TTL:53 TOS:0x0 ID:59575 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBED8745 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

```
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43518->170.129.50.120:3128
TCP TTL:53 TOS:0x0 ID:50174 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF3AC0C Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

This cluster is interesting because the two anomalous alerts should actually be correlated together, but not with the rest of the alerts in the cluster. There should be a division placed between the alerts 58 and 59, and the other alerts in the cluster. We know that alert 58 should probably be correlated with alert 59 because the Euclidean distance between the two alerts is about 0.19. In contrast, the distance between the other alerts in the cluster and alerts 58 or 59 is at least 3.40. The distance between any of the two other alerts is at most 0.08, showing that they are correctly clustered together. We can conclude again that the two anomalous alerts are responsible for the clustering errors in cluster 6.

Table 7: *Euclidean distances between alerts in cluster 8*

	24	49	57	73	**60**
24	0.0000	0.0000	0.0000	0.0000	**3.7026**
49	0.0000	0.0000	0.0000	0.0000	**3.7026**
57	0.0000	0.0000	0.0000	0.0000	**3.7026**
73	0.0000	0.0000	0.0000	0.0000	**3.7026**
60	**3.7026**	**3.7026**	**3.7026**	**3.7026**	**0.0000**

The matrix of real values in Table 7 shows the distances between the alerts of cluster 8. There is one anomalous alert in this cluster, alert 60, of type “SCAN SOCKS proxy attempt”. The other alerts in cluster 8 are of type “BAD-TRAFFIC ip reserved bit set”. Here’s the Snort representation of alert 60:

```
[**][1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43520->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:4253 IpLen:20 DgmLen:60 DF
*****S* Seq:0xB9A7F6F Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

And here's the representation of alert 24, typical of the other alerts in the cluster:

```
[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-11:21:09.916507 200.200.200.1->170.129.211.200
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014
```

This cluster is interesting because it is such an extreme example of how the mapping problem caused by the reconstruction error formula causes clustering errors in our system. The Euclidean distance between 60 and all the other alerts is 3.7026 and the Euclidean distance between any two of the other alerts is 0. The distance is exactly 0 because the "BAD-TRAFFIC ip reserved bit set" alerts in this cluster all have exactly the same numerical representation. This is a very strong signal that these alerts should be clustered together, and it's obvious that alert 60 is anomalous to this set.

Table 8: *Euclidean distances between alerts in cluster 22*

	40	9
40	**0.0000**	**3.0659**
9	**3.0659**	0.0000

The last cluster containing clustering errors is cluster 22. This cluster contains only two alerts, and the Euclidean distance between the two alerts is 3.0659. (We include this value as a matrix in Table 8 to be consistent with the analysis performed of the other clusters.) Alert 40 is a "BARE BYTE UNICODE ENCODING" alert and alert 9 is a "TCP Data Offset is less than 5" alert. Here are the Snort representation of these two alerts:

```
[**][116:46:1] TCP Data Offset is less than 5!  [**]
11/12-18:38:17.846507 203.80.239.162:0->207.166.182.137:0
TCP TTL:107 TOS:0x0 ID:35119 IpLen:20 DgmLen:48 DF
1*UA**** Seq:0x7930005 Ack:0xD80A04D1 Win:0x64BA TcpLen:0 UrgPtr:0x800
```

```
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-13:03:42.666507 170.129.50.120:63598->64.4.22.250:80
TCP TTL:124 TOS:0x0 ID:56064 IpLen:20 DgmLen:932 DF
**AP*** Seq:0x94851318 Ack:0x9AB18054 Win:0x43E1 TcpLen:20
```

It should be clear that these two alerts being clustered together constitute a clustering error. As mentioned, the Euclidean distance is 3.0659. Clearly these alerts lay some distance apart in 40-dimensional space, but Autocorrel I has clustered them together because their reconstruction error values are similar. This small cluster is a very simple example of how the reconstruction error formula causes clustering errors.

In conclusion, all of the clustering errors in this sample clustering were caused by the reconstruction error formula incorrectly mapping from a 40-dimensional space to a single dimension.

Clearly the errors caused by this 40-to-1 mapping are significant. While the sample size of this experiment is small, we believe it is representative, and that the reconstruction error formula accounts for a significant portion of the clustering errors produced by Autocorrel I. To correct the 40-to-1 mapping problem we considered using a different clustering algorithm (see Section 4.2.4) but we found that other clustering algorithms performed even poorer than the autoassociator without the 40-to-1 problem fixed.

4.2.2 Autoassociator Training

We want to investigate the effect of training the autoassociator in the Autocorrel I system. To do this, we propose an experiment to fully analyze the impact of using a large number of unlabelled alert data to train our system.

Training is a common process for most data mining and machine learning applications. Training in data mining or machine learning represents the learning part of the process; this idea is based on the fact that, in most applications, the past is a good predictor of the future. But in the domain of alert correlation, this assumption is violated. Julisch *et al* [26] explicitly mention the idea that traditional data mining training may not be valuable in the alert correlation domain. McHugh [49] mentions the fact that each site that runs an IDS will have a unique distribution of alarm types. Julisch *et al* strengthen this by showing that even a given network's distribution of alerts changes from month-to-month.

The papers by Julisch *et al* [26] and Dain *et al* [27] represent the most serious attempts by researchers to apply machine learning algorithms to the domain of alert correlation. Both of these papers rely to some extent on supervised training using labelled datasets, and both papers present novel ideas for doing so. For our system we wish to investigate whether our type of unsupervised training is necessary for reasonable performance of a correlation system.

We do this by considering a variation of the Autocorrel I system that is not trained using the 10,000 unlabelled data instances, unlike the original Autocorrel I. For this experiment we train the autoassociator using the evaluation dataset, then reconstruct the same data on the neural network after training has completed. What we expect is that the autoassociator may be able to find more subtle numerical differences between data items, since it has intimately learned the data. We also expect that the reconstruction errors for the data items will be less stable, but we explore this hypothesis in Section 4.2.6.

This experiment will show the effect of the 10,000 item training dataset on the performance of Autocorrel I. We compare the new system variation to the results of Autocorrel I on the basis of performance.

For this experiment, we examine the system with 8, 16, and 32 hidden units. We evaluate the performance against the evaluation data at 500, 2500 and 5000 epochs. We use self-organizing maps, rather than the cluster barrier algorithm, to form discrete clusters with the autoassociator output data. The self-organizing maps are trained for 1000 epochs on the same data that they cluster. The self-organizing maps are configured in a 4 × 6 lattice.

In this section we present results for both the incidents.org [43] and 1999 DARPA [17] datasets. The results of different clustering algorithms are only comparable if the evaluation dataset remains invariant because different sets of data have different innate characteristics. Bear this in mind when considering the performance of the system. We see the statistics for the 10,000 alert training data in Table 9 for the incidents.org data, and we see the statistics the same experiment with the 1999 DARPA dataset in Table 10. The results for the "no training data" experiment on the incidents.org dataset are in Table 11 and on the 1999 DARPA dataset in Table 12.

Table 9: *Results of the autoassociator and self-organizing maps on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	34	6	28	33	4	29	31	3	28
16	33	4	29	33	4	29	34	4	30
32	29	4	25	28	2	26	28	5	23
64	35	4	31	30	4	26	30	4	26

Table 10: *Results of autoassociator and self-organizing maps on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	62	4	58	61	4	57	57	7	50
16	61	3	58	61	2	59	59	3	56
32	58	6	52	60	9	51	59	2	57
64	60	9	51	58	6	52	58	5	53

We see from Table 13 that the "no training set" system produces very similar averages to the averages produced by the equivalent systems using the 10,000 unlabelled training data. (For the statistics in this table, mean averages are in normal font and standard deviations are in *italics*.)

Table 11: *Results of autoassociator trained on evaluation data, clustered with self-organizing maps, on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	33	5	28	34	6	28	34	4	30
16	29	2	27	31	2	29	28	2	26
32	27	1	26	32	5	27	28	2	26

Table 12: *Results of autoassociator trained on evaluation data, clustered with self-organizing maps, on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	63	3	60	60	6	54	61	3	58
16	61	8	53	58	2	56	63	9	54
32	62	3	59	64	14	50	65	6	59

Table 13: *Average errors for both datasets*

	Average Errors		
Dataset	TE	CE	SE
incidents.org (with training set)	31.5 (*6.1*)	4	27.5
1999 DARPA (with training set)	59.5 (*2.5*)	5	54.5
incidents.org (without training set)	30.7 (*7.5*)	3.2	27.4
1999 DARPA (without training set)	61.9 (*4.7*)	6	55.9

From Table 11 we see that the autoassociator using the self-organizing maps clustering algorithm produces an average number of errors of 31.5, whereas the “no training set” version produces 30.7 errors on average. Also, from Table 12 we see that 10,000 data trained system produces 59.5 errors on average, and the comparable “no training set” system produces 61.9 errors on average.

This experiment shows that the system doesn’t particularly benefit and isn’t impaired by the use of the unlabelled training data. The fact that the averages are so close for the two datasets shows that the training data doesn’t affect performance. Note that this says nothing of the stability of the reconstruction error values, though, which we explore in depth in Section 4.2.6.

We acknowledge a small methodology problem with the previous experiment though. We didn’t test the autoassociator with 64 hidden units in the “no training set” experiments. We believe this is only a minor problem, though, since it is unlikely that the test with 64 hidden units would have varied wildly from the other tests.

4.2.3 Data Scaling Algorithms

In this experiment we’ll evaluate some different scaling methods, particularly those discussed in Section 3.2.1.3. In artificial neural networks it is often important to scale data so that computational errors are don’t occur in neural network training. In our experiments we will evaluate different scaling methods with respect to system performance.

The experiments we describe here are intended to show two things. The first thing we’d like to show is the importance of scaling the data. We show this by showing how our system performs with unscaled data. We compare the results to the results of our Autocorrel I system. The second thing we’d like to determine is which scaling method performs best. The scaling algorithms we consider make the overall system perform differently and we find that the cluster barrier parameter needs to be adjusted for each different algorithm. We vary this parameter for our experiments to give an idea of the potential for each scheme.

For the results of this section we use the five 100 alert evaluation clusterings from the *incidents.org* [43] dataset from the gold standard, in Section 4.1.1. We determine the performance of our system on each of these five evaluation sets and we average the errors produced by each system to gain a better view of how the system performs. The average number of errors is the percentage of the total possible number of errors. The system with the lowest percentage of errors is considered the best.

We expected that we’d have to adjust the cluster barrier parameter for this experiment by necessity because this parameter is directly dependent on how the data is encoded and scaled. We do this graphically by showing the percentage of total errors, clustering errors and separations errors (see Section 4.1.3 for definitions of these types of errors) plotted against the value for the cluster barrier parameter for the system in question.

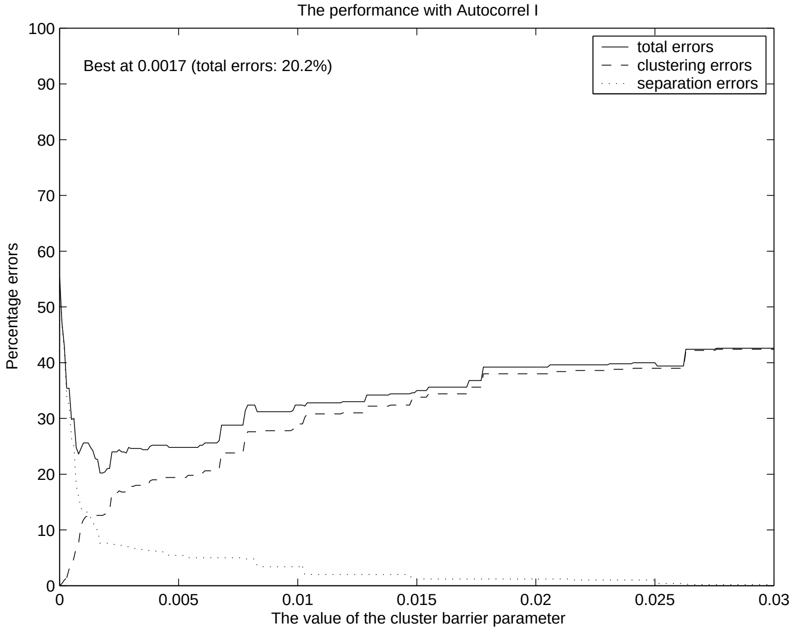

Figure 5: *The performance of Autocorrel I system, varying the cluster barrier parameter*

Figure 5 shows how our Autocorrel I system performs (with the bug where the training set scaling data wasn't used to scale the evaluation sets). (The Autocorrel I system scales features to a $[0, 1]$ interval, it should be noted here.) Figure 6 shows the exact same system as Figure 5 except with this bug removed. As you can see from the these graphs, the best performance is attained with cluster barrier parameters 0.0017 and 0.0015 respectively. These optimal values are fairly close to the value used in our Autocorrel I system, 0.0025. It is interesting to note that correcting the bug has given better performance, but the performance in the second graph also worsens quicker, implying that it's more important to get the cluster barrier correct, which of course is impossible in a deployed system where there is not optimal correlation sets to reference.

The last variation of a system scaling to the interval $[0, 1]$ is seen in Figure 7. For this graph we used the predefined feature ranges we discussed in Section 3.2.1.3. You can see from the graph that this system performs noticably worse than the other systems linearly scaling to this interval.

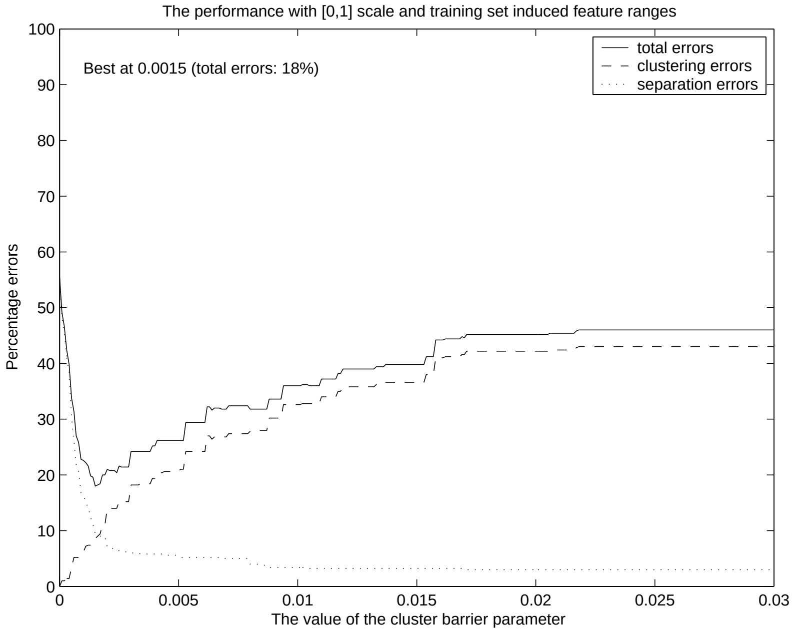

Figure 6: *The performance of the system using a [0,1] scale with the training set to determine ranges*

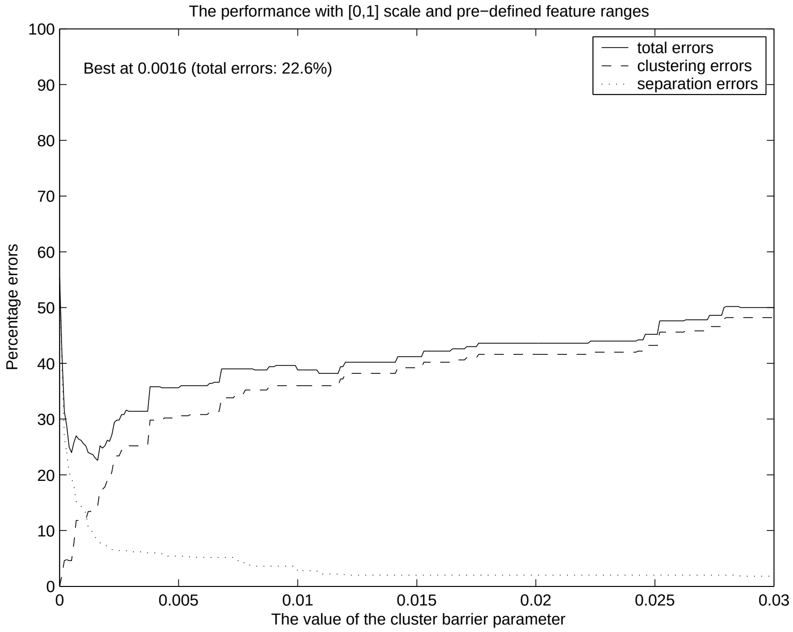

Figure 7: *The performance of the system using a [0,1] scale and predefined feature ranges*

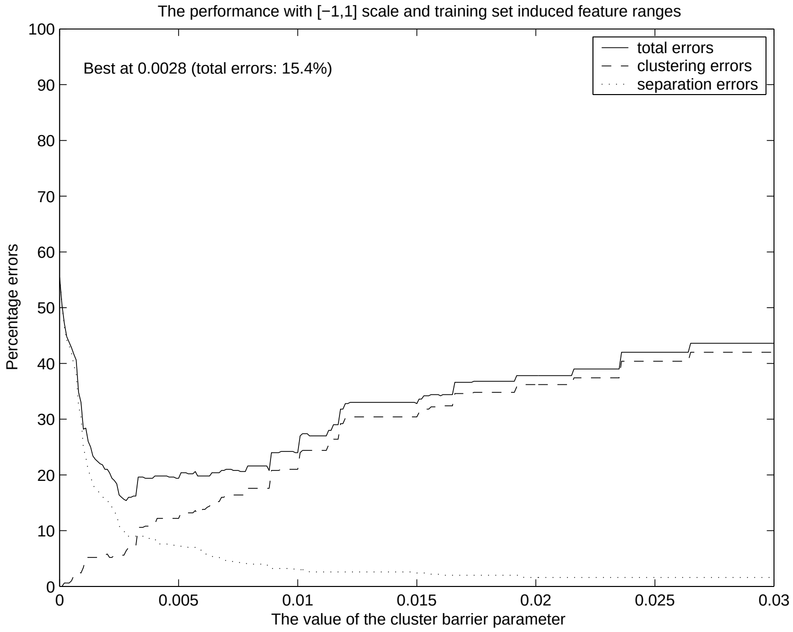

Figure 8: *The performance of the system using a [-1,1] scale with the training set to determine ranges*

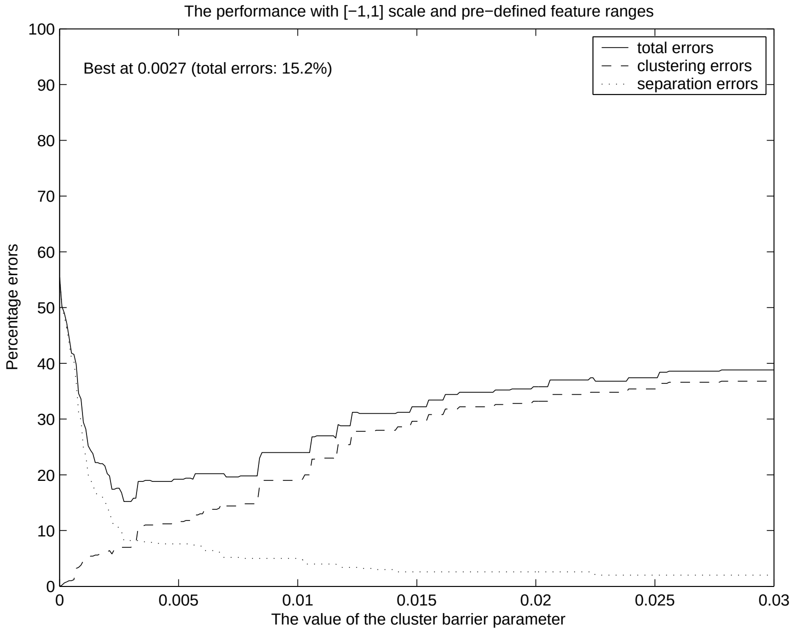

Figure 9: *The performance of the system using a [-1,1] scale and predefined feature ranges*

Figure 8 shows the performance for the system scaling to the $[-1, 1]$ interval using *high* and *low* values from the training set. Figure 9 shows the performance for this same interval but with the predefined *high* and *low* feature range values. Figure 9 showed the lowest error rate of any of the systems we examined, and Figure 8 was so close to this that the difference isn't statistically significant. The two graphs show essentially the same performance, and they show this with almost the same value for the cluster barrier parameter, 0.0028 and 0.0027.

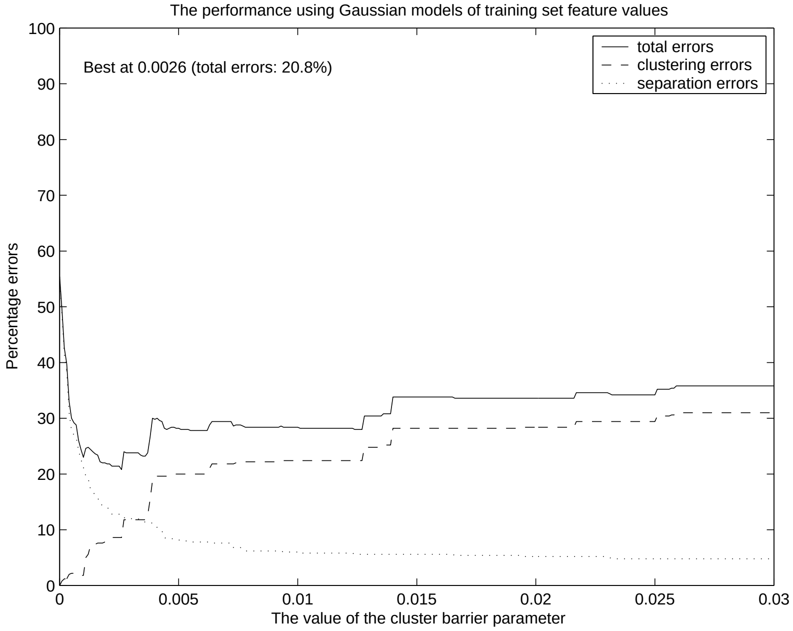

Figure 10: *The performance of the system using Gaussian models for features determined from the training set*

Figure 10 shows how well our system performed with the Gaussian scaling method from Section 3.2.1.3. As you can see from this graph, the best results are not as good as the other methods, but it seems that the performance was less affected by the value for the cluster barrier parameter, because larger values for this parameter didn't produce significantly worse results, unlike in the other systems. We found this interesting, but we do not choose this system of scaling because we plan to fix the cluster barrier parameter at a lower value anyway.

Lastly we tried running our system on unscaled datasets. The results were so poor (55% error rate in the best system) that we do not present them in graphical form here. The results hardly varied at all with the cluster barrier parameter.

We conclude that selection of which scaling method to use is important. We also conclude that linearly scaling to the interval $[-1, 1]$ (using either of the methods we presented results for) at a cluster barrier parameter value of about 0.0025 produces the best results.

4.2.4 First Correlation Step Parameter Exploration

In this experiment we want to discover the best performance for the first correlation step of our system, described in Section 3.2. We vary two autoassociator parameters and examine three candidate clustering algorithms to determine the best performing system.

We compare the performance — the number of clustering and separation errors — produced by various combinations of the machine learning algorithms on two datasets. Namely, we vary the autoassociator's neural network parameters *hidden units* and *epochs*, and we feed the 40 autoassociator outputs to three clustering algorithms: the EM algorithm, self-organizing maps, and the single-link clustering algorithm that we previously used.

To compare the performance of the various modifications of our system, we've taken 100-consecutive alert windows from both the 1999 DARPA dataset (Lippmann *et al* [17]) and the incidents.org dataset [43]. We take these 100 alerts from the golden standard that we've created (see Section 4.1.1).

The hidden units parameter in neural networks defines the learning capacity of the network. The hidden unit layer of a multi-layer perceptron neural network is responsible for allowing the learning of non-linear concepts. There exists a (possibly non-unique) optimal number of hidden units in a neural network for a given training set. By tuning this neural network parameter, one can approximate the optimal number of hidden units, and thus attain better performance in the particular domain. In the Autocorrel I system we fixed the number of hidden units of the autoassociator at 8 because we didn't see any significant differences in the performance of the system at different values. For this experiment we take a more rigourous approach and compare the output of our system against predefined desired outputs — the gold standard. Because of the existence of a gold standard, we consider the following four candidate hidden units numbers: 8, 16, 32, and 64.

The epochs parameter in neural networks defines the number of iterations of the neural network training algorithm. As the neural network becomes more trained, the neural network is more capable of recognizing the particular data on which it's being trained. One notable pitfall of the training process is that if the neural network is trained for too many epochs, then *overfitting* may occur. An overfitted neural network is a neural network that has been trained for more epochs than it should have for optimal performance. It described the state of the network when it will no longer generalize to the testing data (or other data that is at all different from the training data), and thus cannot be used as a general classifier (or clustering algorithm). (See Pattern Classification by Duda, Hart and Stork [45] for a more detailed analysis of the problem of overfitting.) For this report, we examine three candidate epoch values for the autoassociator: 500, 2500, and 5000.

Since the self-organizing maps algorithm has parameters to be chosen, we include the values we've chosen for them. The number of training epochs is fixed at 1000 and the dimensions of the map lattice are 4 × 6.

In this experiment we compare the results of clustering the incidents.org evaluation dataset of 100 alerts using a trained autoassociator and three different clustering algorithms: the single-link heuristic algorithm we developed for Autocorrel I, the EM algorithm, and self-organizing maps. In the following tables, we list the total number of errors (TE), the clustering errors (CE), and the separation errors (SE).

Table 14: *Results of the autoassociator and the single-link algorithm on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	29	8	21	37	3	34	37	3	34
16	47	8	39	48	8	40	47	6	41
32	42	2	40	43	1	42	38	0	38
64	40	0	40	40	1	39	43	5	38

Table 15: *Results of the autoassociator and the EM algorithm on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	32	4	28	39	3	36	43	2	41
16	44	0	44	44	0	44	44	0	44
32	39	2	37	40	0	40	46	2	44
64	39	2	37	43	1	42	38	1	37

We see from Tables 14, 15, and 16 that the lowest number of total errors we achieved is 28, achieved by self-organizing maps clustering the output of the autoassociator with 32 hidden units trained for 2500 and 5000 epochs. We see that this system, if trained for 2500 epochs, has fewer clustering errors than the system trained for 5000 epochs. Since we regard clustering errors as more harmful than separation errors — we have discussed why there is a difference in importance of these two types of errors in previous reports — we can conclude that the best system performance in this experiment was attained by the self-organizing maps with an autoassociator trained for 2500 epochs.

Table 16: *Results of the autoassociator and self-organizing maps on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	34	6	28	33	4	29	31	3	28
16	33	4	29	33	4	29	34	4	30
32	29	4	25	28	2	26	28	5	23
64	35	4	31	30	4	26	30	4	26

To analyze the effectiveness of the three clustering algorithms, it might be useful to consider the averages of total errors and clustering errors for each of Tables 14, 15, and 16, to show the general trends of the algorithms. We show these averages in Table 17. (Note that *SOM* abbreviates self-organizing maps.) The values in *italics* are the variance measures of the total errors.

Table 17: *Clustering algorithm average errors for incidents.org dataset*

	Average Errors		
Clustering Algorithm	TE	CE	SE
Single-link	40.9 (*28.8*)	3.8	37.2
EM	40.9 (*14.8*)	1.4	39.5
SOM	31.5 (*6.1*)	4	27.5

We see from Table 17 that self-organizing maps clearly have the lowest overall average number of errors. Another interesting point of this table is that the EM algorithm produces a lower average number of clustering errors than the other two clustering algorithms.

Next, we use the same algorithms with the same parameters to test which system performs best with the 1999 DARPA dataset. In Tables 18, 19, and 20 we see that all of the clustering algorithms preform poorer than on the incidents.org dataset. The reason for this is that our gold standard for the 1999 DARPA differs from the innate structures in the dataset.

While we may not be able to achieve precisely the gold standard for this dataset, the clusters produced by the systems are still valuable. For instance, we see that the number of clustering errors is fairly low, implying that the clustering algorithms found differences within clusters of the gold standard.

We see that our best preformance lies with the EM algorithm on this dataset, with 16 hidden units in the autoassociator. The system performs equally well at both 2500 epochs and 5000 epochs.

Table 18: *Results of autoassociator and the single-link algorithm on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	52	6	46	52	0	52	52	4	48
16	52	2	50	50	5	45	48	1	47
32	51	4	47	50	2	48	51	0	51
64	48	0	48	50	0	50	50	0	50

Table 19: *Results of autoassociator and the EM algorithm on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	44	3	41	61	0	61	62	3	59
16	63	4	59	43	3	40	43	3	40
32	44	8	36	51	4	47	60	3	57
64	65	3	62	45	3	42	64	3	61

Table 20: *Results of autoassociator and self-organizing maps on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	62	4	58	61	4	57	57	7	50
16	61	3	58	61	2	59	59	3	56
32	58	6	52	60	9	51	59	2	57
64	60	9	51	58	6	52	58	5	53

Although the EM algorithm produces the best single system for the 1999 DARPA dataset, the results produced by this algorithm vary significantly more than the results produced by self-organizing maps and the single-link clustering algorithm, indicating that the results produced by this algorithm may be less stable. We see in Table 21 that the EM algorithm was responsible for the best performing system, but that the single-link clustering algorithm outperformed the EM algorithm on average.

Table 21: *Clustering algorithm average errors for 1999 DARPA dataset*

	Average Errors		
Clustering Algorithm	TE	CE	SE
Single-link	50.5 (*2.1*)	2	48.5
EM	53.8 (*89.3*)	3.3	50.4
SOM	59.5 (*2.5*)	5	54.5

From Table 21 we see that the single link algorithm has both the lowest average total errors as well as the lower average clustering errors. Another interesting fact presented in this table is that self-organizing maps performed the poorest on this dataset. They had both the highest average total error and the highest average clustering error. We find this interesting, because they performed the best on the incidents.org dataset.

One thing we notice immediately from the tables with self-organizing map results in Tables 16 and 20 is that the results vary less than the other results. Table 21 reports the variance in the total error statistic for the three algorithms on the two datasets in *italics*. From this we can hypothesize that self-organizing maps form more stable clusters, which are less likely to be changed by the intermediate step of using the autoassociator. This could be considered both a positive and a negative feature of the algorithm. Stability of results is good because it can imply a predictable end system, but it can be bad at the research stage if different results are needed and the output of the algorithm won't produce them.

One last thing we'd like to point out about the previous experiments is that the single-link algorithm seems to preform much better if the autoassociator is constructed with 32 or 64 hidden units. One can see in Tables 14 and 18 that with 64 hidden units at 500 epochs, in particular, the single-link algorithm produces very few clustering errors. This is a very desirable property of any algorithm we choose, so we'll have to continue considering the single-link autoassociator combo in future research.

Another desirable trait of the autoassociator algorithm combined with the cluster barrier algorithm is that there is no need to determine the number of clusters to output before the clustering has started. This is automatic with the cluster barrier algorithm. A possible disadvantage of the cluster barrier is that it requires a parameter to tune the shape of the clusters, but this addressed in Section 4.2.6.

4.2.5 Second Correlation Step Parameter Exploration

For this experiment we want to explore the clustering algorithms available to us and their respective parameters to find a system that will produce the best possible clusters for the second correlation step. We define the performance of the systems we explore as the number of errors the system produces with respect to the gold standard from Section 4.1.1.

In our experiments with first correlation step of the Autocorrel I system, we found that if we used a base window of 100 alerts, we would produce about 25 clusters with our system. Obviously this figure would hopefully vary dramatically with the dataset we're using (and thus with the number of innate clusters in the data), but we found this to be a good estimate in general. For the second correlation step of our system, where each cluster produced by the first step becomes its own data item, finding clustering in a set of only 25 data items wouldn't give us meaningful results. So we decided to run our first correlation step, with the parameters we discovered in other experiments (especially those in Section 4.2.4), on the five consecutive 100-alert windows that were used to create our gold standard in Section 4.1.1. Once the first correlation step had produced clusters for each of these five windows, we ran our new feature construction methods on each of the new clusters then combined all of the data produced into one dataset. We didn't run Autocorrel I on just the 500-alert window because we would have had to modify our previously created gold standards when we devised this experiment, and because we might have had to spend time experimenting with parameters — notably the cluster barrier parameter used in our one-dimensional single-link clustering. (See Jain *et al* [47] for a description of this type of algorithm.)

We ended up with 83 data items for our new dataset. Since the purpose of this experiment was to explore feature construction for this dataset, we hold the number of data items in the set static, and just experiment by analyzing the output when we select particular features that we've constructed.

Our results in this experiments are based on the results of clustering already flawed data, which causes errors to propagate. We could have chosen to use the five 100-alert window gold standard clusters as our the base data for this experiment — effectively creating super-clusters from this already prefectly clustered data — but we felt that this wouldn't give a realistic indication of the performance of our system. This would have been an interesting experiment, though, because it would have allowed us to see how well this new part of our system works independently of the old part.

For this experiment we test two variations of our system for the second correlation step. We test our system with all of the features we've constructed in Section 3.3.1 and we also test our system with only the features taken from Dain and Cunningham paper [27]. So for the latter test we use the following features: ipSrcAddrCommonPart, ipDestAddrCommonPart, ipSrcAddrCommonBits, avgTimeSig, varTimeSig, and avgReconsErr. For the former test we use all of the Dain and Cunningham features as well as some others, to form the feature set: ipSrcAddrCommonPart, ipDestAddrCommonPart, ipSrcAddrCommonBits, avgTimeSig, varTimeSig, avgReconsErr, modePortSrc, modePortDest, and avgSeqNumDiff.

We want to test which clustering algorithms perform best for our new dataset. To this end, we experiment with all of the autoassociator [35], the EM algorithm [36], and self-organizing maps [38]. We are guided by some of the results from the experiments of Section 4.2.4. For instance we know that the autoassociator, using the single-link, one-dimensional clustering algorithm to do the actual clustering [47], seems to work well with 64 hidden units. Also, we use a 4 × 6 lattice for the self-organizing maps, since this has worked well previously.

It is not possible to use training data for this problem because our training data is unlabelled, and thus has no discrete clusters from which would could construct our new features. So we do not use training data as such for any of the experiments here. We do however train the autoassociator and self-organizing maps on the data that we are about to cluster, as we've done previously. We hold the number of epochs for this training at 500 for the autoassociator, which we have found to be a good number in the previous reports.

We will vary one parameter for each of the three algorithms we use here. For the autoassociator and single-link clustering algorithm, we're interested in observing the effect of varying the cluster barrier parameter (previously held constant at 0.0025) because in our initial experimentation with our new system we found that this parameter will be the greatest indicator of performance. We will test this parameter between values 0.0 and 0.03, testing at increments of 0.0001.

With self-organizing maps, we'll vary the number of epochs that we train for testing for all values between 0 epochs and 5000 epochs, at 50 epoch intervals.

For the EM algorithm, the number of clusters to form is the parameter that we'll vary. We'll vary this parameter between the values of 5 and 60, testing at increments of 5.

We now present the results of the system using only the Dain and Cunningham features. The performance of the autoassociator is listed in Figure 11, the performance of the self-organizing maps is in Figure 12, and the performance of the EM algorithm is in Figure 13.

As you can see from Figure 11, the results of varying the cluster barrier parameter produce much more predicable results than the results for either of Figure 12 or

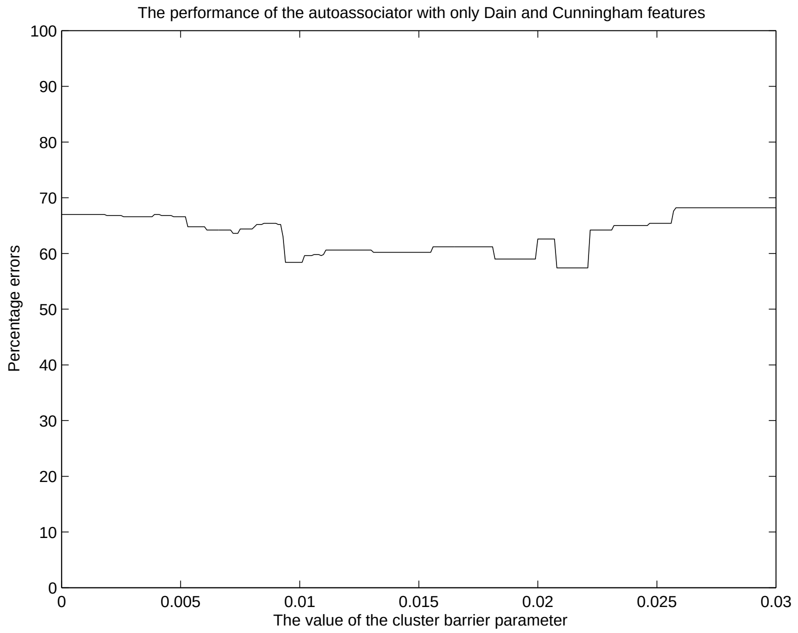

Figure 11: *Autoassociator varying cluster barrier for Dain* et al *features*

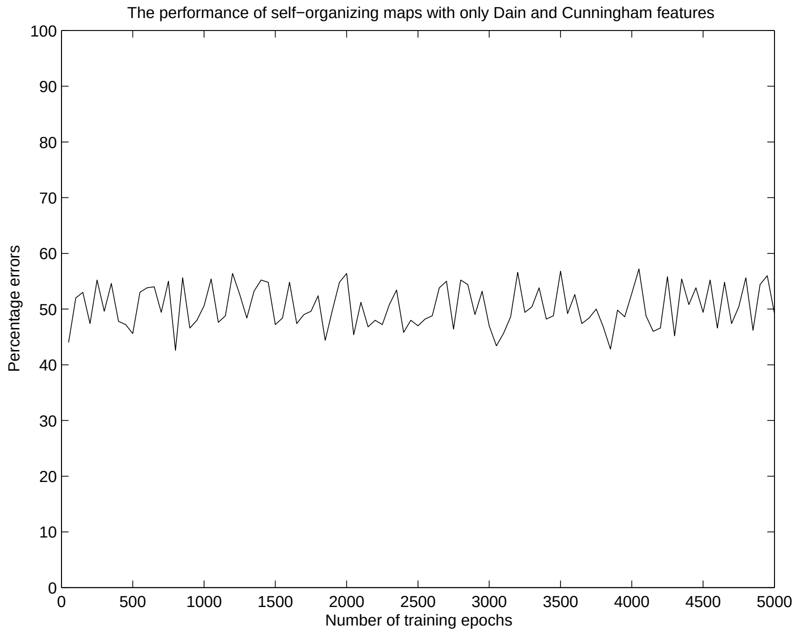

Figure 12: *Self-organizing maps varying epochs for Dain* et al *features*

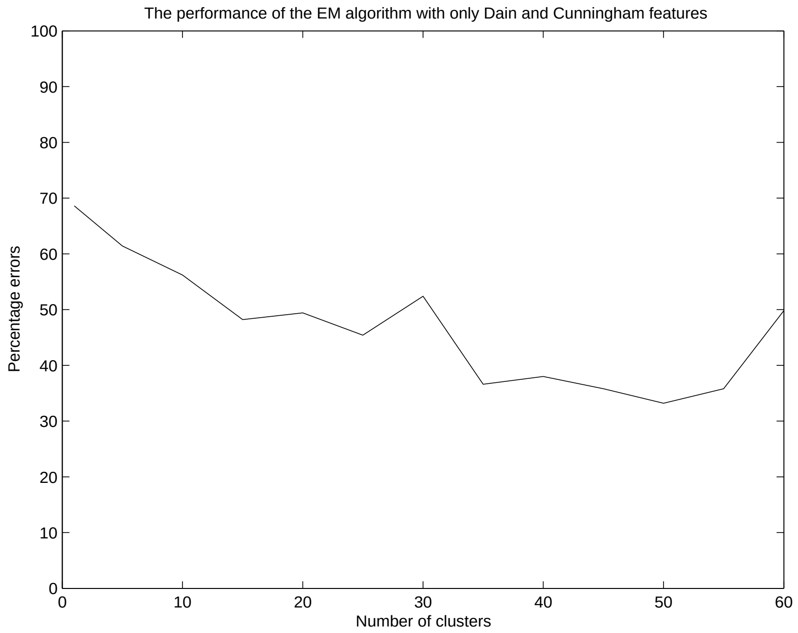

Figure 13: *EM algorithm varying number of clusters for Dain* et al *features*

Figure 13. That said, the best performance attained by the autoassociator as a clustering algorithm under these conditions was 57.4% errors, which is more than the average number of errors for the entire range of self-organizing maps tests. The autoassociator produced noticably worse results than the EM algorithm in this test as well. Self-organizing maps and the EM algorithm had similar performance results for this test — though the EM algorithm attained a slightly lower average number of errors, and a lower best error rate.

We were alarmed by the unpredictability of the results of the self-organizing maps. We expected a much more smooth curve, rather than one that varies so much between nearby neighbours. A curve like that seen in Figure 12 indicates a lack of stability in the results because it indicates a lack of predictability. As a consequence of this, combined with the fact that the EM algorithm attains better overall performance, we can conclude that the EM algorithm seems to have the best performance (under the predefined conditions). The best performance of the EM algorithm is seen with the number of clusters set at 50, but we see interesting results with a number of clusters of 35.

Next we present the results of our system on all of the features from Section 3.3.1. The performance of the autoassociator is listed in Figure 14, the performance of the self-organizing maps is in Figure 15, and the performance of the EM algorithm is in Figure 16.

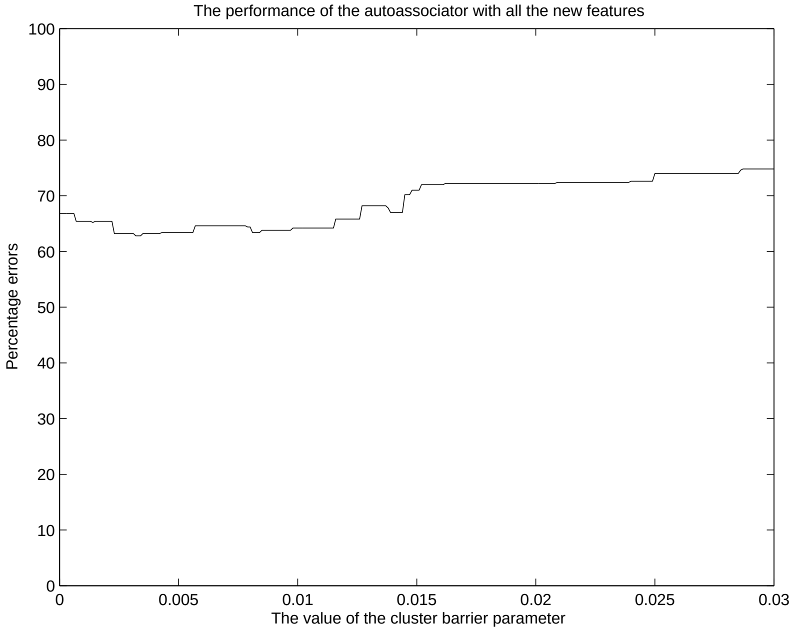

Figure 14: *Autoassociator varying cluster barrier for all features*

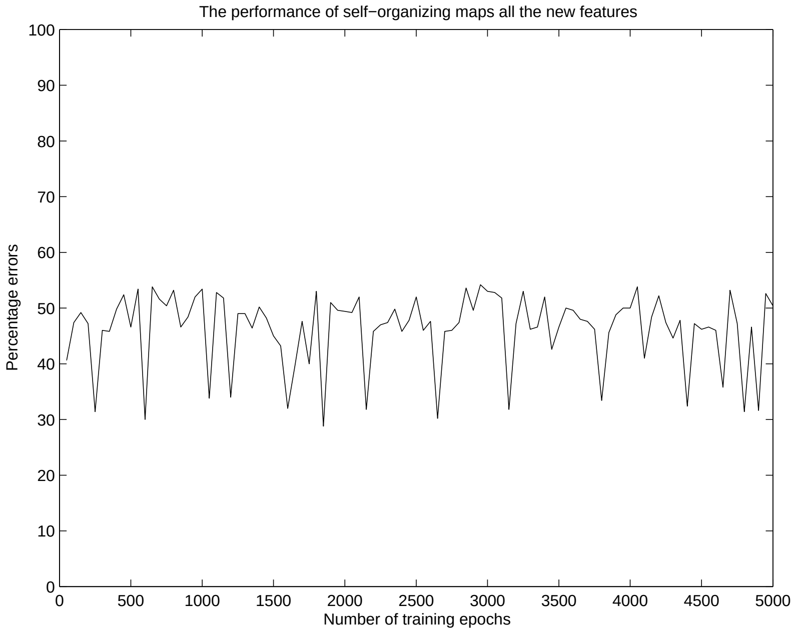

Figure 15: *Self-organizing maps varying epochs for all features*

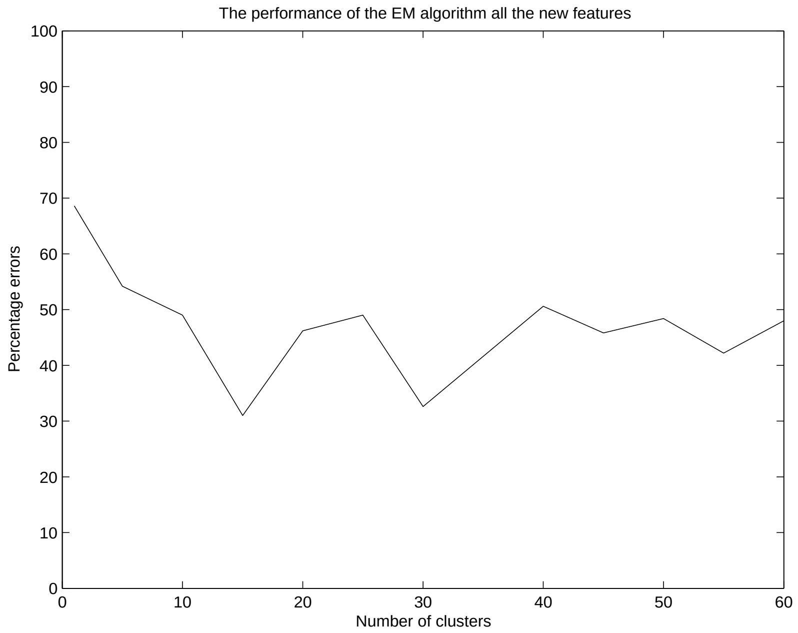

Figure 16: *EM algorithm varying number of clusters for all features*

As you can see from the graphs, the same sort of trends occur that we saw in the previous set of graphs. When comparing the graph for the autoassociator with the Dain and Cunningham features (Figure 11) to the graph for the autoassociator with all the features (Figure 14) we see in the former graph that the autoassociator performs better with larger values for the cluster barrier parameter (best value found: 0.021), whereas for the latter graph, the performance peaks at at smaller values for this parameter (best value found: 0.003).

For the pair of graphs representing the performance of self-organizing maps (Figure 12 and Figure 15) we see that the results were both very volatile. Just as with the Dain and Cunningham feature set, the new features feature set produced results that were unpredictable. One important thing to note between these two graphs, though, is that the latter graph clearly shows that self-organizing maps perform better with all the features than with just the Dain and Cunningham features.

For the pair of graphs representing the performance of the EM algorithm (Figure 13 and Figure 16) we see that the overall shapes of the graphs are very similar, and, indeed, the performance shown in the two graphs is similar as well. The latter graph shows that the best performance happens at 15 clusters and 30 clusters. The most interesting results are seen at the 30 cluster mark. The results at this point seem to balance clustering errors versus separation errors well.

At this point we'd like to delve a bit deeper into the results to perform more thorough analysis. Some of the best results we've seen are using the EM algorithm with all new features selected and with the number of clusters fixed at 30. With the number of clusters set to 30, the system produced 58 clustering errors and 105 separation errors, for a total of 163 errors, which is a percentage error of 32.6%.

A significant portion of the errors occur because of the problems of two super-clusters. One super-cluster is composed of two separate super-clusters from the gold standard. The two super-clusters from the gold standard are homogenously composed of `nmap` alerts. These two optimal super-clusters are pushed together by the EM algorithm, and this accounts for 32 of the clustering errors. 43 separation errors occur because a large super-cluster from the gold standard (composed of "BAD-TRAFFIC tcp port 0 traffic" alerts) is split into two large clusters. Another 21 separation errors occur because another super-cluster of the same type of alerts is split from its optimal super-cluster.

From this analysis we can conclude that many of the errors that we've seen in the results are not critical nor dangerous. We have seen that some of the errors from the level of clustering have propagated upwards and cause clustering errors in the larger super-clusters where there would otherwise be no errors as well.

To conclude we find that the EM algorithm produces the most interesting and stable results. We found the surprising result that the autoassociator is ill-suited to the second correlation step, and we found that self-organizing maps produce unpredictable results.

4.2.6 Autoassociator Reconstruction Error Stability

We hope to show that the reconstruction error values produced by the autoassociator change less if we train the autoassociator with a separate 10,000-item training set. Minimizing change in the reconstruction error values is important because it allows the IDS operator to find correlation between different data windows more easily. The second correlation step (Section 3.3) requires this trait of stability as well.

We wish to show stability in reconstruction error values because if reconstruction error values for a set of similar alerts are static, or vary little, then alerts are comparable between test sets, which bolsters our super-cluster creation system introduced in the previous report. This is also a property of the autoassociator we've reported to want since creating the Autocorrel I system [1], but we've never proven that using an unlabelled training dataset produces this desired stability.

So we set up an experiment to test the stability of a set alerts. To do this we use the parameter values that we've found to work well. Namely, we train the autoassociator for 500 epochs and with 64 hidden units. We use the 10,000 unlabelled alert training set. We contrast this with the reconstruction error values of a system that doesn't use the training data, a system that both trains on and clusters the evaluation data.

For this experiment we choose a set of 100 alerts for which we'll report the reconstruction error values. This set of 100 alerts contains 50 `Scan Proxy Port 8080` alerts and 50 `Short UDP Packet` alerts. For this set of 100 alerts we report on the reconstruction error values seen if clustered together with a greater set of 400 other alerts. For one test this set of 400 alerts will be more `Scan Proxy Port 8080` alerts and for another test the additional set of 400 alerts will be composed of different `Short UDP Packet` alerts.

We expect the reconstruction error values of the 100 evaluation alerts will exhibit less stability if trained with the complete set of alerts they're clustered with than if the training set of 10,000 alerts is used. By changing the distribution of the evaluation set we hope to show that using the evaluation set as the training data leads to unstable reconstruction error values.

To explore the impact on reconstruction error values we use our system to determine the reconstruction errors on the two datasets. As mentioned both evaluation datasets have 500 alerts, but one dataset has 450 `Scan Proxy Port 8080` alerts with only 50 `Short UDP Packet` alerts, and the other dataset of 500 alerts has 450 of the latter type of alert and only 50 of the former. We report on the reconstruction error values for 100 of the 500 alerts from each dataset. The 100 alerts that we report with are the same in both datasets. See Figure 17 for a graph of the results.

To measure the stability of the reconstruction error values we determined the reconstruction error values for both of the datasets for each different training set, then took the difference of the reconstruction errors for the data items that were in both datasets. The results of this experiment for the autoassociator trained with the 10,000 training set alerts are the solid line in Figure 17. The results for the autoassociator trained on the evaluation set are the dotted line in the same figure.

As you can see from the graph the difference in reconstruction error values for the autoassociator trained on the evaluation set are much larger than those for the autoassociator with the 10,000 alert training set. This is especially true of the `Short UDP Packet` alerts which are represented by the high plateau on the right side of the graph. You can see for the `Scan Proxy Port 8080` alerts that the autoassociator trained with the 10,000 alert training set displays more stable reconstruction error values too. The differences for these alerts are not as dramatic, but it is still clear that the autoassociator trained with 10,000 alerts is uniformly more stable than if trained on the evaluation data.

From the previous experiment we can conclude that the reconstruction error values

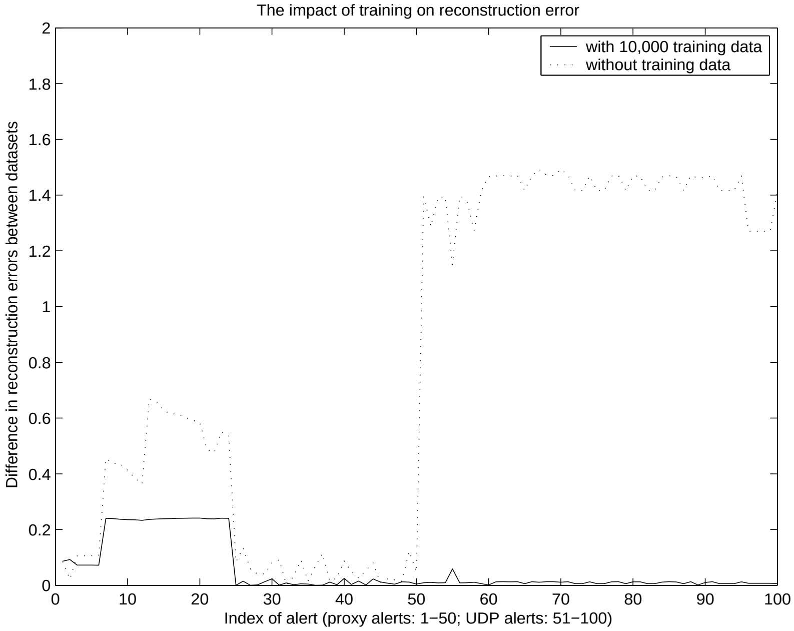

Figure 17: *The impact of using a training set on the stability of reconstruction error values*

produced by an autoassociator trained on the 10,000 alert training set are much more stable than if the autoassociator is not trained on this data and instead trained on the evaluation data as we've previously experimented with in Section 4.2.2.

4.3 Overall Performance Results

In this section we want to give results for the performance of our overall system. To do this fairly we felt it was necessary to compare our system to the results of another system. As we've mentioned, though, there isn't a comparable alert correlation system based on unsupervised machine learning to which we could compare our system. The systems of Dain and Cunningham [27] and Julisch *et al* [26] use labelled training data that we don't have access to.

To solve this problem we create our own simple rule-based correlation system that we can use as a benchmark for our performance. We can then compare the performance of our system to the performance of the simple system to see how well our system does.

Our simple correlation system is based on ideas presented in Network Intrusion Detection: An Analyst's Handbook by Stephen Northcutt [3], who is also the author of the Shadow IDS [6]. His ideas for alert correlation are also present the free software program ACID [9].

The system Northcutt advocates, and the system we create, is based on correlation using three primary variables: type of alert to be correlated, time that the alert was generated, and source IP address of the alert. This information alone allows us to form clusters similar to those of our golden standard of Section 4.1.1.

We form clusters by first splitting the test dataset into 100-alert time windows. We then sort the alerts into groups such that all of the alerts of a certain type with a given source IP address form a single cluster. This type of correlation is naïve and simple, but as you'll see it does reasonably well. More importantly it adequately forms a base line to which we can compare our alert correlation system.

After creating the simple correlation system, we tested it against the same dataset used to create our gold standard. We found that there were 292 errors for the evaluation dataset of 500 using this simple correlation scheme. That means the accuracy of the simple correlation system is 41.6%. From Section 4.2.5 we saw that our system had 163 errors for the gold standard data, which is an accuracy percentage of 67.4%.

Something to note, though, with the simple correlation algorithm is that it only produced separation errors and did not produce any clustering errors. This is significant because if it is true in general, the simple correlation step could be used to reduce the number of alerts before any are seen by our system. This kind of performance modification is beyond the scope of our work, but it would be interesting to consider this type of hybrid machine learning system if the system was to be deployed.

Clearly our system performs better than the simple base line that we've created. It could be argued that the simple correlation system that we've created is too simple to compare to our complex, optimized system. But to that we note that many of the correlation methods used by intrusion detection systems are as simple as the one we presented here. Therefore we believe the comparison is fair.

4.4 Lessons Learned and Future Work

The first valid criticism of the work we've done is that we haven't tested our system with enough data to provide statistically solid results. Because of the nature of the data we work with it would have taken quite a bit of time to create more testing or evaluation data, so we decided to focus on other aspects of the research. We felt that most of the work we've done for this contract is exploratory, so we needed to react quickly to bad ideas and poor performance to guide the system in the direction of what we wanted from it. Because the field of alert correlation is relatively young, there isn't significant precedent in the field to guide research in a clear way.

The next criticism is related to the first: the type of data we tested wasn't varied enough. We wanted to test our system with multiple datasets, but we were only able to test the full system on the incidents.org data. This leads to a methodological problem in our research; we have used the incidents.org dataset as the evaluation dataset as well as the testing dataset. We've made many architectural choices for our system based on the performance of the system on the incidents.org dataset, then we've tested the system's overall performance on only that dataset. To really show that our system isn't just optimized for the incidents.org dataset (or datasets with that same innate structure) we need to test our system on other datasets.

The final error we made is that we haven't sufficiently corrected the 40-to-1 reconstruction error formula mapping problem. We've determined that the errors caused by the formula lead to significant clustering errors in the first step of correlation, but we also tuned the parameters for the autoassociator to minimize the effect of this. We found that the system performs better as-is than if another clustering algorithm are substituted for the cluster barrier algorithm part of our system. We know this is an error in our system that degrades performance, but we decided to focus on the second correlation step for the latter part of this project.

The most important future work for us is to test our system on more data and more datasets. We think that this is important because it should show the real performance of our system. We expect that it won't be significantly different from the current performance, but it is important that we show this using a different dataset.

Another future work task might be to correct the 40-to-1 mapping problem, if we're able to do so in a way that increases performance. We looked into graph theory-based methods for splitting apart clusters that are known to have clustering errors. We constructed a graph based on the distance matrices seen in Section 4.2.1 where each node of the graph is an alert in the cluster and each edge of the graph is placed between nodes with sufficiently short Euclidean distances in the distance matrix. A cluster was broken apart if the graph we produced was disconnected. We tested this method but nothing came of it because it produced more errors in the overall system than it fixed. But we still think that this line of research could be fruitful and we'd like to explore it further in the future.

A final future work task might be to use non-machine learning methods to try to bolster performance in a closer-to-operational setting. The idea of combining our system with more simple correlation methods such as that in Section 4.3 could be a powerful one. We think that rule-based systems could add a lot to our performance, but creating a hybrid system was beyond our scope here.

5 Conclusions

Our system of finding correlation between IDS alerts has evolved significantly from our previous report for DRDC. Our Autocorrel I system now looks more similar to the first step of correlation in our Autocorrel II system. We've attained better performance than we did in our previous system, and we are now more confident in our performance too, having examined more aspects of the system. We've also added more to our correlation effort by correlating different steps of an attack as well as the simpler task of threading alerts. We are happy with our performance, but we also feel that there is room for improvement.

As we mentioned, the first correlation step of our new Autocorrel II system is directly based on our work for the Autocorrel I system. The Autocorrel I system is effective at correlating some types of alerts. Since this first project we've been able to characterize this type of correlation as threading. *Threading* clusters alerts from a set such that each cluster corresponds to a part of an attack. This process is useful as a first step in alert correlation because it reduces the number of alerts that must be considered for higher-level correlation and because it highlights important relationships between alerts. Autocorrel I is competent at threading, but we were interested in achieving better threading performance. So for this second project we learned more about the Autocorrel I system by experimenting with the system and by proving aspects of the system that we'd previously only conjectured. We improved the performance of the Autocorrel I system by fine-tuning the autoassociator and other aspects of the system such as the data scaling algorithm and the single-link clustering algorithm.

Many of the experiments we ran for this project were designed to examine aspects of the Autocorrel I system. For instance, we showed that by training the autoassociator neural network for this first correlation task we attain a system that continues to perform well even as the distribution of types of alerts gathered from the network radically changes. We also proved that an aspect peripheral to the autoassociator neural network, the reconstruction error formula, is responsible for a significant portion of the errors caused by Autocorrel I. We then experimented with autoassociator parameters and other tunable algorithms to mitigate the effect of this problematic formula.

Aside from tuning the Autocorrel I system for better performance and proving the reliability of the system through further experimentation, we decided that the threading style of correlation didn't help us find all of the correlations in the data for which we'd hoped. While threading can be used to find the alerts for a single step of an attack (such as a scan of a network with `nmap`), we were interested in finding higher level correlations as well (such as a buffer overflow attack on a service that was just scanned by `nmap`). These higher level correlations make the job of the intrusion analyst easier by giving him more information, but they do not add to the performance of the threading component per se. They add a different type of information that is not quantifiable in terms of threading.

The second correlation step that we introduced for Autocorrel II was our attempt at discovering this higher-level correlation information. We created a set of features to represent the clusters from the threading step so that the different threads could easily be compared and grouped together based on similarity in thread features. Our attempt at this was fairly successful, as we demonstrated by showing that 67.4% of alerts were clustered in their correct group in a set of 500 alerts. We compared this to a simple rule-based correlation system of our creation and found that the rule-based system attained only 41.6% success. This result indicates that our unsupervised machine learning system was significantly more successful that our simple rule-based system at finding the higher-level correlation that we're looking for.

Our success in this comparison, combined with our bolstered knowledge of the threading part of our Autocorrel II, has shown the viability of an unsupervised machine learning-based alert correlation system.

References

[1] Japkowicz, Nathalie and Smith, Reuben (2005). Autocorrel I: A Neural Network Based Network Event Correlation Approach. (CR 2005-030). DRDC-Ottawa. Scientific Authority: Maxwell DOndo.

[2] Stevens, W. Richard (1994). TCP/IP Illustrated : The Protocols, Vol. 1. Addison-Wesley.

[3] Northcutt, Stephen (1999). Network Intrusion Detection: An Analyst's Handbook, Indianapolis, IN: New Riders Publishing.

[4] Bejtlich, Richard (2000). Interpreting Network Traffic: A Network Intrusion Detector's Look at Suspicious Events. *first.org conference*.

[5] The TCPDump Program (Online). tcpdump.org. http://www.tcpdump.org/ (Jan. 20, 2004).

[6] SHADOW: Second Heuristic Analysis for Defensive Online Warfare (Online). NSWC. http://www.nswc.navy.mil/ISSEC/CID/ (Sept. 12, 2004).

[7] Peter G. Neumann, Phillip A. Porras (1999). Experience with EMERALD to Date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, California: SRI International Computer Science Lab.

[8] Roesch, Martin (1999). Snort—Lightweight Intrusion Detection for Networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, pp. 229–238. Seattle, Washington: The USENIX Association.

[9] Danyliw, Roman. ACID: Analysis Console for Intrusion Detections (Online). AIRCERT. http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html (Sept. 12, 2004).

[10] Vigna, Giovanni and Kemmerer, Richard A. (2001). NetSTAT: A Network-based Intrusion Detection System. *Reliable Software Group, Department of Computer Science, U.C. Santa Barbara.*

[11] QuIDScor (Online). Qualys Inc.. http://quidscor.sourceforge.net/ (Sept. 12, 2004).

[12] Intellitactics NSM: Network Security Manager (Online). Intellitactics. http://www.intellitactics.com/products/nsm_overview.html (Sept. 12, 2004).

[13] NFR: Network Flight Recorder (Online). NFR Security. http://www.nfr.com/ (Sept. 12, 2004).

[14] Fan, Wei, Lee, Wenke, Stolfo, Salvatore J., and Miller, Matthew (2000). A Multiple Model Cost-Sensitive Approach for Intrusion Detection. *Department of Computer Science, Columbia University.*

[15] Lee, Wenke and Stolfo, Salvatore J. (2000). A Framework for Constructing Features and Models for Intrusion Detection Systems. In *ACM Transactions on Information and System Security, Vol. 3, No. 4*, pp. 227–261. ACM.

[16] Cohen, William (1995). Fast Effective Rule Induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123.

[17] Lippmann, Robert, Haines, Joshua W., Fried, David J., Korba, Jonathan, and Das, Kumar (2000). The 1999 DARPA Off-Line Intrusion Detection Evaluation. *Computer Networks*, **34**(4), 579–595.

[18] Georges, Jim and Milley, Anne H. (1999). KDD'99 Competition: Knowledge Discovery Contest. *SAS Institute Inc.*

[19] Eskin, Eleazar, Arnold, Andrew, Prerau, Michael, Portnoy, Leonid, and Stolfo, Sal (2001). A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data. *Department of Computer Science, Columbia University.*

[20] Fawcett, Tom (2003). ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. (Technical Report HPL-2003-4). HP Laboratories. Palo Alto, CA.

[21] Antti Hätälä, Camillo Särs, Addams-Moring, Ronja, and Virtanen, Teemupekka (2004). Event Data Exchange and Intrusion Alert Correlation in Heterogeneous Networks. In *Proceedings of the 8th Colloquium for Information Systems Security Education (CISSE)*, pp. 84–92. Westpoint, NY: CISSE.

[22] Debar, H., Curry, D., and Feinstein, B. (2004). The Intrusion Detection Message Exchange Format. *IETF Working Group.*

[23] Ning, Peng and Cui, Yun (2002). An Intrusion Alert Correlator Based on Prerequisites of Intrusions. *Department of Computer Science, North Carolina State University.*

[24] Debar, Hervé and Wespi, Andreas (2001). Aggregation and Correlation of Intrusion-Detection Alerts. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pp. 85–103. Springer-Verlag.

[25] Valdes, Alfonso and Skinner, Keith (2001). Probabilistic Alert Correlation. In *RAID 2001, LNCS 2212*, pp. 54–68. Springer-Verlag.

[26] Julisch, Klaus and Dacier, Marc (2002). Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of SIGKDD '02, the 8th International Conference on Knowledge Discovery and Data Mining*, pp. 366–375. Edmonton, Alberta, Canada: ACM.

[27] Dain, Oliver and Cunningham, Robert K. (2001). Fusing a Heterogeneous Alert Stream into Scenarios. In *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, pp. 1–13. Philadelphia, PA.

[28] The DEFCON Data Set (Online). defcon.org. http://www.defcon.org/ (31 Mar 2005).

[29] Japkowicz, Nathalie and Smith, Reuben (2005). Autocorrel I: A Neural Network Based Network Event Correlation Approach. (Contractor Report CR 2005-030). DRDC Ottawa. Ottawa, ON.

[30] Lippman, R.P. (1987). An Introduction to Computing with Neural Nets. In *IEEE ASSP Magazine*, pp. 4–22.

[31] Demuth, H. and Beale, M. (2001). MATLAB: Neural Network Toolbox, User's Guide v. 4.0, Nantick, MA: The MathWork, Inc.

[32] Zurada, J.M. (1992). Introduction to Artificial Neural Systems, New York, NY: West Publishing Company.

[33] Widrow, B. and Lehr, M.A. (1990). 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. In *IEEE Proceedings*, Vol. 78, pp. 1415–1442.

[34] Hagan, Martin T., Demuth, Howard B., and Beale, Mark (1996). Neural Network Design, PWS Publishing Company.

[35] Japkowicz, Nathalie (2001). Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks. *Mach. Learn.*, **42**(1-2), 97–122.

[36] Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977). Maximum Likelihood from Incoming Data via the EM Algorithm. *J. Royal Stat. Soc., Series B*, **39**(1), 1–36.

[37] Witten, Ian H. and Frank, Eibe (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, The Morgan Kaufmann Series in Data Management Systems. San Francisco, CA: Morgan Kaufmann Publishers.

[38] Kohonen, Teuvo (1995). Self-Organizing Maps, Vol. 30 of *Springer Series in Information Sciences*. Berlin, Heidelberg: Springer-Verlag. (Second Extended Edition 1997).

[39] Tang, Bin, Heywood, Malcolm I., and Shepherd, Michael (2002). Input Partitioning to Mixture of Experts. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, pp. 227–232. Honolulu, HI.

[40] Chiu, S.L. (1994). Fuzzy Model Identification Based on Cluster Estimation. *Journal of Intelligent and Fuzzy Systems*, **2**(3), 267–278.

[41] Vesanto, Juha and Alhoniemi, Esa (2000). Clustering of the Self-Organizing Map. *IEEE Transactions on Neural Networks*, **11**(3), 586–600.

[42] Kohonen, Teuvo, Hynninen, Jussi, Kangas, Jari, and Laaksonen, Jorma (1996). SOM_PAK: The Self-Organizing Map Program Package. (Report A31). Helsinki University of Technology, Laboratory of Computer and Information Science.

[43] The incidents.org Data Set (Online). incidents.org. http://www.incidents.org/logs/ (Mar. 14, 2004).

[44] Wall, Larry. The Perl programming language (Online). www.perl.com. http://www.perl.com/ (Sept. 1, 2004).

[45] Duda, Richard O., Hart, Peter E., and Stork, David G. (2001). Pattern Classification, New York, NY: John Wiley and Sons. (Second Edition).

[46] Hsu, Chih-Wei, Chang, Chih-Chung, and Lin, Chih-Jen (2003). A Practical Guide to Support Vector Classification. *National Taiwan University*.

[47] Jain, A.K., Murty, M.N., and Flynn, P.J. (1999). Data Clustering: A Review. *ACM Comput. Surv.*, **31**(3), 264–323.

[48] Manning, Christopher D. and Schütze, Hinrich (1999). Foundations of Statistical Natural Language Processing, Cambridge, MA: The MIT Press.

[49] McHugh, J. (2000). The 1998 Lincoln Laboratory IDS Evaluation – A Critique. In *Proceedings of the 3rd Workshop on Recent Advances in Intrusion Detection (RAID)*, pp. 145–161. Toulouse, France: Springer-Verlag.

[50] The comp.ai.neural-nets FAQ (Online). comp.ai.neural-nets. http://www.faqs.org/faqs/ai-faq/neural-nets/part2/ (Sept. 1, 2004).

[51] Yang, Yiming and Liu, Xin (1999). A Re-examination of Text Categorization Methods. In *Proceedings of SIGIR-99, 22nd Annual International Conference on Research and Development in Information Retrieval*, pp. 42–49. Berkeley, CA.

[52] Cuppens, F. and Miege, A. (2002). Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 187–200. Oakland, CA: IEEE Computer Society.

This page intentionally left blank.

Annex A: Acronyms and Abbreviations

ACL	Access Control List
ANN	Artificial Neural Network
CGI	Common Gateway Interface
DARPA	Defense Advanced Research Projects Agency
DEFCON	Defence Conference
DDOS	Distributed Denial of Service
DOS	Denial of Service
DNS	Domain Name System
DRDOS	Distributed Reflected Denial of Service
DREnet	Defence Research Establishment Network
ECN	Explicit Congestion Control
FQDN	Fully Qualified Domain Name
IDS	Intrusion Detection System
IDWG	Intrusion Detection Working Group
ISN	The Initial Sequence Number
IP	Internet Protocol
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
HTRQ	Hypertext Request
MSE	Mean Square Error
OS	Operating System
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
WWW	World Wide Web

This page intentionally left blank.

Annex B: Project Implementation Plan

B.1 Problem Overview

In recent years *network event correlation* has become an important topic within the research area of *intrusion detection systems*. Network event correlation research is important because the volume of alerts produced by intrusion detection systems, even if correctly tuned, can easily overwhelm intrusion detection system analysts. In fact, there exist tools created by attackers designed to flood intrusion detection systems with alerts so that the analysts can't do their job and attacks go undetected [24]. Research in the area of network event correlation works to ameliorate the chances of detecting false alarms and understanding real alerts in intrusion detection systems by bringing more meaning to individual alerts. To do this is to bolster the intrusion detection system analysts in their important work of detecting attacks and reconnaissance probes against the network.

Network event correlation, also called sensor alert correlation or simply alert correlation, is the task of identifying relationships between other data, such as other alerts or known-vulnerability information. Three types of network event correlation are known to us. The first type of alert correlation attempts to correlate alerts from a single stream of alerts originating from one network sensor. (Note that Dain *et al* [27] call this type of alert correlation *threading*.) This is the task we'll focus on, and have focused on, in our research. This type of alert correlation tries to determine the context of an individual alert by analyzing its relationship with other alerts produced by the same network sensor. Often an attack or reconnaissance probe is represented by multiple alerts produced by the sensor, so by determining the relationship with other alerts from the same attack, the intrusion detection analyst may analyze an attack with more relevant information. As well, the analyst's time isn't spent determining the relationship between different sensor alarms, so he is more efficient.

The second type of alert correlation attempts to determine the relationship between alerts from distinct network sensors. Security-diligent organizations deploy multiple intrusion detection systems within the network, and often have other devices and software packages that produce security alerts, so determining which alerts relate to each other, or which are duplicate alerts, is a formidable task. Because of the lack of datasets representing network events with different network sensors, we cannot research this task. We conjecture, though, that any system which solves the first type of correlation will also help solve the second and vice versa, because the problems are so similar in nature.

The final type of alert correlation that we'll talk about tries to correlate network events to published vulnerabilities [11]. By correlating network events to a known vulnerability, you get better information about the meaning of the alert and how the attacker might be trying to break into your network. This type of correlation is naturally quite different to the first two types of correlation and we don't consider it any further.

In our experiments, we use the freely-available intrusion detection system *Snort*. Snort is a lightweight, rule-based intrusion detection system [8]. We explore alert correlation with Snort because it has no correlation system of its own and because it is freely available. In our alert correlation system, we take the alerts generated from Snort and find correlations within the set of alerts produced by that single sensor. We read the attributes directly from the Snort output.

The data we feed to Snort is the data from the DARPA 1999 intrusion detection dataset [17]. The DARPA 1999 intrusion detection dataset is a dataset with labelled attacks. It is an artificial dataset created for testing intrusion detection systems. The dataset consists of packets captured from the artificial network and lists of system calls from hosts on the network. We only use the captured packets in our testing because we focus on network intrusion detection in this research project. Unfortunately, Snort produces many false alarms using the default rules on this dataset. Our system ("AutoCorrel I") is meant to correlate false alarms as well as true alarms, but because only real attacks are labelled in the dataset, we can't use the fact that the dataset contains labels to test the performance of our system in general. Using the dataset labels, we can only test the performance of our system correlating the real attacks. We must take this into account when choosing any system of performance evaluation.

Our system currently uses only the *autoassociator* [35], a neural network architecture specialized to the task of recognition, to cluster alerts. Each cluster of alerts produced by our system represents a group correlated from a set of the Snort output. For this project, we plan to experiment with other clustering algorithms such as *self-organizing maps* [38] and the *expectation-maximization* algorithm [36].

For this project proposal, we discuss our goals in this project and give a timeline for those goals.

B.2 Solving the 40-to-1 Mapping Problem

In the system we presented as our final report for our previous contract with DRDC ("Neural Network Application to Network Event Correlation"), we speculated that a significant source of error in our methods is caused by our use the *reconstruction error* formula. This formula computed a single real value from the 40 autoassociator outputs, causing a 40-to-1 mapping. The use of a 40-to-1 mapping formula was necessary because it allowed us to form discrete clusters from the otherwise non-interpretable 40-dimensional output of the autoassociator easily. We chose to use the reconstruction error formula as the formula for performing the 40-to-1 mapping because it was used in Japkowicz [35], where the formula was necessary to generate ROC curves [20] in a binary classification problem.

To conclusively prove that the 40-to-1 is a significant source of errors, we propose an experiment. We propose to cluster 100 data items using our previously presented system, and then, for each data item in each cluster formed, we want to list the data item with its 40 autoassociator outputs, before the reconstruction error for the data item has been computed. We believe that it will be clear which data items within the cluster are similar and which are anomalous, and from this the extent of the errors caused by the 40-to-1 mapping problem will be evident.

We hope to solve some of the clustering performance problems caused by the reconstruction error formula and the simple, ad hoc clustering method by using a different clustering method. We'd like to experiment with self-organizing maps, an unsupervised neural-network-based clustering algorithm created by Kohonen [38], and the expectation-maximization algorithm, a well-known clustering algorithm described by Dempster *et al* [36], as possible candidates for clustering algorithms.

We will still use the autoassociator for recognition of trained alerts as a pre-clustering algorithm step. We believe that the autoassociator increases the performance of clustering on this type of data. (In particular, we believe that the customizability of the autoassociator is a powerful asset to the algorithm.) We will investigate whether this is actually true by the experimenting of clustering the data both with and without the autoassociator.

We will also investigate the impact on the clustering performance of training the autoassociator with training data. It is possible to use the autoassociator without training it, or by training it with different amounts of unlabelled data for a different number of epochs. We will investigate, without determining optimality of the neural network, the effects of these variables on the clustering performance with the autoassociator involved. For these experiments, we will include graphs of the *mean-squared error* recorded during training in the interim reports.

B.3 Feature Selection

Feature construction and selection are important parts of any machine learning exercise [47]. In the previous report we completed for DRDC, feature selection was heuristic and non-rigourous. We gave deductive justifications of the effects of features such as source and destination IP addresses and TCP sequence numbers. For this project we will do rigourous analysis of these features and others using unsupervised methods such as *principal component analysis (PCA)*. This type of rigourous analysis will give justification to the features we select for our final system.

For features that are not aiding the performance of the machine learning algorithms, we may investigate ways to help the features add to performance. For instance, if we find that source and destination IP addresses only hinder clustering performance as they're currently encoded, we will try encoding the IP address information in a different way.

Because we will be using the autoassociator to recognize training input then a clustering algorithm as a second step, we may want to use different sets of features at the two steps. (That is, feed a different set of features to the autoassociator than to the clusterer.) To illustrate why we might want to use two feature selection steps, envision the plausible scenario where we find that training the autoassociator with IP addresses produces worse results for clustering (since the IP addresses in the training data are bound to be different than those in the testing data) but clustering with IP addresses produces better results. In this scenario, having two levels of feature select — one level for the autoassociator and one level for the clustering algorithm — would be useful. We plan to experiment with this.

As well, we investigate the linear attribute scaling methods that we used previously. In the last project we scaled all training data to the interval $[0, 1]$ and applied the automatically-determined scaling constants to the testing data. (This method is common for neural networks [50].) We'd like see how the performance of this scaling affects the clustering results. We'll compare the results of this scaling to the application of a mean/standard deviation scaling method [50], the application of a linear map to the interval $[-1, 1]$ instead, and the results of a neural network without scaled inputs.

B.4 Evaluation Analysis

Clustering accuracy is a very simple measure that might not be well suited to our problem. Accuracy, as a percentage measure, doesn't always reveal the true performance of machine learning algorithms. For instance, accuracy doesn't say anything about the types of errors that are occurring. For this reason, in our final report for the last contract we decomposed the errors into two types: *clustering error* and *separation error*. We defined clustering errors to be alerts that were incorrectly clustered with dissimilar alerts, and we defined separation errors to be alerts separated from their optimal cluster but not grouped with any dissimilar alerts. Although these types of errors are intuitive, this evaluation method is not supported by literature, so we propose investigating other evaluation methods.

Precision and *recall* are evaluation measures commonly used in the research areas of *information retrieval* and *text categorization*. Yang [51] defined precision and recall:

> Recall is defined to be the ratio of correct [label] assignments by the

system divided by the total number of correct assignments. Precision is the ratio of correct assignments by the system divided by the total number of the system's assignments.

Precision and recall could be calculated for each labelled evaluation cluster in the test dataset. The multiple precision and recall values could be combined into one using micro-averaging or macro-averaging [51]. Precision and recall can be graphed against each other to produce ROC graphs [20], which are useful in determining how to trade off these measures against each other.

Another attribute of the formed clusters we noticed in our previous system was that, if separation of an optimal cluster has occurred, the separate pieces of the optimal cluster are often positioned beside each other, because of the nature of the autoassociator. It might be worth investigating the weighting of errors incorporating some sort of distance measure since clusters separated by a number of unrelated clusters are more dangerous than cluster separated but packed together. We will have to analyze the merits of such a measure, if it has relevance in our completed system.

Part of our evaluation should be the comparison of our system to another system. Since we determined in the last project that there are very few functional alert correlation systems with which we could compare ours, we will try to justify our system by devising and running experiments that would help us show our superiority over rule-based systems. Because this task may not be feasible given the time constraints of our research, we include it in this proposal only as a possibility.

B.5 Project Timeline

1. **January 3-February 4**: Analysis of the true effect of the 40-to-1 mapping problem; initial experimentation with combining the autoassociator and self-organizing maps or the expectation-maximization algorithm.

2. **February 5-February 28**: Detailed analysis of combining autoassociator with self-organizing maps or expectation-maximization algorithm; analysis of autoassociator's parametric variables (with respect to correlation problem); analysis of the effect of training the autoassociator.

3. **March 1-March 31**: Application of rigourous feature selection methods.

4. **April 1-April 30**: Experimentation with two-level feature selection; experimentation with other data scaling methods; possible reconstruction of underperforming features.

5. **May 1-May 31**: Analysis of evaluation measures; writing of final report.

This page intentionally left blank.

Annex C: First Progress Report

C.1 Introduction

In the *Autocorrel I* system we presented in our final report for our last research contract with DRDC ("Autocorrel I: A Neural Network Based Network Event Correlation Approach"), we conjectured that a significant source of error in our system is caused by the use of the *reconstruction error* formula. In the proposal of this current contract, we proposed to show conclusively, through experimentation, that this is true. We do that in this report, showing that the formula is the most significant source of errors in our present system. The errors caused by the formula are the most damaging sort of errors, as well.

The reconstruction error formula causes a 40-to-1 mapping of neural network outputs to a single real value. It was necessary to use this mapping because analysis of 40-dimensional vectors is difficult. Using this mapping simplified our problem sufficiently, allowing us to form discrete clusters. But we are now rethinking this simplification because it is clear the use of this formula is causing many of the errors we hope to avoid. In Section C.2 we set up our experiments and discuss how the results of them show that the formula is a significant source of errors in our system.

In this report we also begin to discuss a solution to the problem of using the reconstruction error formula as the clustering step in the correlation process. We propose the use of a different machine learning clustering algorithm, such as *self-organizing maps* (SOM) by Kohonen [38] or the *EM algorithm* in Dempster *et al* [36]. We show some preliminary results with the EM algorithm in Section C.3 and with SOMs in Section C.4. (We also did some further experimentation with the k-means algorithm for this report, but the output of our experiments was inferior to the output of the EM algorithm and the SOMs in every way, so we decided not to consider any further this algorithm as a candidate for a clustering algorithm. We do not discuss k-means any further.)

We conclude this report in Section C.5 with a summary of the mapping experiment and the tests with the new clustering algorithms, the EM algorithm and the SOM.

C.2 The 40-to-1 Mapping Problem

The reconstruction error formula (Equation C.1) computes a single real value from the 40 autoassociator outputs, causing a 40-to-1 mapping. The use of a 40-to-1 mapping formula was necessary because it allowed us to form discrete clusters from the otherwise non-interpretable 40-dimensional output of the autoassociator. We chose to use the reconstruction error formula as the formula for performing the 40-

to-1 mapping because it was used in Japkowicz [35], where the formula was used to generate ROC graphs (Fawcett [20]) in a binary classification problem.

$$E_x = \sum_{i=1}^{n} [x_i - f_i(x)]^2 \tag{C.1}$$

In Formula C.1 x_i is the value of the i^{th} attribute of data item x and $f_i(x)$ is the value of the i^{th} output of the neural network simulated with input x.

To conclusively prove that the 40-to-1 mapping is a significant source of errors, we devised and ran an experiment. To perform the experiment, we first hand-clustered 100 data items to obtain an optimal clustering against which we could compare any system-produced clustering of the same data items. Next, we clustered those 100 data items using the Autocorrel I system, and then analyzed the errors produced by the system.

We are most concerned with the errors where intuitively dissimilar alerts are grouped together. We call these types of errors *clustering errors* because they indicate the problem of finding correlation where there is none. If two or more unrelated alerts are clustered together in a production system, it could have damaging consequences for an organization relying on the alert correlation system. Any alert correlation system should only make the intrusion detection analyst's job easier, but clustering errors could make the analyst's job harder by hiding important alerts. More dangerously, an attacker could possibly craft an attack to avoid detection.

We hypothesize that some of the clustering errors we've seen in our tests of the Autocorrel I system are caused the 40-to-1 mapping, but we do not know the percentage of clustering errors. To calculate the percentage of clustering errors caused by the mapping, we need to determine whether each individual clustering error was caused by the mapping problem, then simply count the number of clustering errors caused by the mapping problem and compare that to the total number of clustering errors.

We note that we do not hypothesize that the reconstruction error formula causes any *separation errors*, the other type of error encountered in our clustering system. Separation errors are errors seen when an optimal cluster is split into two or more separate clusters, so our deterministic 40-to-1 mapping, defined by the reconstruction error formula, would always map numerically similar data items to similar domain values. The problem with the reconstruction error formula is that it can map dissimilar data items to similar domain values, and the effects of this are only clustering errors.

To determine whether a particular clustering error is caused by the 40-to-1 mapping we need to look at the values which compose the computed reconstruction error values. Since E_x, from Equation C.1, is the output of the 40-to-1 mapping, we look the 40 inputs to this formula, $d_{x_i} = x_i - f_i(x)$. The clusters in the Autocorrel I system are formed by examining the differences in behaviour between mappings of the components of $d_x = [d_{x_1}, \ldots, d_{x_n}]$ values for different data items x. That is, two data items x and y should only be clustered together if the values d_{x_i} and d_{y_i} are reasonably similar for all i. If we have a suitable test of similarity for high-dimensional vectors, we can test if two data items were correctly clustered together.

There exist a number of vector similarity measures to quantify this similarity (Manning and Schütze [48]), for example *cosine similarity* and *Euclidean distance*, also known as *geometric distance*. We chose to use Euclidean distance because the measure is well-known and the results are easily interpretable. Manning and Schütze define Euclidean distance between vectors x and y as follows:

$$|x - y| = \sqrt{\sum_{i=1}^{n} [x_i - y_i]^2} \tag{C.2}$$

We use the Euclidean distance measure to measure the distances between all the combinations of two vectors in clusters that contain clustering errors. We computed the distance $|d_x - d_y|$ for all such pairs and found that the distances are significant between intuitively anomalous alerts from a cluster and the rest of the cluster. We should emphasize here the difference in our two uses of the Euclidean distance measure. The reconstruction error formula calculates the Euclidean distance between the neural network inputs and outputs for a particular data item, whereas in the following tests we calculate the Euclidean distance between two data items to measure their numeric similarity.

When running our experiment we found four clusters with clustering error, meaning that there were four system-produced clusters that contained alerts from two or more optimal clusters. (We include both the optimal, hand-clustered clusterings and the system-produced clusterings as appendices to this report, Appendices C.6 and C.7 respectively. We performed the hand-clustering of the alerts in Appendix C.6 by clustering the set of 100 alerts into optimal clusters using our own judgement. We created this set of clusters to be a gold standard, to record by hand the clusters that our automatic system would optimally form.) Clusters 4, 6, 8 and 22 were the four system-produced clusters containing clustering errors.

Figure C.1 shows a matrix of distances between the alerts of cluster 4. Each of the alerts in the dataset is assigned a unique number for easy reference, and the numbers of the alerts in this cluster are 26, 27, 82, 81, 80, 7, 79, 78, 77, and 76. The anomalous alert is alert 7, which is an alert of type “Short UDP Packet”. (The

	26	27	82	81	80	**7**	79	78	77	76
26	0.0000	0.0113	0.0299	0.0357	0.0242	**2.8747**	0.0438	0.0542	0.0639	0.0739
27	0.0113	0.0000	0.0188	0.0258	0.0262	**2.8737**	0.0436	0.0534	0.0627	0.0725
82	0.0299	0.0188	0.0000	0.0108	0.0348	**2.8714**	0.0454	0.0531	0.0611	0.0698
81	0.0357	0.0258	0.0108	0.0000	0.0328	**2.8689**	0.0387	0.0451	0.0523	0.0604
80	0.0242	0.0262	0.0348	0.0328	0.0000	**2.8693**	0.0200	0.0304	0.0401	0.0502
7	**2.8747**	**2.8737**	**2.8714**	**2.8689**	**2.8693**	**0.0000**	**2.8647**	**2.8625**	**2.8603**	**2.8581**
79	0.0438	0.0436	0.0454	0.0387	0.0200	**2.8647**	0.0000	0.0104	0.0202	0.0302
78	0.0542	0.0534	0.0531	0.0451	0.0304	**2.8625**	0.0104	0.0000	0.0098	0.0198
77	0.0639	0.0627	0.0611	0.0523	0.0401	**2.8603**	0.0202	0.0098	0.0000	0.0101
76	0.0739	0.0725	0.0698	0.0604	0.0502	**2.8581**	0.0302	0.0198	0.0101	0.0000

Figure C.1: *Euclidean distances between alerts in cluster 4*

distances in Figure C.1 that involve the anomalous alert, alert 7, are typeset in **bold**.) Here's the Snort representation of this alert (also found in Appendix C.7):

```
[**][116:97:1] Short UDP packet, length field > payload length [**]
11/11-13:29:54.796507 211.194.68.39:0->207.166.72.218:0
UDP TTL:109 TOS:0x0 ID:2062 IpLen:20 DgmLen:78
Len:129
```

The rest of the alerts in this cluster are "Scan nmap TCP" alerts, which are correctly clustered together. Here's the Snort representation of alert 26, a sample of this type of alert:

```
[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:45.976507 167.79.91.3:80->170.129.50.122:53
TCP TTL:49 TOS:0x0 ID:11661 IpLen:20 DgmLen:40
**A**** Seq:0x2A5 Ack:0x0 Win:0x578 TcpLen:20
```

From Figure C.1 you can see that the Euclidean distance between alert 7 and the rest of the alerts in the cluster ranges between 2.85 and 2.88, whereas the distance between any two of the rest is at most 0.07.

Alert 7 is clearly numerically anomalous in this cluster, yet Autocorrel I has clustered it with the other alerts because the reconstruction error value of alert 7 is similar to the reconstruction error values of the other alerts in the cluster. This shows that the reconstruction error formula is responsible for the clustering error in cluster 4.

The matrix in Figure C.2 shows the distances between the alerts of cluster 6. (Again, the distances involving anomalous alerts in this table are typeset in **bold**.) In this cluster there are two anomalous alerts that contribute to the clustering errors. The

	28	29	30	31	**59**	32	33	**58**
28	0.0000	0.0114	0.0224	0.0342	**3.4449**	0.0655	0.0762	**3.4049**
29	0.0114	0.0000	0.0110	0.0228	**3.4431**	0.0543	0.0649	**3.4027**
30	0.0224	0.0110	0.0000	0.0118	**3.4415**	0.0436	0.0542	**3.4006**
31	0.0342	0.0228	0.0118	0.0000	**3.4398**	0.0324	0.0427	**3.3984**
59	**3.4449**	**3.4431**	**3.4415**	**3.4398**	**0.0000**	**3.4353**	**3.4339**	**0.1864**
32	0.0655	0.0543	0.0436	0.0324	**3.4353**	0.0000	0.0108	**3.3927**
33	0.0762	0.0649	0.0542	0.0427	**3.4339**	0.0108	0.0000	**3.3909**
58	**3.4049**	**3.4027**	**3.4006**	**3.3984**	**0.1864**	**3.3927**	**3.3909**	**0.0000**

Figure C.2: *Euclidean distances between alerts in cluster 6*

alerts 58 and 59 are anomalous; alert 58 is of type “SCAN Proxy Port 8080 attempt” and alert 59 is of type “SCAN Squid Proxy attempt”. The other alerts in this cluster are “Scan nmap TCP” alerts, similar to alert 26 that was previously shown. Here are the Snort representations of alerts 58 and 59:

```
[**][1:620:6] SCAN Proxy Port 8080 attempt [**]
11/14-16:00:56.996507 66.159.18.66:43517->170.129.50.120:8080
TCP TTL:53 TOS:0x0 ID:59575 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBED8745 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

```
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43518->170.129.50.120:3128
TCP TTL:53 TOS:0x0 ID:50174 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF3AC0C Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

This cluster is interesting because the two anomalous alerts should actually be correlated together, but not with the rest of the alerts in the cluster. There should be a division placed between the alerts 58 and 59, and the other alerts in the cluster. We know that alert 58 should probably be correlated with alert 59 because the Euclidean distance between the two alerts is about 0.19. In contrast, the distance between the other alerts in the cluster and alerts 58 or 59 is at least 3.40. The distance between any of the two other alerts is at most 0.08, showing that they are correctly clustered together. We can conclude again that the two anomalous alerts are responsible for the clustering errors in cluster 6.

The matrix of real values in Figure C.3 shows the distances between the alerts of cluster 8. There is one anomalous alert in this cluster, alert 60, of type “SCAN SOCKS proxy attempt”. The other alerts in cluster 8 are of type “BAD-TRAFFIC ip reserved bit set”. Here’s the Snort representation of alert 60:

	24	49	57	73	**60**
24	0.0000	0.0000	0.0000	0.0000	**3.7026**
49	0.0000	0.0000	0.0000	0.0000	**3.7026**
57	0.0000	0.0000	0.0000	0.0000	**3.7026**
73	0.0000	0.0000	0.0000	0.0000	**3.7026**
60	**3.7026**	**3.7026**	**3.7026**	**3.7026**	**0.0000**

Figure C.3: *Euclidean distances between alerts in cluster 8*

```
[**][1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43520->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:4253 IpLen:20 DgmLen:60 DF
*****S* Seq:0xB9A7F6F Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
```

And here's the representation of alert 24, typical of the other alerts in the cluster:

```
[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-11:21:09.916507 200.200.200.1->170.129.211.200
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014
```

This cluster is interesting because it is such an extreme example of how the mapping problem caused by the reconstruction error formula causes clustering errors in our system. The Euclidean distance between 60 and all the other alerts is 3.7026 and the Euclidean distance between any two of the other alerts is 0. The distance is exactly 0 because the "BAD-TRAFFIC ip reserved bit set" alerts in this cluster all have exactly the same numerical representation. This is a very strong signal that these alerts should be clustered together, and it's obvious that alert 60 is anomalous to this set.

	40	9
40	**0.0000**	**3.0659**
9	**3.0659**	0.0000

Figure C.4: *Euclidean distances between alerts in cluster 22*

The last cluster containing clustering errors is cluster 22. This cluster contains only two alerts, and the Euclidean distance between the two alerts is 3.0659. (We include this value as a matrix in Figure C.4 to be consistent with the analysis performed of the other clusters.) Alert 40 is a "BARE BYTE UNICODE ENCODING" alert and alert 9 is a "TCP Data Offset is less than 5" alert. Here are the Snort representation of these two alerts:

```
[**][116:46:1] TCP Data Offset is less than 5!  [**]
11/12-18:38:17.846507 203.80.239.162:0->207.166.182.137:0
TCP TTL:107 TOS:0x0 ID:35119 IpLen:20 DgmLen:48 DF
1*UA**** Seq:0x7930005 Ack:0xD80A04D1 Win:0x64BA TcpLen:0 UrgPtr:0x800
```

```
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-13:03:42.666507 170.129.50.120:63598->64.4.22.250:80
TCP TTL:124 TOS:0x0 ID:56064 IpLen:20 DgmLen:932 DF
**AP*** Seq:0x94851318 Ack:0x9AB18054 Win:0x43E1 TcpLen:20
```

It should be clear that these two alerts being clustered together constitute a clustering error. As mentioned, the Euclidean distance is 3.0659. Clearly these alerts lay some distance apart in 40-dimensional space, but Autocorrel I has clustered them together because their reconstruction error values are similar. This small cluster is a very simple example of how the reconstruction error formula causes clustering errors.

In conclusion, all of the clustering errors in this sample clustering were caused by the reconstruction error formula incorrectly mapping from a 40-dimensional space to a single dimension. This experiment is a strong impetus for us to discover a new method of clustering the autoassociator output. While the sample size of this experiment is small, we believe it is representative, and that the reconstruction error formula accounts for a significant portion of the errors produced by Autocorrel I.

C.3 EM Algorithm

The EM algorithm is one of the most well-known clustering algorithms. Initially presented in Dempster *et al* [36], it consists of two repeated steps, expectation and maximization. According to Witten and Frank [37], the EM algorithm uses a statistical model called *finite mixtures* to achieve the goal of producing the most likely set of clusters given the number of clusters, k, and a set of data. The model consists of a set of k probability distributions, one to represent the data each cluster. There are parameters that define each of the k distributions. The EM algorithm begins by making initial guesses for these parameters based on the input data, then determines the probability that a particular data instance belongs to a particular cluster for all data using these parameter guesses. The distribution parameters are revised again and this process is repeated until the resulting clusters have some level of overall cluster “goodness” or until a maximum number of algorithm iterations is reached. The expectation step of the algorithm estimates the clusters of each data instance given the parameters of the finite mixture; the maximization step of the algorithm tries to maximize the likelihood of the distributions that make up the finite mixture, given the data.

We use the public domain WEKA data mining package (Witten and Frank [37]) for its implementation of the EM algorithm. We chose to use this implementation because it is freely available and because the WEKA package is widely used. In this implementation of the algorithm there are a few important tuneable parameters. The `maxIterations` parameter helps control when the algorithm will terminate, and we found that increasing this parameter — to allow the algorithm to produce better clusters — had no affect on the output or the running time of the algorithm, indicating that the algorithm "goodness" converged to a fixed point in less than the default `maxIterations`.

The `seed` parameter is used to randomly seed this algorithm. If we ran this algorithm more than once to produce statistically-sound results, we would change the seed value. Having the `seed` parameter is also a way to ensure the reproducibility results if needed.

The `minStdDev` parameter is used to specify minimum acceptable standard deviation values for the k distributions. We tried varying this parameter but found that it is optimal or nearly optimal at the default value of 0.000001.

The `numClusters` is the value k, which indicates the number of clusters to be formed. If `numClusters` is set to -1, then an optimal number of clusters is deduced using the cluster goodness measure. In practice we've found that setting this parameter to -1 produces very poor results for our data. We tested this parameter extensively with the incidents.org [43] and 1999 Darpa [17] datasets and we found that the algorithm always produces between 1 and 3 clusters, when there is an optimal number of between 7 to 10. Producing such a small number of clusters is problematic because the information produced is effectively worthless. The amount of clustering error (when two dissimilar errors are clustered together) far outweighs the benefit of performing the clustering.

Because the empirically-determined values of k determined by the algorithm are detrimental to clustering, we need a way to find a close-to-optimal value for this parameter. In fact, we find that this parameter is essential to the viability of the output of EM algorithm. In tests with particular datasets — either of the incidents.org or the 1999 Darpa datasets — the best clusterings were found when the parameter k was set to be equal to the number of clusters in the optimal clustering for the particular dataset. For the *incidents.org* dataset we decided that there are optimally 10 clusters, as you can see in Appendix C.6. So when we used the EM algorithm on this set of 100 alerts, without first using the autoassociator on the data and with $k = 10$, it produced very well-formed clusters.

In this test there were surprisingly only four alerts clustered differently than our optimal, hand-clustered groups. Unfortunately, this result can be presented as nothing more than a performance goal because we used the k value of the test set, when in a live situation we wouldn't know the optimal number of clusters for the data we want to cluster. As mentioned we present the result as a performance goal, because it clearly demonstrates the effectiveness of the EM algorithm if correct values of k are chosen.

This variation of our system uses the EM algorithm for clustering without first feeding the data through the autoassociator, so we'll call the variation Em-aa, which we'll later compare to other variations. (An interesting note about this variation was that for our test, the same result was produced regardless of whether we scaled the data beforehand.)

The errors produced in this run of the algorithm (with $k = 10$) give insight into the type of clusters that the EM algorithm produces. Referencing the optimal clustering of Appendix C.6, the first error occured when one of the "SCAN nmap TCP" alerts of cluster 5 was incorrectly placed with the alert from cluster 2, with the "Short UDP packet" alert. The most likely reason that this "SCAN nmap TCP" alert was incorrectly clustered is that it is an anomalous alert within cluster 5. This alert has a destination TCP port of 63874, but all the other alerts in this cluster have destination TCP 53 (for DNS service) or 80 (for web service).

The next error also uncovers an anomalous alert within a greater cluster of alerts. In cluster 1 there is an anomalous alert with title "TCP Data Offset is less than 5", which has its UrgPtr set (in the TCP payload) and also has uncommon TCP flags set. In our run of the Em-aa algorithm this anomalous alert was incorrectly clustered with "SCAN Squid Proxy attempt" alerts in cluster 9.

The final two errors were caused by the fact that the EM algorithm merged clusters 3 (containing "BARE BYTE UNICODE ENCODING" alerts) and 7 (containing "P2P Outbound GNUTella client request" alerts). Because of this, the number of clusters produced by the EM algorithm was only 9, instead of the given $k = 10$.

Despite the errors in this clustering, we were impressed by this result because the EM algorithm helped discover new things about our set of 100, namely that some of our hand-clustered clusters contain anomalous alerts.

Since we know that the EM algorithm works very well with the optimal — yet artificial — k value of the optimal number of clusters, our task may well be to estimate an accurate value for k. To estimate k for this report we look back to our previously presented system Autocorrel I. In that system we formed clusters using our own cluster barrier algorithm, which we've shown produces significantly sub-optimal results. Although this algorithm sometimes produces poorly defined clusters, we use it as a starting point for estimating the value of k. We choose this method of estimating k because it is the best method available to us at the time of writing this report. As we can see from Appendix C.7, our Autocorrel I system estimates k to be 22. We use this value to re-run the previous test, to examine the results of clustering our data with the EM algorithm, given a non-optimal value for k.

The clusters produced by this varied parameter are interesting in their similarity to the clusters produced with $k = 10$. Although we count 28 separation errors in our test run, we have only two clustering errors. (Note that "separation error" is a notation we use for incorrectly clustered alerts occurring when a cluster in the optimal clustering is split into 2 or more clusters in the test clustering.) The two clustering errors occur in a similar situation to the merge of optimal clusters 3 and 7, seen with Em-aa with $k = 10$. That is, once again "BARE BYTE UNICODE ENCODING" alerts are merged together with "P2P Outbound GNUTella client request" alerts. Because these alerts are improbably placed together in two separate runs of the EM algorithm on this data, we can conclude that the EM algorithm truly sees these data items as similar.

All of the remaining errors are those caused by optimal clusters being split in the test clustering. Most of the 28 separation errors are the "SCAN nmap TCP" alerts of optimal cluster 5 being logically split up into probes on different destination IP addresses occuring during different time frames. One could easily argue that some of these errors represent useful gains in information, because the optimal clusters we hand-clustered may be lacking finesse.

Again, despite the errors in this experiment with Em-aa and $k = 22$ we found that the EM algorithm performs well for our purposes of extracting from a set of alerts. From this experiment we can also conclude that it is better to have a value of k that is higher than the optimal rather than lower because although separation error grows, clustering error is generally more damaging, and estimated values less than the optimal k will increase the amount of clustering error.

For this report we also wanted to examine another variation of our system incorporating the EM algorithm. We wanted to test the effectiveness of running the EM algorithm on the outputs of the autoassociator $x_i - f_i(x)$. We call this algorithm variation Em+aa. For this report we didn't plan to do a rigourous analysis of this system, which we leave to the next report. What we do for this report is present the output of this variation, Em+aa, for $k = 22$, with fixed autoassociator parameters. The autoassociator parameters we use for this experiment are the same as those we used in our final report for our previous DRDC contract. We fix the number of neural network training epochs to be 500 and the number of neural network hidden units to be 8. We use a learning rate of 0.05, a training momentum constant of 0.9, and we train the network using the gradient descent with momentum training algorithm.

For our experiment with $k = 22$ we look at the errors produced by the system. We ran this experiment and we counted 6 clustering errors and 26 separation errors.

The first two clustering errors that the Em+aa algorithm made are, unsurprisingly, merging the “BARE BYTE UNICODE ENCODING” alerts of cluster 3 with the “P2P Outbound GNUTella client request” alerts of cluster 7. This error needs no further discuss, with respect to the EM algorithm.

Two other clustering errors were made by merging optimal clusters 8 and 9 together. Both of these clusters contain “Squid Scan Proxy attempt” alerts, and the autoassociator probably helped push the two numerically similar alerts together even though they logically aren’t related.

The final two clustering errors produced by this test run are caused by the lumping together of two anomalous alerts from other clusters. There is one alert in optimal cluster 4, dominated by “BACKDOOR Q access” alerts, that is slightly anomalous. This alert is identical to all the other alerts, except that it has an IP *time-to-live* (TTL) of 14 rather than 15, like all of the other packets in the cluster. The other anomalous alert that this “BACKDOOR Q access” alert was clustered with is a “SCAN nmap TCP” alert from optimal cluster 5. This second anomalous alert has a very high destination TCP port, unlike all of the other “SCAN nmap TCP” alerts in optimal cluster 5.

Similar to the test runs of Em-aa, all of the separation errors in the test run of Em+aa are alerts of type “SCAN nmap TCP” and “SHELLCODE x86 NOOP”. These misclustered alerts are only questionably errors, so we do not spend time analyzing them.

In conclusion for this section, we found that the EM algorithm works very well if it invoked with an optimal number for k, but we have have no way of determining this value exactly, so we estimate it using the autoassociator output from the Autocorrel I system. We found that both the Em-aa and Em+aa algorithms did well with this estimated parameter, and there was no clearly better system. More experimentation is necessary to conclusively determine which is the better algorithm.

C.4 Self-Organizing Maps

Self-organizing maps (SOMs) were created by Kohonen and they are explained in detail in his book [38]. SOMs are an unsupervised method of mapping a high-dimensional input to a two-dimensional output such that similar inputs are mapped to close locations in the two-dimensional map output. (Actually, other dimensionalities are available for output as well, but two-dimensional output is the most common.) This mapping is primarily used for the visualization of high-dimensional data, but it can also be used for clustering. SOMs are a type of neural network, and, accordingly, have similar training parameters. A good document on how to use SOMs is the SOM_PAK program tutorial by Kohonen *et al* [42]. This document explains in depth the possible parameters for SOMs, which we'll only discuss briefly here.

SOMs have a number of different parameters to tune. The *lattice type*, or *topology*, of the network is one such parameter. There are two options for this lattice type, rectangular (also known as grid) and hexagonal. According to Kohonen *et al* [42] the lattice type hexagonal is best suited for visual display. As well, this type is the default for both SOM_PAK and the MATLAB Neural Network Toolbox (Demuth and Beale [31]). Because of this we choose the hexagonal lattice type when experimenting with our SOMs. Another important parameter is the distance function, used to determine the distance between points in the map. We use the default distance function in the MATLAB Neural Network Toolbox, `LINKDIST`.

There are two phases to training an SOM in MATLAB, an ordering phase and a tuning phase. Kohonen *et al* [42] also recommends this two step process. During the ordering phase the training data are given their approximate correct position in the map, and during the tuning phase the positions of the data are fine-tuned. With SOM_PAK you're able to change all the parameters at both these steps, but in MATLAB you can only change the learning rate, `OLR`, and maximum number of steps, `OSTEPS`, at the ordering phase. (`OSTEPS`, the maximum number of training steps, is analogous to the "epochs" parameter for training the autoassociator.) At the tuning phase you only control the learning rate, `TLR`, and the neighbourhood distance function, `TND`.

For this initial test of the potential of SOMs for clustering we use the default values for all these parameters. The default value of `OLR` is 0.9, `OSTEPS` is 1000, `TLR` is 0.02 and `TND` is 1. The learning rate parameter in SOMs is similar to that parameter in other neural networks. It controls how quickly the neural network converges to a desired state and it decays while doing so. The neighbourhood distance function defines the radius of the neighbourhood during training. A larger radius allows the training vectors to move greater distances in the map at each training step.

As with other neural networks, the scaling of the data can become an issue. According to Kohonen *et al* [42], "one may try heuristically justifiable rescalings and check the quality of the resulting maps by means of [...] average quantization errors." We only try one type of scaling, but we plan to experiment with other scaling algorithms in a later report.

Self-organizing maps are not natively intended for clustering, though they can easily be adapted to this application. In fact, the previously mentioned SOM_PAK tutorial only discusses clustering areas of the SOM using previously labelled data, which is not necessarily an option for our dataset. Despite this, there exist papers that analyze clustering with SOMs. Tang *et al* [39] succinctly discuss clustering the output of the SOM. The requirements of their experiments are identical to ours: (1) the need for hard-clustering, (2) automatic determination of the number of clusters k, and (3) robustness of the clusters formed. Given these requirements they chose to employ the *potential function*. This algorithm is described in detail in a paper by Chiu [40], so we do not describe it here. As well, we weren't able to find an existing implementation of this algorithm, so we decided to go with a different method of clustering the SOM, despite the obvious suitability of this method. In a paper by Vesanto and Alhoniemi [41] other methods of clustering the SOM are discussed.

We decided to use the MATLAB implementation of SOMs, which conveniently allows a form of clustering. In this implementation the SOM neural network is trained, then, as with other neural networks, data can be simulated on the network to discover the output of the network for that input. For SOMs the network output of a data instance input is a point in the SOM. Any two vectors which share the same point in the SOM should have similar representation in the higher dimensional space, by the design of the SOM algorithm. With our clustering of the SOM output in MATLAB we simply create one cluster for every point in the SOM output that has at least one data instance associated with it. In practice this produces acceptable clusters, but we would like to experiment with the potential function described by Chiu [40] for our next report.

For this report we decided not to train the SOM with the 10,000 unlabelled training data we've used in training the Autocorrel I system. While we could have used this data to train the SOM for these tests, we chose to only train the SOM with the 100 unlabelled testing data. This allows us to make the results more comparable to those produced by the EM algorithm, because the EM algorithm isn't structured to include a training step. We plan to test the effect of training the SOM with a different set of alerts than the testing set in a future report.

Since our method of clustering the SOM output is to simply look at the data associated with each point in the SOM, the dimensions of the SOM we choose are important. We want to choose dimensions such that the number of points in the SOM is roughly equal to the number of clusters we want to produce. As with the EM-AA and EM+AA algorithms, we use the Autocorrel I system's estimation for the optimal number of clusters present in the dataset. To review, the Autocorrel I system estimates that there are $k = 22$ clusters in the set of 100 unlabelled alerts from the incidents.org dataset. To approximate this value of k, we choose an SOM with x-dimension 4 and y dimension 6. There will be $4 * 6 = 24$ points in the SOM, so there will be at most $k = 24$ clusters in the output.

For the first test we run the SOM on the 100 alerts without involving the autoassociator other than to determine the value of k. We'll call this algorithm SOM-AA. We ran this test and we counted 6 clustering errors and 27 separation errors.

The first clustering errors, involving four incorrectly clustered alerts, clustered together cluster 1 and cluster 6 of the optimal clustering of Appendix C.6. Cluster 1 is composed of "TCP Data Offset is less than 5!" alerts and cluster 6 is composed of "BAD-TRAFFIC ip reserved bit set" alerts. It is clear from the time signatures of the packets and the IP addresses that these two clusters aren't related and thus are incorrectly grouped together.

The other clustering errors, involving two incorrectly clustered alerts, are related to the problem seen with the EM algorithm. Two "P2P Outbound GNUTella client request" alerts are incorrectly clustered with the "BARE BYTE UNICODE ENCODING" alerts, just as with the EM algorithm. This shows us that these two types of alerts must have very similar numerical representations and that both the EM algorithm and the SOM are probably functioning correctly by clustering these types of alerts together. These two alerts are probably clustered together because our method of encoding the alerts make the two different types of alerts look similar.

The remaining errors are separation errors, most of which split optimal cluster 5 into multiple smaller "SCAN nmap TCP" clusters. This situation appears to be similar to the situation we saw with the EM algorithm when the clustering algorithm sees numeric differences where we did not while creating the optimal clustering.

The next test we run with the SOM involves using the SOM to cluster the output of the autoassociator as presented in the Autocorrel I system. We call this algorithm Som+aa. The parameters of this system remain unchanged. The SOM is trained with the same parameters as before. We ran this test and we found 6 clustering errors, like with Som-aa, and 28, rather than 27, separation errors.

The first clustering error in this test run merges optimal clusters 2 and 6. Cluster 2 is a "Short UDP packet" alert and cluster 6 is a group of "BAD-TRAFFIC ip reserved bit set" alerts. Looking at the time signatures and IP addresses for these alerts, it is clear that they should not have been grouped together.

The second clustering error groups the "BARE BYTE UNICODE ENCODING" and "P2P Outbound GNUTella client request" sets of alerts, but also throws in a stray "TCP Data Offset is less than 5!" alert. The clustering of these first two types of alerts is now expected, but the "TCP Data Offset is less than 5!" alert shows a different clustering error.

The final clustering error groups optimal clusters 8 and 9 together, showing a lack of finesse in discerning the numeric differences in the two different optimal clusters containing "SCAN Squid Proxy attempt" alerts.

All of the separation errors were caused by larger clusters, optimal clusters 5 ("SCAN nmap TCP") and 10 ("SHELLCODE x86 NOOP"), being broken into smaller clusters. These smaller clusters are similar to those found with Som-aa.

To conclude this section, we found that Som-aa algorithm outperformed the Som+aa algorithm, but only slightly. The results are so similar that neither algorithm is a clear winner. More experimentation with parameters, especially in training the autoassociator, is necessary to determine if one of these two algorithms is a clear winner.

C.5 Report Summary

We first performed an experiment where it was conclusively shown that our use of the reconstruction error formula we used in Autocorrel I is responsible for a significant number of errors in our testing. As we expected, the type of errors caused by our use of the formula are errors where unrelated alerts are clustered together. This type of error is damaging, so we proceeded to propose solutions to the problem by investigating other clustering algorithms not based on the computation performed by the reconstruction error formula of the autoassociator outputs.

The different variations of the Autocorrel I system we experimented with were Em-aa, Em+aa, Som-aa and Som+aa. For this initial investigation we held most algorithm parameters static, but we plan to investigate these different parameters in depth for the next report. Among the many parameters we'd like to explore, the parameter k, the number of clusters to form, will be one of the most important, since it seems to significantly impact the performance of our clustering systems. We also only presented results for the incidents.org dataset, neglecting the DARPA dataset. We will consider both datasets for the next report.

	Total Error	Clustering Error	Separation Error
Em-aa ($k = 22$)	30	2	28
Em+aa ($k = 22$)	32	6	26
Som-aa	33	6	27
Som+aa	34	6	28

Figure C.5: *Error values for tests on the incidents.org data*

In Figure C.5 we list the total error and the decomposed error for each of the system variations we presented. As you can see, Em-aa did the best, but only slightly. As well, we found that analyzing the errors produced by the systems showed that all of the algorithms produced similar errors. Some of the errors produced by the algorithms were a matter of opinion rather than fact, as well, since we created the testing set. Some of the errors actually uncovered more information about the set of test alerts than what we'd previously known.

Finally, we conclude that we should examine these four system variations more before judging which to use in our final system.

C.6 Appendix: Optimal, Hand-clustered Clusters

```
Cluster 1:
--------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/09-20:51:11.676507 62.13.27.29:0->207.166.33.145:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x81F9750 Ack:0x81F9750 Win:0x0 TcpLen:0

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:12:17.866507 172.20.10.199:0->207.166.119.62:0
TCP TTL:235 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xBDD2D468 Ack:0xBDD2D468 Win:0x0 TcpLen:16

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:28:34.556507 62.13.27.29:0->207.166.78.44:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
****R** Seq:0x91D8C02 Ack:0x91D8C02 Win:0x0 TcpLen:12

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-05:38:56.936507 62.13.27.29:0->207.166.25.86:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xA02C678 Ack:0xA02C678 Win:0x0 TcpLen:8

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-07:20:11.976507 172.20.10.199:0->207.166.168.10:0
TCP TTL:235 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xBF23A8C4 Ack:0xBF23A8C4 Win:0x0 TcpLen:0

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/11-09:08:23.926507 172.20.10.199:0->207.166.207.98:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xC4AD19A6 Ack:0xC4AD19A6 Win:0x0 TcpLen:16

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-03:07:49.886507 210.243.145.141:0->207.166.159.139:0
TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x158662C0 Ack:0x158662C0 Win:0x0 TcpLen:16

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-18:38:17.846507 203.80.239.162:0->207.166.182.137:0
TCP TTL:107 TOS:0x0 ID:35119 IpLen:20 DgmLen:48 DF
```

```
1*UA**** Seq:0x7930005 Ack:0xD80A04D1 Win:0x64BA TcpLen:0 UrgPtr:0x800

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-20:25:11.826507 217.209.183.235:0->207.166.252.249:0
TCP TTL:236 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x24A1C4C Ack:0x24A1C4C Win:0x0 TcpLen:0

Cluster 2:
--------------------
[**][116:97:1] (snort_decoder) Short UDP packet, length field > payload
length [**]
11/11-13:29:54.796507 211.194.68.39:0->207.166.72.218:0
UDP TTL:109 TOS:0x0 ID:2062 IpLen:20 DgmLen:78
Len:129

Cluster 3:
--------------------
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:38.316507 170.129.50.120:63362->159.153.199.24:80
TCP TTL:125 TOS:0x0 ID:38908 IpLen:20 DgmLen:436 DF
**AP*** Seq:0x99AD8FC7 Ack:0x732FBFCD Win:0x4230 TcpLen:20

[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:47.286507 170.129.50.120:63387->159.153.199.24:80
TCP TTL:125 TOS:0x0 ID:38984 IpLen:20 DgmLen:436 DF
**AP*** Seq:0x99C8B003 Ack:0x733D8EE2 Win:0x4230 TcpLen:20

[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-13:03:42.666507 170.129.50.120:63598->64.4.22.250:80
TCP TTL:124 TOS:0x0 ID:56064 IpLen:20 DgmLen:932 DF
**AP*** Seq:0x94851318 Ack:0x9AB18054 Win:0x43E1 TcpLen:20

Cluster 4:
--------------------
[**][1:184:4] BACKDOOR Q access [**]
11/14-09:29:14.826507 255.255.255.255:31337->170.129.172.186:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-09:32:53.016507 255.255.255.255:31337->170.129.132.79:515
```

```
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-09:49:26.156507 255.255.255.255:31337->170.129.129.188:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-10:10:22.596507 255.255.255.255:31337->170.129.195.178:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-10:44:04.836507 255.255.255.255:31337->170.129.30.34:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-11:44:56.156507 255.255.255.255:31337->170.129.137.174:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-12:56:26.746507 255.255.255.255:31337->170.129.89.87:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-13:35:08.896507 255.255.255.255:31337->170.129.200.84:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-14:01:20.986507 255.255.255.255:31337->170.129.23.133:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-14:04:08.966507 255.255.255.255:31337->170.129.190.188:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20
```

```
[**][1:184:4] BACKDOOR Q access [**]
11/14-16:14:32.966507 255.255.255.255:31337->170.129.192.22:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:16:56.986507 255.255.255.255:31337->170.129.134.5:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:17:12.036507 255.255.255.255:31337->170.129.156.132:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:30:57.086507 255.255.255.255:31337->170.129.146.62:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:47:18.156507 255.255.255.255:31337->170.129.176.42:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-17:06:30.606507 255.255.255.255:31337->170.129.80.5:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-17:11:03.616507 255.255.255.255:31337->170.129.1.102:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-17:23:24.646507 255.255.255.255:31337->170.129.41.171:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
```

```
11/14-19:34:10.596507 255.255.255.255:31337->170.129.94.129:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-19:38:10.756507 255.255.255.255:31337->170.129.181.145:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-20:56:56.236507 255.255.255.255:31337->170.129.72.205:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-21:10:50.476507 255.255.255.255:31337->170.129.156.91:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-21:29:40.696507 255.255.255.255:31337->170.129.161.211:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

Cluster 5:
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:03.816507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30084 IpLen:20 DgmLen:40
**A**** Seq:0x134 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:08.786507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30366 IpLen:20 DgmLen:40
**A**** Seq:0x198 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:13.826507 61.218.161.210:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30662 IpLen:20 DgmLen:40
**A**** Seq:0x20A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
```

```
11/14-10:10:18.856507 61.218.161.210:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30943 IpLen:20 DgmLen:40
**A**** Seq:0x278 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:23.856507 163.23.238.9:80->170.129.19.170:80
TCP TTL:44 TOS:0x0 ID:31290 IpLen:20 DgmLen:40
**A**** Seq:0x300 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:52:22.116507 63.211.17.228:80->170.129.50.120:63874
TCP TTL:54 TOS:0x0 ID:563 IpLen:20 DgmLen:40
**A**** Seq:0x3DC Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:45.976507 167.79.91.3:80->170.129.50.122:53
TCP TTL:49 TOS:0x0 ID:11661 IpLen:20 DgmLen:40
**A**** Seq:0x2A5 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:46.046507 167.79.91.3:80->170.129.50.122:53
TCP TTL:47 TOS:0x0 ID:11664 IpLen:20 DgmLen:40
**A**** Seq:0x2A7 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:11.266507 61.222.14.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:48192 IpLen:20 DgmLen:40
**A**** Seq:0x144 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:16.266507 61.222.14.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:48734 IpLen:20 DgmLen:40
**A**** Seq:0x1A8 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:21.266507 61.222.192.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:49258 IpLen:20 DgmLen:40
**A**** Seq:0x20A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:26.276507 61.222.192.98:80->170.129.81.112:80
```

```
TCP TTL:49 TOS:0x0 ID:49820 IpLen:20 DgmLen:40
**A**** Seq:0x271 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:41.366507 210.66.117.5:80->170.129.81.112:80
TCP TTL:47 TOS:0x0 ID:51450 IpLen:20 DgmLen:40
**A**** Seq:0x39E Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:46.366507 210.66.117.5:80->170.129.81.112:80
TCP TTL:47 TOS:0x0 ID:51968 IpLen:20 DgmLen:40
**A**** Seq:0x400 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:49.906507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28299 IpLen:20 DgmLen:40
**A**** Seq:0x11B Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:54.916507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28601 IpLen:20 DgmLen:40
**A**** Seq:0x18D Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:59.916507 61.218.161.210:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28911 IpLen:20 DgmLen:40
**A**** Seq:0x209 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:52:04.916507 61.218.161.210:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:29197 IpLen:20 DgmLen:40
**A**** Seq:0x279 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:52:09.906507 163.23.238.9:80->170.129.14.62:80
TCP TTL:44 TOS:0x0 ID:29544 IpLen:20 DgmLen:40
**A**** Seq:0x2FD Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:28:54.256507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23204 IpLen:20 DgmLen:40
**A**** Seq:0x2E6 Ack:0x0 Win:0x578 TcpLen:20
```

```
[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:04.266507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23404 IpLen:20 DgmLen:40
**A**** Seq:0x348 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:14.306507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23584 IpLen:20 DgmLen:40
**A**** Seq:0x3A6 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:24.186507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23770 IpLen:20 DgmLen:40
**A**** Seq:0x0 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:34.136507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23947 IpLen:20 DgmLen:40
**A**** Seq:0x5D Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:18.166507 61.218.161.202:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:64698 IpLen:20 DgmLen:40
**A**** Seq:0x26 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:23.136507 61.218.161.202:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:64986 IpLen:20 DgmLen:40
**A**** Seq:0x8C Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:28.126507 61.218.161.210:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:65271 IpLen:20 DgmLen:40
**A**** Seq:0x100 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:33.266507 61.218.161.210:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:32 IpLen:20 DgmLen:40
**A**** Seq:0x17A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
```

```
11/14-14:50:38.206507 163.23.238.9:80->170.129.151.28:80
TCP TTL:44 TOS:0x0 ID:336 IpLen:20 DgmLen:40
**A**** Seq:0x1EC Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:29.606507 61.218.15.118:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:7722 IpLen:20 DgmLen:40
**A**** Seq:0x92 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:34.876507 61.218.15.118:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:8252 IpLen:20 DgmLen:40
**A**** Seq:0xF4 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:39.636507 61.218.15.126:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:8768 IpLen:20 DgmLen:40
**A**** Seq:0x158 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:44.536507 61.218.15.126:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:9312 IpLen:20 DgmLen:40
**A**** Seq:0x1BF Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:54.676507 61.221.88.198:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:10358 IpLen:20 DgmLen:40
**A**** Seq:0x286 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:34:00.056507 192.192.171.251:80->170.129.69.49:80
TCP TTL:44 TOS:0x0 ID:10868 IpLen:20 DgmLen:40
**A**** Seq:0x2EA Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:34:04.926507 192.192.171.251:80->170.129.69.49:80
TCP TTL:44 TOS:0x0 ID:11428 IpLen:20 DgmLen:40
**A**** Seq:0x353 Ack:0x0 Win:0x578 TcpLen:20

Cluster 6:
--------------------
[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
```

```
11/14-11:21:09.916507 200.200.200.1->170.129.211.200
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-14:37:18.296507 200.200.200.1->170.129.2.16
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-15:54:39.456507 200.200.200.1->170.129.79.180
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-17:59:31.346507 200.200.200.1->170.129.239.44
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

Cluster 7:
--------------------
[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.096507 170.129.50.120:61121->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22720 IpLen:20 DgmLen:158 DF
**AP*** Seq:0x5A0CA92C Ack:0x53A65413 Win:0x4038 TcpLen:20

[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.316507 170.129.50.120:61122->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22766 IpLen:20 DgmLen:62 DF
**AP*** Seq:0x5A129557 Ack:0x53A7A3BA Win:0x4038 TcpLen:20

Cluster 8:
--------------------
[**][1:620:6] SCAN Proxy Port 8080 attempt [**]
11/14-16:00:56.996507 66.159.18.66:43517->170.129.50.120:8080
TCP TTL:53 TOS:0x0 ID:59575 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBED8745 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43518->170.129.50.120:3128
```

```
TCP TTL:53 TOS:0x0 ID:50174 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF3AC0C Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

[**][1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43520->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:4253 IpLen:20 DgmLen:60 DF
*****S* Seq:0xB9A7F6F Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

[**][1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43521->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:2662 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF46594 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

Cluster 9:
--------------------
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:21.366507 206.48.61.139:4006->170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:24710 IpLen:20 DgmLen:48 DF
*****S* Seq:0x13D84F7 Ack:0x0 Win:0x2000 TcpLen:28
TCP Options (4) => MSS:536 NOP NOP SackOK

[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:24.316507 206.48.61.139:4006->170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:26758 IpLen:20 DgmLen:48 DF
*****S* Seq:0x13D84F7 Ack:0x0 Win:0x2000 TcpLen:28
TCP Options (4) => MSS:536 NOP NOP SackOK

Cluster 10:
--------------------
[**][1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
11/14-16:10:30.806507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:56986 IpLen:20 DgmLen:1420 DF
**A**** Seq:0x8217BFFC Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.566507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46490 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA074E240 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
```

```
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.576507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46491 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA074E7A4 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.676507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0x0 ID:46498 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0750D60 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.756507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46504 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0752DB8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.026507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0x0 ID:46521 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075895C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.146507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46529 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075B47C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.216507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46532 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075C4A8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.296507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46538 IpLen:20 DgmLen:1420 DF
**AP*** Seq:0xA075E500 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.526507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46553 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA07635DC Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.626507 129.118.2.10:57425->170.129.50.120:63414
```

```
TCP TTL:51 TOS:0xA0 ID:46560 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0765B98 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.856507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46575 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076AC74 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.866507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46576 IpLen:20 DgmLen:1420 DF
**AP*** Seq:0xA076B1D8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.876507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46577 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076B73C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:38.016507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46584 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076DCF8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:38.936507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46646 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0782B30 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
```

C.7 Appendix: Autocorrel I-produced Clusters

Note: all alerts considered to be the source of cluster errors are in *italics*.

```
Cluster 1:(2.9280)
--------------------
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:21.366507 206.48.61.139:4006->170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:24710 IpLen:20 DgmLen:48 DF
*****S* Seq:0x13D84F7 Ack:0x0 Win:0x2000 TcpLen:28
TCP Options (4) => MSS:536 NOP NOP SackOK

Cluster 2:(2.9336)
--------------------
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:06:24.316507 206.48.61.139:4006->170.129.23.239:3128
TCP TTL:114 TOS:0x0 ID:26758 IpLen:20 DgmLen:48 DF
*****S* Seq:0x13D84F7 Ack:0x0 Win:0x2000 TcpLen:28
TCP Options (4) => MSS:536 NOP NOP SackOK

Cluster 3:(3.1047)
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:03.816507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30084 IpLen:20 DgmLen:40
**A**** Seq:0x134 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:08.786507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30366 IpLen:20 DgmLen:40
**A**** Seq:0x198 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:13.826507 61.218.161.210:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30662 IpLen:20 DgmLen:40
**A**** Seq:0x20A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:18.856507 61.218.161.210:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30943 IpLen:20 DgmLen:40
**A**** Seq:0x278 Ack:0x0 Win:0x578 TcpLen:20
```

```
[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:23.856507 163.23.238.9:80->170.129.19.170:80
TCP TTL:44 TOS:0x0 ID:31290 IpLen:20 DgmLen:40
**A**** Seq:0x300 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:49.906507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28299 IpLen:20 DgmLen:40
**A**** Seq:0x11B Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:54.916507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28601 IpLen:20 DgmLen:40
**A**** Seq:0x18D Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:59.916507 61.218.161.210:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28911 IpLen:20 DgmLen:40
**A**** Seq:0x209 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:52:04.916507 61.218.161.210:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:29197 IpLen:20 DgmLen:40
**A**** Seq:0x279 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:52:09.906507 163.23.238.9:80->170.129.14.62:80
TCP TTL:44 TOS:0x0 ID:29544 IpLen:20 DgmLen:40
**A**** Seq:0x2FD Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:28:54.256507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23204 IpLen:20 DgmLen:40
**A**** Seq:0x2E6 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:04.266507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23404 IpLen:20 DgmLen:40
**A**** Seq:0x348 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:14.306507 202.29.28.1:80->170.129.238.112:80
```

```
TCP TTL:45 TOS:0x0 ID:23584 IpLen:20 DgmLen:40
**A**** Seq:0x3A6 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:24.186507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23770 IpLen:20 DgmLen:40
**A**** Seq:0x0 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:29:34.136507 202.29.28.1:80->170.129.238.112:80
TCP TTL:45 TOS:0x0 ID:23947 IpLen:20 DgmLen:40
**A**** Seq:0x5D Ack:0x0 Win:0x578 TcpLen:20

Cluster 4:(3.1137)
--------------------
[**] [116:97:1] (snort_decoder):Short UDP packet, length field > payload
length [**]
11/11-13:29:54.796507 211.194.68.39:0->207.166.72.218:0
UDP TTL:109 TOS:0x0 ID:2062 IpLen:20 DgmLen:78
Len:129

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:45.976507 167.79.91.3:80->170.129.50.122:53
TCP TTL:49 TOS:0x0 ID:11661 IpLen:20 DgmLen:40
**A**** Seq:0x2A5 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:02:46.046507 167.79.91.3:80->170.129.50.122:53
TCP TTL:47 TOS:0x0 ID:11664 IpLen:20 DgmLen:40
**A**** Seq:0x2A7 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:29.606507 61.218.15.118:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:7722 IpLen:20 DgmLen:40
**A**** Seq:0x92 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:34.876507 61.218.15.118:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:8252 IpLen:20 DgmLen:40
**A**** Seq:0xF4 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
```

```
11/14-20:33:39.636507 61.218.15.126:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:8768 IpLen:20 DgmLen:40
**A**** Seq:0x158 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:44.536507 61.218.15.126:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:9312 IpLen:20 DgmLen:40
**A**** Seq:0x1BF Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:54.676507 61.221.88.198:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:10358 IpLen:20 DgmLen:40
**A**** Seq:0x286 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:34:00.056507 192.192.171.251:80->170.129.69.49:80
TCP TTL:44 TOS:0x0 ID:10868 IpLen:20 DgmLen:40
**A**** Seq:0x2EA Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-20:34:04.926507 192.192.171.251:80->170.129.69.49:80
TCP TTL:44 TOS:0x0 ID:11428 IpLen:20 DgmLen:40
**A**** Seq:0x353 Ack:0x0 Win:0x578 TcpLen:20

Cluster 5:(3.1334)
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:33.266507 61.218.161.210:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:32 IpLen:20 DgmLen:40
**A**** Seq:0x17A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:38.206507 163.23.238.9:80->170.129.151.28:80
TCP TTL:44 TOS:0x0 ID:336 IpLen:20 DgmLen:40
**A**** Seq:0x1EC Ack:0x0 Win:0x578 TcpLen:20

Cluster 6:(3.1518)
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:11.266507 61.222.14.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:48192 IpLen:20 DgmLen:40
**A**** Seq:0x144 Ack:0x0 Win:0x578 TcpLen:20
```

```
[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:16.266507 61.222.14.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:48734 IpLen:20 DgmLen:40
**A**** Seq:0x1A8 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:21.266507 61.222.192.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:49258 IpLen:20 DgmLen:40
**A**** Seq:0x20A Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:26.276507 61.222.192.98:80->170.129.81.112:80
TCP TTL:49 TOS:0x0 ID:49820 IpLen:20 DgmLen:40
**A**** Seq:0x271 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:41.366507 210.66.117.5:80->170.129.81.112:80
TCP TTL:47 TOS:0x0 ID:51450 IpLen:20 DgmLen:40
**A**** Seq:0x39E Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-12:14:46.366507 210.66.117.5:80->170.129.81.112:80
TCP TTL:47 TOS:0x0 ID:51968 IpLen:20 DgmLen:40
**A**** Seq:0x400 Ack:0x0 Win:0x578 TcpLen:20

[**] [1:620:6] SCAN Proxy Port 8080 attempt [**]
11/14-16:00:56.996507 66.159.18.66:43517->170.129.50.120:8080
TCP TTL:53 TOS:0x0 ID:59575 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBED8745 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

[**] [1:618:5] SCAN Squid Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43518->170.129.50.120:3128
TCP TTL:53 TOS:0x0 ID:50174 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF3AC0C Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

Cluster 7:(3.1637)
--------------------
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.296507 129.118.2.10:57425->170.129.50.120:63414
```

```
TCP TTL:51 TOS:0xA0 ID:46538 IpLen:20 DgmLen:1420 DF
**AP*** Seq:0xA075E500 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.866507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46576 IpLen:20 DgmLen:1420 DF
**AP*** Seq:0xA076B1D8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

Cluster 8:(3.1789)
--------------------
[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-11:21:09.916507 200.200.200.1->170.129.211.200
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-14:37:18.296507 200.200.200.1->170.129.2.16
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-15:54:39.456507 200.200.200.1->170.129.79.180
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

[**] [1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43520->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:4253 IpLen:20 DgmLen:60 DF
*****S* Seq:0xB9A7F6F Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-17:59:31.346507 200.200.200.1->170.129.239.44
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014

Cluster 9:(3.1835)
--------------------
[**][1:615:5] SCAN SOCKS Proxy attempt [**]
11/14-16:00:56.996507 66.159.18.66:43521->170.129.50.120:1080
TCP TTL:53 TOS:0x0 ID:2662 IpLen:20 DgmLen:60 DF
*****S* Seq:0xBF46594 Ack:0x0 Win:0x16D0 TcpLen:40
```

```
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0

Cluster 10:(3.2179)
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:18.166507 61.218.161.202:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:64698 IpLen:20 DgmLen:40
**A**** Seq:0x26 Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:23.136507 61.218.161.202:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:64986 IpLen:20 DgmLen:40
**A**** Seq:0x8C Ack:0x0 Win:0x578 TcpLen:20

[**][1:628:3] SCAN nmap TCP [**]
11/14-14:50:28.126507 61.218.161.210:80->170.129.151.28:80
TCP TTL:48 TOS:0x0 ID:65271 IpLen:20 DgmLen:40
**A**** Seq:0x100 Ack:0x0 Win:0x578 TcpLen:20

Cluster 11:(3.2413)
--------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/09-20:51:11.676507 62.13.27.29:0->207.166.33.145:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x81F9750 Ack:0x81F9750 Win:0x0 TcpLen:0

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:12:17.866507 172.20.10.199:0->207.166.119.62:0
TCP TTL:235 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xBDD2D468 Ack:0xBDD2D468 Win:0x0 TcpLen:16

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-05:38:56.936507 62.13.27.29:0->207.166.25.86:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xA02C678 Ack:0xA02C678 Win:0x0 TcpLen:8

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-07:20:11.976507 172.20.10.199:0->207.166.168.10:0
TCP TTL:235 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xBF23A8C4 Ack:0xBF23A8C4 Win:0x0 TcpLen:0

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
```

```
11/11-09:08:23.926507 172.20.10.199:0->207.166.207.98:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0xC4AD19A6 Ack:0xC4AD19A6 Win:0x0 TcpLen:16

[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-20:25:11.826507 217.209.183.235:0->207.166.252.249:0
TCP TTL:236 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x24A1C4C Ack:0x24A1C4C Win:0x0 TcpLen:0

Cluster 12:(3.2484)
--------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-03:07:49.886507 210.243.145.141:0->207.166.159.139:0
TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40
****R** Seq:0x158662C0 Ack:0x158662C0 Win:0x0 TcpLen:16

Cluster 13:(3.2768)
--------------------
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.566507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46490 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA074E240 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.576507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46491 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA074E7A4 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.756507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46504 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0752DB8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.146507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46529 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075B47C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.216507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46532 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075C4A8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
```

```
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.526507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46553 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA07635DC Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.626507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46560 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0765B98 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.856507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46575 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076AC74 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.876507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46577 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076B73C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:38.016507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46584 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA076DCF8 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:38.936507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46646 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0782B30 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

Cluster 14:(3.2972)
--------------------
[**][1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
11/14-16:10:30.806507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:56986 IpLen:20 DgmLen:1420 DF
**A**** Seq:0x8217BFFC Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

Cluster 15:(3.3117)
--------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-10:52:22.116507 63.211.17.228:80->170.129.50.120:63874
```

```
TCP TTL:54 TOS:0x0 ID:563 IpLen:20 DgmLen:40
**A**** Seq:0x3DC Ack:0x0 Win:0x578 TcpLen:20

Cluster 16:(3.3743)
--------------------
[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.096507 170.129.50.120:61121->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22720 IpLen:20 DgmLen:158 DF
**AP*** Seq:0x5A0CA92C Ack:0x53A65413 Win:0x4038 TcpLen:20

Cluster 17:(3.3848)
--------------------
[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.316507 170.129.50.120:61122->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22766 IpLen:20 DgmLen:62 DF
**AP*** Seq:0x5A129557 Ack:0x53A7A3BA Win:0x4038 TcpLen:20

Cluster 18:(3.4330)
--------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:28:34.556507 62.13.27.29:0->207.166.78.44:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
****R** Seq:0x91D8C02 Ack:0x91D8C02 Win:0x0 TcpLen:12

Cluster 19:(3.4421)
--------------------
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:38.316507 170.129.50.120:63362->159.153.199.24:80
TCP TTL:125 TOS:0x0 ID:38908 IpLen:20 DgmLen:436 DF
**AP*** Seq:0x99AD8FC7 Ack:0x732FBFCD Win:0x4230 TcpLen:20

[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:47.286507 170.129.50.120:63387->159.153.199.24:80
TCP TTL:125 TOS:0x0 ID:38984 IpLen:20 DgmLen:436 DF
**AP*** Seq:0x99C8B003 Ack:0x733D8EE2 Win:0x4230 TcpLen:20

Cluster 20:(3.4689)
--------------------
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.676507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0x0 ID:46498 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA0750D60 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
```

```
[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:37.026507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0x0 ID:46521 IpLen:20 DgmLen:1420 DF
**A**** Seq:0xA075895C Ack:0x90CF9E29 Win:0x16D0 TcpLen:20

Cluster 21:(3.4770)
--------------------
[**][1:184:4] BACKDOOR Q access [**]
11/14-09:29:14.826507 255.255.255.255:31337->170.129.172.186:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-09:32:53.016507 255.255.255.255:31337->170.129.132.79:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-09:49:26.156507 255.255.255.255:31337->170.129.129.188:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-10:10:22.596507 255.255.255.255:31337->170.129.195.178:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-10:44:04.836507 255.255.255.255:31337->170.129.30.34:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-11:44:56.156507 255.255.255.255:31337->170.129.137.174:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-12:56:26.746507 255.255.255.255:31337->170.129.89.87:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20
```

```
[**][1:184:4] BACKDOOR Q access [**]
11/14-13:35:08.896507 255.255.255.255:31337->170.129.200.84:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-14:01:20.986507 255.255.255.255:31337->170.129.23.133:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-14:04:08.966507 255.255.255.255:31337->170.129.190.188:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:14:32.966507 255.255.255.255:31337->170.129.192.22:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:16:56.986507 255.255.255.255:31337->170.129.134.5:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:17:12.036507 255.255.255.255:31337->170.129.156.132:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:30:57.086507 255.255.255.255:31337->170.129.146.62:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-16:47:18.156507 255.255.255.255:31337->170.129.176.42:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
```

```
11/14-17:06:30.606507 255.255.255.255:31337->170.129.80.5:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-17:11:03.616507 255.255.255.255:31337->170.129.1.102:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-17:23:24.646507 255.255.255.255:31337->170.129.41.171:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-19:34:10.596507 255.255.255.255:31337->170.129.94.129:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-19:38:10.756507 255.255.255.255:31337->170.129.181.145:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-20:56:56.236507 255.255.255.255:31337->170.129.72.205:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-21:10:50.476507 255.255.255.255:31337->170.129.156.91:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

[**][1:184:4] BACKDOOR Q access [**]
11/14-21:29:40.696507 255.255.255.255:31337->170.129.161.211:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
**A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20

Cluster 22:(3.4861)
--------------------
[**] [116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!
```

```
[**]
11/12-18:38:17.846507 203.80.239.162:0->207.166.182.137:0
TCP TTL:107 TOS:0x0 ID:35119 IpLen:20 DgmLen:48 DF
1*UA**** Seq:0x7930005 Ack:0xD80A04D1 Win:0x64BA TcpLen:0 UrgPtr:0x800
```

```
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-13:03:42.666507 170.129.50.120:63598->64.4.22.250:80
TCP TTL:124 TOS:0x0 ID:56064 IpLen:20 DgmLen:932 DF
**AP*** Seq:0x94851318 Ack:0x9AB18054 Win:0x43E1 TcpLen:20
```

This page intentionally left blank.

Annex D: Second Progress Report

D.1 Introduction

The purpose of the work reported in this document is to select one of the candidate clustering algorithms introduced in the previous report and find an optimal or at least acceptable setting for its parameters and training regime. In particular, we compare the performance of different clustering algorithms and we construct experiments that assess the value of using a training dataset.

In our first report for this contract we reported two conclusions. Firstly, we found that the 40-to-1 mapping generated by the reconstruction error, combined with the simplistic one-dimensional clustering algorithm, is a significant source of error in our previous system Autocorrel I. Secondly, we reported that more research is needed to determine which of the two candidate clustering algorithms (self-organizing maps or the EM algorithm) should be selected for further use. We reported that these two algorithms perform similarly well, and we expand on the initial report to develop a more rigourous comparison.

In our proposal for this research contract, we planned to report on three aspects of our research in this second report. We proposed to perform a detailed analysis of combining the autoassociator with self-organizing maps (Kohonen [38]) or the EM algorithm (Dempster *et al* [36]). We also proposed to analyze the parametric values of the autoassociator in our system. Lastly, we proposed to perform analysis of the effect of training the autoassociator.

We're able to complete the first two tasks in one experiment. We discuss this experiment in Section D.2.1. In this experiment we vary three important neural network parameters of the Autocorrel I system, then report the results using self-organizing maps and the EM as clustering algorithms. These results are presented in Section D.3.1.

In the work description for this research contract, we were charged with the task of investigating the effect of training the autoassociator in the Autocorrel I system. Because we've already concluded that some changes to the Autocorrel I system must be made, we investigate the effect of training neural networks in this new system. To this end, we propose an experiment to fully analyze the impact of both (i) using a large number of unlabelled alert data to train our system and (ii) using just a self-organizing map for clustering rather than using the autoassociator and a self-organizing map together.

We present the summarized conclusions of the experiments of Section D.3 in Section D.4.

D.2 Technical Overview

This section explains the experiments we'll perform and discusses how the experiments will uncover the aspects of our system in which we're interested.

D.2.1 Clustering Algorithm Comparison

In our last report we concluded that the reconstruction error formula in Autocorrel I is responsible for many of the clustering errors seen in our testing. (We decompose errors in our system into two types. We call the type of error that incorrectly clusters unrelated alerts together "clustering errors". The other type of error is called "separation error", which occurs when two related alerts are not clustered together.) The single-link, one-dimensional clustering algorithm (explained in Jain *et al* [47]) with which we used to form discrete clusters was a heuristic byproduct of our use of the reconstruction error formula. We also presented weak initial evidence in our last report that either of the EM algorithm or self-organizing maps perform better at clustering than the reconstruction error formula combined with the single-link algorithm. With the following experiment, we will be able to rigourously investigate the performance of either of these clustering algorithms compared with the clustering method using in Autocorrel I.

In our first experiment we'll compare the number of clustering and separation errors produced by various combinations of machine learning algorithms on two datasets. Namely, we vary the autoassociator's neural network parameters *hidden units* and *epochs*, and we feed the 40 autoassociator outputs to 3 clustering algorithms: the EM algorithm, self-organizing maps, and the single-link clustering algorithm that we previously used.

To compare the outputs of the various modifications of our system, we've taken 100-consecutive alert windows from both the 1999 DARPA dataset (Lippmann *et al* [17]) and the incidents.org dataset [43]. We clustered these 100 alerts by hand, just as in the first report. (We choose 100 alert windows because it didn't take too long to hand cluster 100 alerts, and 100 alerts was sufficient to discover experimental trends. This parameter of how many consecutive alerts are in each window can be changed.) These two hand-clustered groups of alerts form our *gold standards* for the two datasets; they represent what we'd like our system to produce automatically. Accordingly, we use these two gold standards as validation sets to which we compare our automatically-generated clusters.

The hidden units parameter in neural networks defines the learning capacity of the network. The hidden unit layer of a multi-layer perceptron neural network is responsible for allowing the learning of non-linear concepts. There exists a (possibly non-unique) optimal number of hidden units in a neural network for a given training set. By tuning this neural network parameter, one can approximate the optimal number of hidden units, and thus attain better performance in the particular domain. In the Autocorrel I system we fixed the number of hidden units of the autoassociator at 8 because we didn't see any significant differences in the performance of the system at different values. For this report, we take a more rigourous approach, and compare the output of our system against predefined desired outputs — the gold standard. Because of the existence of a gold standard, we consider the following four candidate hidden units numbers: 8, 16, 32, and 64.

The epochs parameter in neural networks defines the number of iterations of the neural network training algorithm. As the neural network becomes more trained, the neural network is more capable of recognizing the particular data on which it's being trained. One notable pitfall of the training process is that if the neural network is trained for too many epochs, then *overfitting* may occur. An overfitted neural network is a neural network that has been trained for more epochs than it should have for optimal performance. It described the state of the network when it will no longer generalize to the testing data (or other data that is at all different from the training data), and thus cannot be used as a general classifier (or clustering algorithm). (See Pattern Classification by Duda, Hart and Stork [45] for a more detailed analysis of the problem of overfitting.) For this report, we examine three candidate epoch values for the autoassociator: 500, 2500, and 5000.

In the previous report we described the details of the clustering algorithms we'll consider for this report – please refer to that report to understand how the EM algorithm (Dempster *et al* [36]) and self-organizing maps (Kohonen [38]) work. Since the self-organizing maps algorithm has parameters, we include them in this report. The number of training epochs is fixed at 1000 and the dimensions of the map lattice are 4×6.

D.2.2 Neural Network Training

In the work description for this research contract, it was clearly stated that the technology to be used for network event correlation is to be neural networks. Self-organizing maps are a type of neural network, but the EM algorithm is a statistical, non-neural network algorithm. Because of this contract constraint, self-organizing maps are the preferred algorithm, so we'll move forward using this algorithm for experimentation. Also, since the EM algorithm performs similarly to self-organizing maps in the task of clustering alert data (Section D.3.1), we do not feel that further examination of the EM algorithm is needed. We focus on the autoassociator and self-organizing maps, two neural network architectures, for this section on training.

Training is a common process for most data mining and machine learning applications. Training in data mining or machine learning represents the learning part of the process; this idea is based on the fact that, in most applications, the past is a good predictor of the future. But in the domain of alert correlation, this assumption is violated. Julisch *et al* [26] explicitly mention the idea that traditional data mining training may not be valuable in the alert correlation domain. McHugh [49] mentions the fact that each site that runs an IDS will have a unique distribution of alarm types. Julisch *et al* strengthen this by showing that even a given network's distribution of alerts changes from month-to-month.

Julisch *et al* [26] is a paper that describes a method for alert correlation that uses a conceptual clustering algorithm of their own creation. They resolve the problem of training by relying on clustering algorithms rather than classification algorithms, similar to our own approach. They do require some training in their system, though, to train the system which alerts can be handled automatically, since their system explicitly has the goal to reduce the number of alerts presented to the administrator. The training of their system fits into the general framework they've created for intrusion detection system operation. They advocate having the administrator hand-modify the correlation system once per month to reflect the changes in the network environment. As well, in their system they require that the correlation system has information specific to the local network before initial operation of the system. They require a seasoned analyst to perform this initial configuration.

Dain *et al* [27] have presented another data mining-centered correlation system in their paper "Fusing a Heterogeneous Alert Stream into Scenarios". In this paper, Dain *et al* train machine learning algorithms such as neural networks and decision trees to recognize attack scenarios based on a novel list of features. Hätälä *et al* [21] criticize the Dain *et al* work for it's use of a simplistic dataset. The Dain *et al* research is only tested with a DEF CON conference dataset. The use of this dataset simplifies the problem of alert correlation because attackers are motivated by points in the competition and no points are awarded for stealthy attacks. As such, many of the attacks originate from a single IP address (Dain *et al* [27]).

We have a methodological criticism of the Dain *et al* work. In the explanation of their preparation of the dataset, they offer the detail that, "a script was written to randomly split the data into [training, evaluation and testing] files." This statement implies that care was not taken to ensure that all training and evaluation instances occurred before any testing instances, and in fact that it is very likely that many testing instances were derived from events occurring before training instances. This is a methodological problem because it doesn't model a real world realtime system, where the future is truly unknown. As well, because the authors split up individual attack scenarios over the training and testing datasets, the system was trained on one half of an attack scenario then tested on a different part of the same attack scenario, which is implausible for any realistic training set. That their datasets were unfairly biased towards success accounts for their marvellous performance in the domain.

(Their system attained a performance level of over 99%. Another reason this figure is so high is the way they formulated their experiment. The machine learning algorithm gets credit for many correct decisions not to place alerts into existing scenarios, even when it has placed the alert in question into an incorrect scenario.) In fact, because of this methodological error, their testing results can be discounted, regardless of the novel ideas they present in their paper.

The papers by Julisch *et al* [26] and Dain *et al* [27] represent the most serious attempts by researchers to apply machine learning algorithms to the domain of alert correlation. Both of these papers rely to some extent on training, and both papers present novel ideas for doing so. In our system we wish to investigate whether this type of training is necessary for reasonable performance of a correlation system. We acknowledge information useful to the correlation of some alerts lies outside the presentation of those alerts (see Cuppens *et al* [52] for a potent example of such information), but we wish to consider the possibility that a non-rule-based correlation system that doesn't rely on external training data can add value to an IDS. We do this by considering a variation of the Autocorrel I system that is not trained using the 10,000 unlabelled data instances, unlike the original Autocorrel I. It should be noted that even when we train our system, we are doing so in a different way than Julisch *et al* and Dain *et al* did so in their systems. In their systems training data is labelled, so that the machine learning algorithm can internally formulate performance goals. In our system, we train with unlabelled data, which we claim helps the autoassociator contextualize data to be clustered – leading to better algorithm performance – and also helps the autoassociator form consistent clusters. We examine this first assertion in the next experiment. The autoassociator is not able to target its performance, unlike in a traditional classification problem. The training algorithm has no precise goals, but we provide some goals with our gold standard, and we help the system achieve these goals by examining the system while varying system parameters. (We can also change the distributions of the alerts in the training data to enhance performance, but we do not explore this option here.)

To determine whether the use of unlabelled training data has a positive effect on the performance of the autoassociator, we compare the results of the experiment in Section D.2.1 to a new variation of the Autocorrel I system that doesn't use the 10,000 unlabelled training data. For this experiment, we train the autoassociator using the validation dataset, then reconstruct the same data on the neural network after training has completed. What we expect is that the autoassociator may be able to find more subtle numerical differences between data items, since it has intimately learned the data. We do not use the 10,000 unlabelled training data from the Autocorrel I system at any point in this experiment, but the 100 validation data on which we train the autoassociator is not labelled from the perspective of the autoassociator.

This experiment will show the effect of the training dataset on the performance because this experiment is almost identical to the experiment in Section D.2.1, except that the 10,000 unlabelled training data are not used. The results of the two experiments are directly comparable.

For this experiment, we examine the system with 8, 16, and 32 hidden units. We evaluate the performance against the validation data at 500, 2500 and 5000 epochs. We use self-organizing maps to form discrete clusters with the autoassociator output data. The self-organizing maps are trained for 1000 epochs on the same data that they cluster. The self-organizing maps are configured in a 4×6 lattice.

The last experiment will evaluate what value is added by involving the autoassociator in the clustering process. To do this, we try to cluster the validation data using only self-organizing maps. Similar to the previous experiment, we train the self-organizing maps on the unlabelled validation data, then cluster that same data. This experiment is comparable to the previous experiment because this experiment – like the last one – doesn't make use of the 10,000 unlabelled training data. Like the last experiment, the clustering algorithm is first trained on the validation dataset, then used to cluster that same set.

A more typical use of self-organizing maps would train the maps on a representative set of unlabelled training data, then cluster the validation data based on this trained mapping to the self-organizing map lattice (Kohonen [38]). The validation data can then be classified depending whether the training data is classified, and depending on where the validation data appears in the map lattice. Using self-organizing maps in this way gives the results more stability and context if the training dataset remains static. We would have trained our self-organizing maps on the 10,000 unlabelled training data, as with the Autocorrel I system for direct comparison to the results in Section D.2.1, but this training was infeasible. The process of training with this data for only a very short number of epochs took so long that we concluded that training the self-organizing map for a reasonable number of epochs would be infeasible.

D.3 Experimental Results

In this section we present the results of the experiments we outlined in Section D.2. Unlike previous reports, we focus on the numbers of errors made, rather than thoroughly analyzing the types of errors made. We take this approach because we'll be comparing the output of many systems, and the number of errors reported for each of these systems is an adequate measure of their performance against the gold standard clusters.

The results of different clustering algorithms are only comparable if the validation dataset remains invariant because different sets of data have different innate characteristics. Accordingly, we'll group the results for the incidents.org dataset separately from the results for the 1999 DARPA dataset within each section.

D.3.1 Clustering Experiment

First, we'll present the results we obtained for our first experiment, which we outlined in Section D.2.1. In this experiment we compare the results of clustering the incidents.org validation dataset of 100 alerts using a trained autoassociator and three different clustering algorithms: the single-link heuristic algorithm we developed for Autocorrel I, the EM algorithm, and self-organizing maps. In the following tables, we list the total number of errors (TE), the clustering errors (CE), and the separation errors (SE).

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	29	8	21	37	3	34	37	3	34
16	47	8	39	48	8	40	47	6	41
32	42	2	40	43	1	42	38	0	38
64	40	0	40	40	1	39	43	5	38

Figure D.1: *Results of the autoassociator and the single-link algorithm on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	32	4	28	39	3	36	43	2	41
16	44	0	44	44	0	44	44	0	44
32	39	2	37	40	0	40	46	2	44
64	39	2	37	43	1	42	38	1	37

Figure D.2: *Results of the autoassociator and the EM algorithm on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	34	6	28	33	4	29	31	3	28
16	33	4	29	33	4	29	34	4	30
32	29	4	25	28	2	26	28	5	23
64	35	4	31	30	4	26	30	4	26

Figure D.3: *Results of the autoassociator and self-organizing maps on incidents.org dataset*

We see from Figures D.1, D.2, and D.3 that the lowest number of total errors we achieved is 28, achieved by self-organizing maps clustering the output of the autoassociator with 32 hidden units trained for 2500 and 5000 epochs. We see that this system, if trained for 2500 epochs, has fewer clustering errors than the system trained for 5000 epochs. Since we regard clustering errors as more harmful than separation errors – we have discussed why there is a difference in importance of these two types of errors in previous reports – we can conclude that the best system performance in this experiment was attained by the self-organizing maps with an autoassociator trained for 2500 epochs.

To analyze the effectiveness of the three clustering algorithms, it might be useful to consider the averages of total errors and clustering errors for each of Figures D.1, D.2, and D.3. We show these averages in Figure D.4. (Note that "SOM" abbreviates self-organizing maps.) The values in *italics* are the variance measures of the total errors.

	Average Errors		
Clustering Algorithm	TE	CE	SE
Single-link	40.9 (*28.8*)	3.8	37.2
EM	40.9 (*14.8*)	1.4	39.5
SOM	31.5 (*6.1*)	4	27.5

Figure D.4: *Clustering algorithm average errors for incidents.org dataset*

We see from Figure D.4 that self-organizing maps clearly have the lowest overall average number of errors. Another interesting point of this table is that the EM algorithm produces a lower average number of clustering errors than the other two clustering algorithms.

Next, we use the same algorithms with the same parameters to test which system performs best with the 1999 DARPA dataset. In Figures D.5, D.6, and D.7 we see that all of the clustering algorithms preform poorer than on the incidents.org dataset. The reason for this is that our gold standard for the 1999 DARPA differs from the innate structures in the dataset.

While we may not be able to achieve precisely the gold standard for this dataset, the clusters produced by the systems are still valuable. For instance, we see that the number of clustering errors is fairly low, implying that the clustering algorithms found differences within clusters of the gold standard.

We see that our best preformance lies with the EM algorithm on this dataset, with 16 hidden units in the autoassociator. The system performs equally well at both 2500 epochs and 5000 epochs.

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	52	6	46	52	0	52	52	4	48
16	52	2	50	50	5	45	48	1	47
32	51	4	47	50	2	48	51	0	51
64	48	0	48	50	0	50	50	0	50

Figure D.5: *Results of autoassociator and the single-link algorithm on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	44	3	41	61	0	61	62	3	59
16	63	4	59	43	3	40	43	3	40
32	44	8	36	51	4	47	60	3	57
64	65	3	62	45	3	42	64	3	61

Figure D.6: *Results of autoassociator and the EM algorithm on 1999 DARPA dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	62	4	58	61	4	57	57	7	50
16	61	3	58	61	2	59	59	3	56
32	58	6	52	60	9	51	59	2	57
64	60	9	51	58	6	52	58	5	53

Figure D.7: *Results of autoassociator and self-organizing maps on 1999 DARPA dataset*

Although the EM algorithm produces the best single system for the 1999 DARPA dataset, the results produced by this algorithm vary significantly more than the results produced by self-organizing maps and the single-link clustering algorithm, indicating that the results produced by this algorithm may be less stable. We see in Figure D.8 that the EM algorithm was responsible for the best performing system, but that the single-link clustering algorithm outperformed the EM algorithm on average.

	Average Errors		
Clustering Algorithm	TE	CE	SE
Single-link	50.5 (*2.1*)	2	48.5
EM	53.8 (*89.3*)	3.3	50.4
SOM	59.5 (*2.5*)	5	54.5

Figure D.8: *Clustering algorithm average errors for 1999 DARPA dataset*

From Figure D.8 we see that the single link algorithm has both the lowest average total errors as well as the lower average clustering errors. Another interesting fact presented in this table is that self-organizing maps performed the poorest on this dataset. They had both the highest average total error and the highest average clustering error. We find this interesting, because they performed the best on the incidents.org dataset.

One thing we notice immediately from the tables with self-organizing map results in Figures D.3 and D.7 is that the results vary less than the other results. Figure D.8 reports the variance in the total error statistic for the three algorithms on the two datasets in *italics*. From this we can hypothesize that self-organizing maps form more stable clusters, which are less likely to be changed by the intermediate step of using the autoassociator. This could be considered both a positive and a negative feature of the algorithm. Stability of results is good because it can imply a predictable end system, but it can be bad at the research stage if different results are needed and the output of the algorithm won't produce them.

One last thing we'd like to point out about the previous experiments is that the single-link algorithm seems to preform much better if the autoassociator is constructed with 32 or 64 hidden units. One can see in Figures D.1 and D.5 that with 64 hidden units at 500 epochs, in particular, the single-link algorithm produces very few clustering errors. This is a very desirable property of any algorithm we choose, so we'll have to continue considering the single-link autoassociator combo in future research.

D.3.2 Training Experiments

In the next experiment we investigate the impact of using training data in the Autocorrel I system. To investigate this, we conduct an experiment where we train the autoassociator on the validation data, rather than training the autoassociator on the 10,000 unlabelled training data. We see the statistics of this experiment on the incidents.org dataset in Figure D.9 and on the 1999 DARPA dataset in Figure D.10.

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	33	5	28	34	6	28	34	4	30
16	29	2	27	31	2	29	28	2	26
32	27	1	26	32	5	27	28	2	26

Figure D.9: *Results of autoassociator trained on validation data, clustered with self-organizing maps, on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
Hidden Units	TE	CE	SE	TE	CE	SE	TE	CE	SE
8	63	3	60	60	6	54	61	3	58
16	61	8	53	58	2	56	63	9	54
32	62	3	59	64	14	50	65	6	59

Figure D.10: *Results of autoassociator trained on validation data, clustered with self-organizing maps, on 1999 DARPA dataset*

	Average Errors		
Dataset	TE	CE	SE
incidents.org	30.7 (*7.5*)	3.2	27.4
1999 DARPA	61.9 (*4.7*)	6	55.9

Figure D.11: *Validation-data trained autoassociator average errors for both datasets*

We see from Figure D.11 that the system produces very similar averages to the averages produced by the equivalent systems using the 10,000 unlabelled training data. From Figure D.3 we see that the autoassociator using the self-organizing maps clustering algorithm produces an average number of errors of 31.5, whereas the validation-data-trained version produces 30.7 errors on average. Also, from Figure D.7 we see that 10,000 data trained system produces 59.5 errors on average, and the comparable validation-data-trained system produces 61.9 errors on average.

This experiment shows that the system doesn't benefit and isn't impaired by the use of the unlabelled training data. The fact that the averages are so close for the two datasets shows that the training data doesn't affect performance.

We acknowledge a small methodology problem with the previous experiment though. We didn't test the autoassociator with 64 hidden units, like the experiments in Section D.3.1. We believe this is only a minor problem, though, since it is unlikely that the test with 64 hidden units would have varied wildly from the other tests.

For the last experiment, we train the self-organizing maps without first simulating the data using the autoassociator. We train the self-organizing maps on the validation data, just as in the last experiment. Figures D.12 and D.13 display the results of the algorithm for various epochs and for lattice dimensions 4 × 6 and 5 × 7, for each dataset. Figure D.14 summarizes the results.

	Epochs=500			Epochs=2500			Epochs=5000		
SOM dimensions	TE	CE	SE	TE	CE	SE	TE	CE	SE
4×6	33	6	27	33	4	29	28	2	26
5×7	36	5	31	37	2	35	30	2	28

Figure D.12: *Results of self-organizing maps trained on validation data on incidents.org dataset*

	Epochs=500			Epochs=2500			Epochs=5000		
SOM dimensions	TE	CE	SE	TE	CE	SE	TE	CE	SE
4×6	60	3	57	57	5	52	60	7	53
5×7	61	5	56	66	7	59	66	5	61

Figure D.13: *Results of self-organizing maps trained on validation data on 1999 DARPA dataset*

	Average Errors		
Dataset	TE	CE	SE
incidents.org	32.8 (*11.8*)	3.5	29.3
1999 DARPA	61.7 (*13.1*)	5.3	56.3

Figure D.14: *Average performance for self-organizing maps trained on the validation data datasets*

We can see clearly from Figure D.14 that self-organizing maps, without involving the autoassociator, perform very similarly to all other uses of the self-organizing maps. We can conclude that not involving the autoassociator in the Autocorrel I system doesn't boost performance in our problem.

We also see that self-organizing maps have a low variance, just as in the other experiments with this algorithm. From this we can conclude that self-organizing maps produce stable clusters.

D.4 Conclusions

We conclude this report with a recap of the results we presented in Section D.3.

In our experiments we found that the EM algorithm produces unstable results, and that self-organizing maps produce very stable results. We found that using self-organizing maps as the clustering algorithm produced the best results for our incidents.org validation set, and we found that using the EM algorithm produced the best results for the 1999 DARPA dataset.

We also found the surprising result that the simple heuristic-based single-link clustering algorithm from Autocorrel I is able to form clusters without clustering errors if the autoassociator is constructed with 64 hidden units. Although we showed in the first report that the reconstruction error formula sometimes maps very different numerical vectors to the same real values, causing clustering errors in the Autocorrel I system, we find here that if we choose the number of hidden units carefully and do not overfit the autoassociator, we're able to attain a very low number of clustering errors. This is due to the fact that a higher number of hidden units allows a different type of reconstruction of data, pushing numerically similar alert representations into tighter clusters and separating numerically dissimilar alert representations by wider margins. As well, luck plays a role in this result. The 40-to-1 mapping problem would still crop up if we were to test this improved system on more data. Despite our luck, though, we can safely conclude that the Autocorrel I system produces substantially better results in this domain with the hidden units autoassociator parameter higher than 16.

Clearly exploring the system on more data with finer-grained values for the hidden units and epochs parameters would be interesting, but we are sufficiently satisfied with our cluster results in this report, so we'll move on to the task of finding higher-level correlations between these clusters in the next report.

We also found that the use of the 10,000 unlabelled training data doesn't negatively impact performance. As well, we found that, in using self-organizing maps, their performance is not dependent on the use of the autoassociator with that algorithm.

In fact, we found that the single most important variable in controlling the output of our system is the choice of clustering algorithm. We found that the different clustering algorithms uncovered different trends in the data, and combined the data in different ways. No one clustering algorithm had close to optimal performance on both datasets. Because of this result, we will move forward with this research project using the output of two clustering algorithms: (i) self-organizing maps (without the autoassociator involved) trained for 2500 epochs, with the lattice dimensions 4×6, and (ii) the autoassociator, with the single-link clustering algorithm, constructed with 64 hidden units and trained for 500 epochs.

This page intentionally left blank.

Annex E: Third Report

E.1 Introduction

The purpose of the work reported in this document is to create a system of alert correlation that can find correlation between different steps of an attack. In the previous reports for this project we developed a system that is able to cluster alerts such that each cluster represents a different step of an attack. For this report we worked towards a system that will cluster together different steps of a single attack.

In the previous report for this contract we discovered reasonable parameters for the use of self-organizing maps and the autoassociator as clustering algorithms for our domain. Inevitably, these algorithms will work differently on the new set of data with new features that we experiment with in this report. We also explored the role of training in the Autocorrel I system in the previous report.

In our last report we also discussed some methodological and more fundamental problems in papers by other authors that have applied data mining to the problem of alert correlation. In this report we construct our new system partially based on these systems, particularly using the ideas for feature construction from Dain and Cunningham [27].

In our proposal for this research contract, we planned to report on feature selection for this third report. We considered the results of our last report satisfactory, and we have decided to move on to the pressing question of determining more valuable correlations in the data. Accordingly, we will not do as we intended in the project proposal; we will not perform rigourous feature selection on the features we've presented previously.

For this report, we take the output clusters of the previous stage and represent them each as new data items, each encoded using a new set of features. The construction of the features of one's data ultimately determines what is learned in machine learning problems. So, we intentionally construct our features such that a cluster of alerts representing a single step of an attack will be similar to another cluster of alerts from the same attack. We construct our features such that the converse is also true, by necessity.

In Section E.2 we discuss the new architecture of our approach and the way we implement this new approach, specifically the construction of new features to represent a cluster. In Section E.3 we present the experiments we use to evaluate the performance of our new system, and we hypothesize on the results of these experiments. In Section E.4 we report on the results of the experiments we proposed in Section E.3. In the final section of this report, Section E.5, we conclude our findings for this report.

E.2 Technical Overview

This section explains the new approach of our system, and explains how we'll construct the features to implement this new approach.

E.2.1 New Architecture

In our system up to now we've used an almost unchanging set of features to represent alerts that we wanted to cluster. This set of features was created to fully represent an IP packet that has been flagged as an alert by an intrusion detection system (IDS), in particular the Snort IDS in our experiments. The feature set allows similar-looking IP packets to be clustered together. Together, this feature set combined with our use of the autoassociator neural network, they cluster IP packets received in a given window into individual steps of greater attacks.

For the purpose of our research we define a "step" of an attack to be one action in the greater attacker's plan. For instance, one step would be running the security tool `nmap` once against a network to discover what services are available. This step wouldn't include the use of tools to attack particular services, or other reconnaisance tools used by the same attacker.

With the revised system we present in this report, we hope to be able to cluster, using a larger test window, each of the clusters previously produced effectively to form *super-clusters*, or clusters of clusters of alerts. From this we hope to link clusters from our previous system that are related, but that do not form one discrete cluster. We hope to be able to link different steps of an attack together by representing each step such that it can be correctly clustered with other steps from the same attack.

The motivation for this new architecture comes from Valdes and Skinner [25] in their paper "Probablistic Alert Correlation". In their approach they use statistical methods primarily based on the type of the alert and the source and destination IP addresses of the alert. They correlate alerts using three distinct steps. The first step threads the alerts of a given sensor so that each logical step of an attack at a given sensor isn't represented multiple times. The second step of their system combines the output of the multiple sensors to fuse the data into one stream. After this second step each step of an attack should be represented only once in their system. For the final step of their system, they try to correlate their already formed groups of alerts based on which groups are part of the same greater attack. At this step they use source IP addresses to determine attackers and destination IP addresses to determine targets. They also use predefined matrices of values that represent how likely it is that two types of alerts are related, if seen in succession.

In our system we combine the first two steps of Valdes and Skinner's system into one step, namely our system as we presented it up to now. Our system is fairly successful at threading a set of alerts, and we believe — without proof, but with strong indication, since our methods are sensor independent — that our system could handle multi-sensor threading. (We haven't proven this latter claim because multi-sensor data isn't available to us.)

The new adjustment to our system of adding another level of clustering effectively mimics Valdes and Skinner's intuitive approach of finding attack step correlation after the threading and multi-sensor correlation has been done. We should note, though, that both of the other systems we model our approach on, the systems of Dain and Cunningham [27] and Julisch and Dacier [26], do not use a multi-step correlation approach. The difference is that their systems are both supervised data mining systems, whereas ours in unsupervised.

E.2.2 New Features

We link different steps of an attack by clustering the output of our previous system. By representing the output clusters of our previous system as individual data items using the correct features, we can cluster similar output clusters to form new super-clusters. The success of our system will rely on how we construct features to represent each cluster.

We take our motivation for feature construction at this level from the paper "Fusing a Heterogeneous Alert Stream into Scenarios" by Dain and Cunningham [27]. In this paper the authors describe a number of features to be used in their trained binary learning system.

Before describing the features in detail, we present our methods on determining the similarity of IP addresses, since it differs slightly with Dain and Cunningham's method. A measure of closeness between two IP addresses is used in the Dain and Cunningham system that is applied to source and destination IP addresses of alerts. Their system closeness is defined as the maximal number of most-significant bits shared between two IP addresses (called r), when an IP address is encoded as a 32-bit integer in the same way we have done for Autocorrel I. (Namely, IP address w.x.y.z is turned into the 32-bit integer $256^3w + 256^2x + 256y + z$.) So, for instance, in their system if they saw one IP address 192.168.2.17 and another IP address 192.168.2.16, these would be deemed very similar ($r = 31$), whereas a source address 192.168.1.16 would be considered less similar to the previous two ($r = 22$). (See their paper [27] for further details.)

Because our system is a clustering system where the features of all data are compared at once for similarity, we cannot implement a measure to compare just two IP addresses. As such, we implement a feature that makes two IP addresses similar if and only if they would be similar in the Dain and Cunningham system. In our system, we represent the similarity of a group of IP addresses by representing them as the IP addresses of the group with the unshared least significant bits masked out. So, using the previous examples, the group of IP addresses (192.168.2.16, 192.168.2.17) would be represented with the single IP address 192.168.2.16, and the group of IP addresses (192.168.2.16, 192.168.2.17, 192.168.1.16) would be represented as 192.168.0.0.

We now discuss the features that Dain and Cunningham use, and the analogous features in our system, in the following list. (Each of the items of text in the following list is preceeded by the name of the feature in our system.)

1. `ipSrcAddrCommonPart`: Attackers often prepetrate an attack from a single host, or from a single IP subnet. So Dain and Cunningham include a feature to indicate similarity between the source IP addresses of two alerts. In our system, we have a feature to indicate the shared part of the most-significant bits of a group of IP addresses from a cluster.

2. `ipDestAddrCommonPart`: Dain and Cunningham asserted that attackers often target a single host or subnet in their attacks. So they include a feature indicating the similarity of two destination IP addresses. In our system we constructed a feature for destination IP addresses to be analogous of the one for source IP addresses.

3. `ipSrcAddrCommonBits`: Dain and Cunningham say, "An attack scenario may contain components with spoofed source IP addresses while other components of the attack may use the attackers real source IP." As such, they include a feature to indicate the minimal r value found between the source IP of an alert and the source IPs of a group of alerts. We include the exact same feature in our system, but computed such that the feature indicates the minimal value of r between any two source IPs in a cluster.

4. `avgTimeSig` and `varTimeSig`: Dain and Cunningham have features to indicate the similarity in time between when two alerts were generated. In our system we have features to indicate the average time when a group of alerts was generated, and the standard deviation (named `varTimeSig`) for the times when a group of alerts were generated.

5. `avgReconsErr`: Dain and Cunningham have features to indicate whether a new alert is of the same type as the most recent alerts of already established groups. In our system, we have a feature to indicate the average reconstruction error of a group of alerts. As we've established in the previous reports, the reconstruction errors of alerts tend to correspond directly with their type. (Note: this feature isn't available if the clustering algorithm from our previous step wasn't based on the autoassociator.)

There are a few features from the Dain and Cunningham paper that we weren't able to represent in our system. In their system (and in other correlation systems) they include a feature to indicate the similarity of a source IP address and a destination IP address. They suggest that this might be useful in classification because the source IP address of a new alert can be the destination of an old one, if an IP address in your network has been compromised in a previous part of an attack. Because of the structure of their data (the packet captures of a DEFCON hacking contest) it is very unlikely that this feature was used for what they designed it. Attackers didn't use the victim machines as "stepping stones" in the contest because the contest wasn't structured that way. In our data we haven't seen any evidence of the use of stepping stones, so while we were able to design a feature to encode this information, we chose not to implement it.

Another set of features they included in their system related to a technical problem of the RealSecure IDS that they used in their experimenting. Apparently this IDS sometimes switches the source and destination IP address in alerts, so they had features to account for that. In our system, using only Snort to experiment with, we didn't see any evidence of this, so we didn't feel there was a problem to overcome by creating new features.

We also experimented with some other features that Dain and Cunningham didn't consider. We list them below in the same format as the previous list.

1. `modePortSrc` and `modePortDest`: We found that using the TCP source and destination ports was sometimes valuable in determining the grouping of a set of alerts. We also found that when the ports were useful, there was almost always a particular source or destination port that was much more common than the rest. Accordingly, we created two features: one to encode the most common TCP source port in a cluster of alerts, and one to encode the most common destination port.

2. `avgSeqNumDiff`: We also found that TCP sequence numbers and TCP acknowledgement numbers are sometimes good predictors of the use of a hacker tool. Often there are obvious patterns in a group of sequence or acknowledgement numbers, and this pattern can usually be encoded by taking the difference of the two numbers. Doing this also helps in the specific situation where there is an explicit relationship between these two sets of numbers within a cluster. We encoded the average difference between these two numbers for each cluster.

E.3 Experiments Proposed

In our previous experiments with the Autocorrel I system, we found that if we used a base window of 100 alerts, we would produce about 25 clusters with our system.

Obviously this figure would hopefully vary dramatically with the dataset we're using (and thus with the number of innate clusters in the data), but we found this to be a good estimate in general. In our new system, where each cluster produced by our old system becomes its own data item, clustering only 25 clusters together wouldn't give us meaningful results. So what we decided to do is to run Autocorrel I, with the parameters we discovered in our last report, on five consecutive 100-alert windows. Once Autocorrel I had produced alerts for each of these five windows, we ran our new feature construction methods on each of the five windows then combined all of the data produced into one dataset. We didn't run Autocorrel I on just the 500-alert window because we would have had to modify our previously created gold standards, and because we might have had to spend time experimenting with parameters — notably the cluster barrier parameter used in our one-dimensional single-link clustering. (See Jain *et al* [47] for a description of this type of algorithm.)

We ended up with 83 data items for our new dataset. Since the purpose of this report was to experiment with feature construction for this dataset, we hold the number of data items in the set static, and just experiment by analyzing the output when we select particular features that we've constructed.

We decided to only use the incidents.org dataset [43] for this exercise. We considered using the 1999 DARPA dataset [17] as well, but we decided that it might be valuable to see how well our system performs on a different dataset after we've discovered what parameters work well. We hope that our system will not have to be tuned for every new data, or every month as with the Julisch and Dacier system [26], so saving a data for testing only will allow us to test how successful we are at that.

We also created a new gold standard to indicate how we'd like to see our system perform. We do not include more than just a small part of this gold standard, because it is so large. We include a sample of this new gold standard as Appendix E.6. There are 25 super-clusters in this new gold standard.

We created this new gold standard based on the gold standards we'd created for each of the five 100-alert windows. Since each of the 100-alert windows had their own gold standard, the new gold standard represents each of the 100-alert window gold standard clusters as a single number. For instance, super-cluster 1 of the new gold standard has six members: 1.1, 1.2, 1.3, 2.5, 4.12, and 4.13. In this notation, 1.2 stands for cluster 2 of 100-alert window 1, 4.12 stands for cluster 12 of 100-alert window 4, *et cetera*. This was a convenient way to encode the relationships between clusters within super-clusters.

Our results in these experiments are based on the results of clustering already flawed data, which causes errors to propagate. We could have chosen to use the five 100-alert window gold standard clusters as our the base data for this experiment — effectively creating super-clusters from this already prefectly clustered data — but we felt that this wouldn't give a realistic indication of the performance of our system. This would have been an interesting experiment, though, because it would have allowed us to see how well this new part of our system works independently of the old part. We would have liked to perform experiments of this sort as well, but we couldn't due to time constraints, and we felt that giving an indicator of the overall performance of our system was most important.

The next two sections contain the experiments we'll perform and discuss how the experiments will uncover the aspects of our system in which we're interested.

E.3.1 Experimenting with the Dain and Cunningham Features

For this experiment we test our system with only the features taken from Dain and Cunningham paper [27]. So, we use the following features for this experiment: `ipSrcAddrCommonPart`, `ipDestAddrCommonPart`, `ipSrcAddrCommonBits`, `avgTimeSig`, `varTimeSig`, and `avgReconsErr`.

In this experiment we want to test which clustering algorithms perform best for our new dataset. To this end, we experiment with all of the autoassociator [35], the EM algorithm [36], and self-organizing maps [38], as we did for our experiments in the last report. We are guided by some of the results from our last report; for instance, we found that combining the autoassociator with self-organizing maps doesn't produce better results. We also know that the autoassociator, using the single-link, one-dimensional clustering algorithm to do the actual clustering [47], seems to work well with 64 hidden units. Also, we use a 4×6 lattice for the self-organizing maps, since this has worked well previously.

It is not possible to use training data for this problem because our training data is unlabelled, and thus has no discrete clusters from which would could construct our new features. So we do not use training data as such for any of the experiments here. We do however train the autoassociator and self-organizing maps on the data that we are about to cluster, as we've done previously. We hold the number of epochs for this training at 500 for the autoassociator, which we have found to be a good number in the previous reports.

We will vary one parameter for each of the three algorithms we use here. For the autoassociator and single-link clustering algorithm, we're interested in observing the effect of varying the cluster barrier parameter (previously held constant at 0.0025 for our previous experiments) because in our initial experimentation with our new system we found that this parameter will be the greatest indicator of performance. We will test this parameter between values 0.0 and 0.03, testing at increments of 0.0001.

With self-organizing maps, we'll vary the number of epochs that we train for testing for all values between 0 epochs and 5000 epochs, at 50 epoch intervals.

For the EM algorithm, the number of clusters to form is the parameter that we'll vary. We'll vary this parameter between the values of 5 and 60, testing at increments of 5.

What we hope to gain from these experiments is a sense of how well our new scheme performs given only the features we've adapted directly from the Dain and Cunningham paper [27]. We also hope to be able to find out which clustering algorithm is best suited to our new data.

We expect that our new system will perform better with the other new features we've presented in Section E.2.2, because we find that using these other aspects of the data is very useful when doing clustering by hand, such as while we were creating the gold standard. We cannot hypothesize about which algorithm we expect to perform best at this time, though. All of the different clustering algorithms perform well on different datasets, and the data we're using for this new system is radically different from that of our previous system.

E.3.2 Experimenting with All Features

For this next experiments we select a different set of features, hopefully to contrast the performance with these features against those selected for the experiments in Section E.3.1. Specifically, for these experiments we'll use the features: `ipSrcAddrCommonPart`, `ipDestAddrCommonPart`, `ipSrcAddrCommonBits`, `avgTimeSig`, `varTimeSig`, `avgReconsErr`, `modePortSrc`, `modePortDest`, and `avgSeqNumDiff`. With these further parameters we hope to attain better performance than the new system as considered in the previous section.

The experiments in this section are the same as those for Section E.3.1, except, obviously, for the change in selection of features. So, our experiments for this section will be with self-organizing maps while varying the number of training epochs, the autoassociator with the single-link one-dimensional clustering algorithm while varying the cluster barrier parameter, and the EM algorithm while varying the number of clusters to produce.

E.4 Experimental Results

Before analyzing the results of our experiments, we need to explain a bit about the evaluation of our performance under the new system.

In our new system we had the option of scoring how well our system does given the old system. We could have matched clusters produced by our old system to their optimal gold standard clusters, then evaluated how well our new system did with respect to the gold standard super-clusters. We chose not to use this system of error-counting, though, because it would have been fraught with problems. There is no perfect match between our gold standard clusters and the clusters produced by our old system. Inevitably, smaller clusters produced by our old system would not have been counted, so we wouldn't have had a very realistic evaluation of our system.

Previously we decomposed errors in our system into two types. We call the type of error that incorrectly clusters unrelated alerts together "clustering errors". The other type of error is called "separation error", which occurs when two related alerts are not clustered together. We will still use this terminology in our new system of evaluation.

We decided to evaluate the performance of our new system by translating our new super-clusters in the set of alerts represented by the sub-clusters in our super-clusters. For instance, using the example given previously, if super-cluster 1 has sub-clusters 1.1, 1.2, 1.3, 2.5, 4.12, and 4.13, and sub-cluster 1.1 has alerts 1, 2, 3, 4, 5, 6, 8, and 10, sub-cluster 1.2 has alert 9, sub-cluster 2.5 has alert 146, *et cetera*, then we translate super-cluster 1 into the set of alerts 1, 2, 3, 4, 5, 6, 8, 10, 9, 7, 146, 303, and 324. Once this translation is done, we can compare all of the alerts of our gold standard of super-clusters to the actual results in the same way that we compared results in our previous reports.

By reporting our results in this way, we get a good sense of the overall performance of our system, since two alerts are counted as successfully being clustered together if and only if they appear in the same, correct super-cluster.

We now move on to the task of reporting and analyzing our results. We do this in Section E.4.1 and Section E.4.2.

E.4.1 Results with the Dain and Cunningham Features

We present the results for the experiments of Section E.3.1 in graphical form. The performance of the autoassociator is listed in Figure E.1, the performance of the self-organizing maps is in Figure E.2, and the performance of the EM algorithm is in Figure E.3.

As you can see from Figure E.1, the results of varying the cluster barrier parameter produce much more predicable results than the results for either of Figure E.2 or Figure E.3. That said, the best performance attained by the autoassociator as a clustering algorithm under these conditions was 287 errors, which is more the average

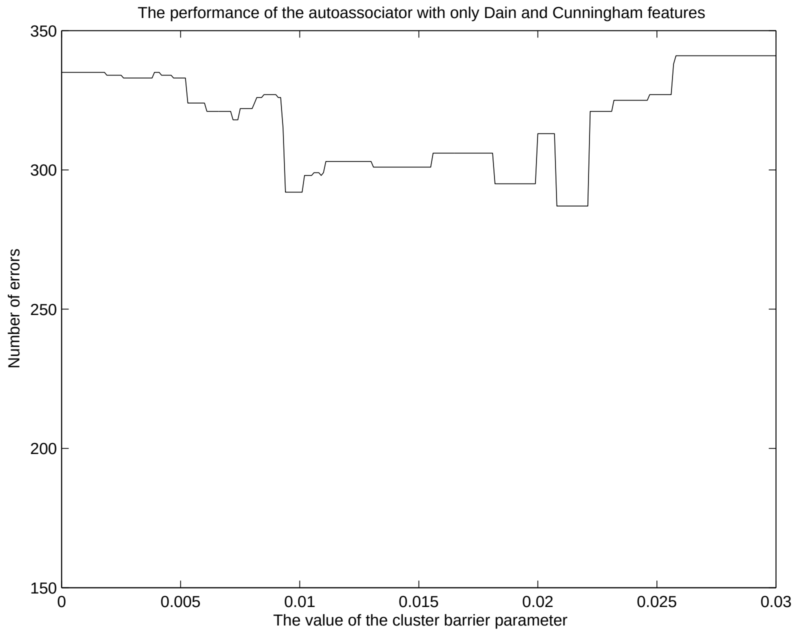

Figure E.1: *Autoassociator varying cluster barrier for Dain* et al *features*

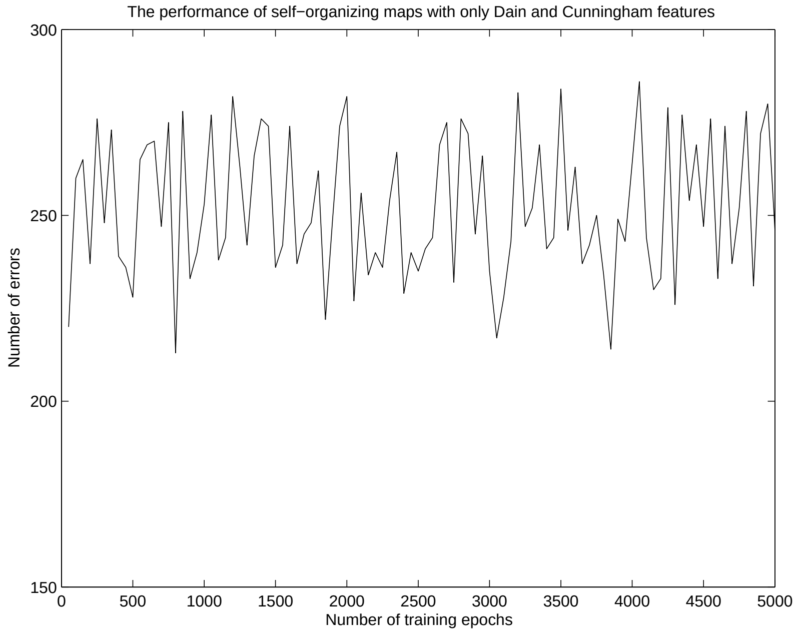

Figure E.2: *Self-organizing maps varying epochs for Dain* et al *features*

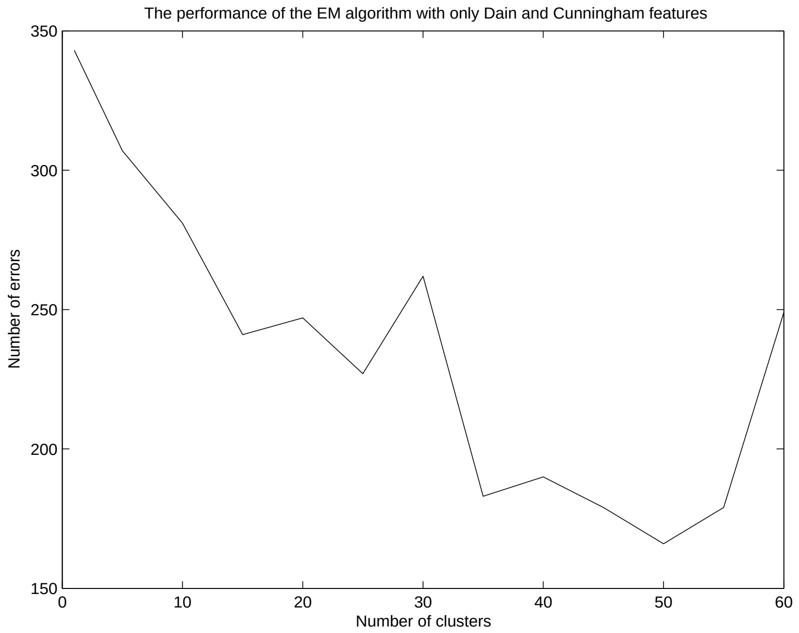

Figure E.3: *EM algorithm varying number of clusters for Dain* et al *features*

number of errors for the entire range of self-organizing maps tests. The autoassociator also produced noticably worse results than the EM algorithm in this test as well. Self-organizing maps and the EM algorithm had similar performance results for this test — though the EM algorithm attained a slightly lower average number of errors, and a lower best error rate.

We were alarmed by the unpredictability of the results of the self-organizing maps. We expected a much more smooth curve, rather than one that varies so much between nearby neighbours. A curve like that seen in Figure E.2 indicates a lack of stability in the results because it indicates a lack of predictability. As a consequence of this, combined with the fact that the EM algorithm attains better overall performance, we can conclude that the EM algorithm seems to have the best performance (under the predefined conditions). The best performance of the EM algorithm is seen with the number of clusters set at 50, but we see interesting results with a number of clusters of 35.

E.4.2 Results with All Features

For this section we present the results for the experiments of Section E.3.2 in graphical form, just as in Section E.4.1. The performance of the autoassociator is listed in Figure E.4, the performance of the self-organizing maps is in Figure E.5, and the performance of the EM algorithm is in Figure E.6.

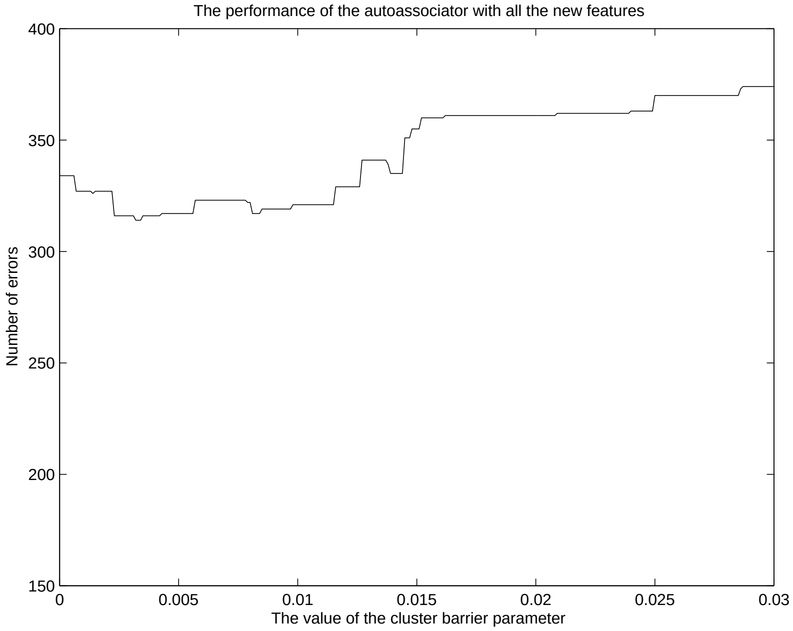

Figure E.4: *Autoassociator varying cluster barrier for all features*

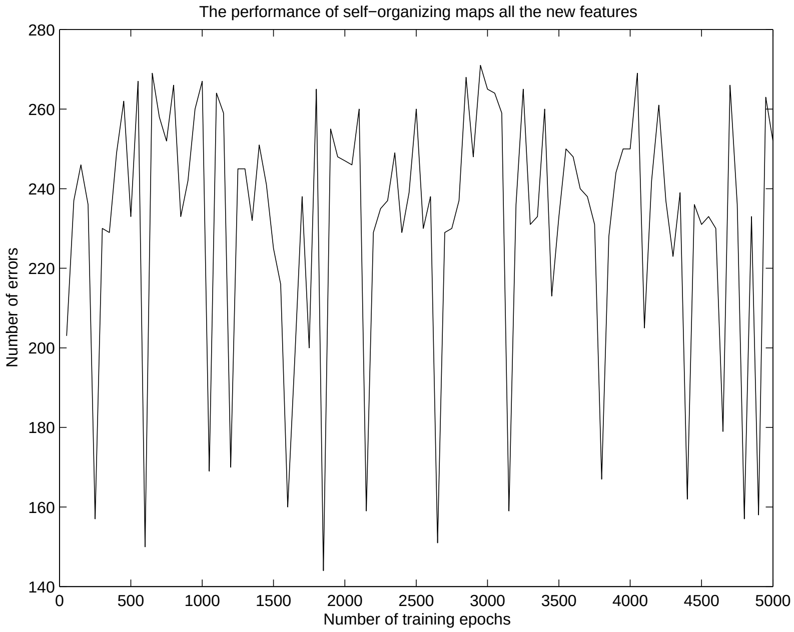

Figure E.5: *Self-organizing maps varying epochs for all features*

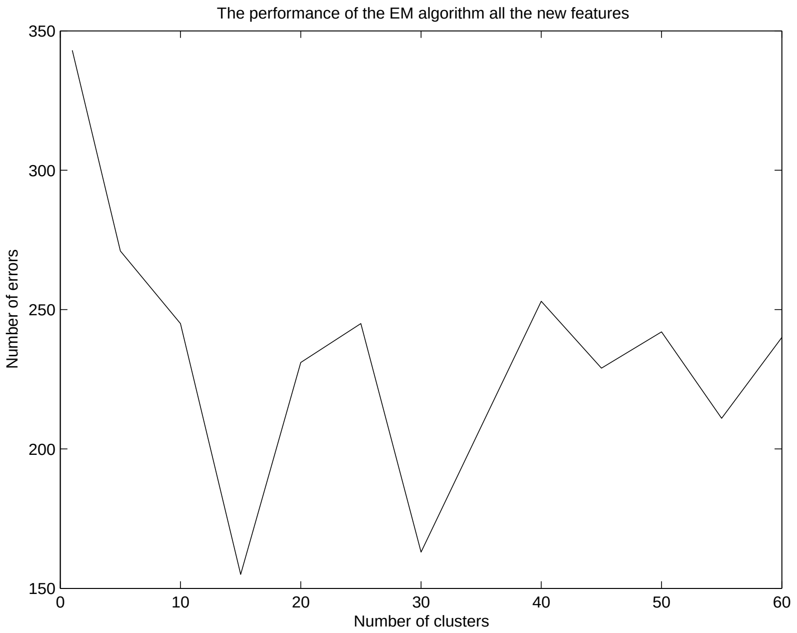

Figure E.6: *EM algorithm varying number of clusters for all features*

As you can see from the graphs, the same sort of trends occur that we saw in the set of graphs for Section E.4.1. When comparing the graph for the autoassociator with the Dain and Cunningham features (Figure E.1) to the graph for the autoassociator with all the features (Figure E.4) we see in the former graph that the autoassociator performs better with larger values for the cluster barrier parameter (best value found: 0.021), whereas for the latter graph, the performance peaks at at smaller values for this parameter (best value found: 0.003).

For the pair of graphs representing the performance of self-organizing maps (Figure E.2 and Figure E.5) we see that the results were both very volatile. Just as with the Dain and Cunningham feature set, the new features feature set produced results that were unpredictable. One important thing to note between these two graphs, though, is that the latter graph clearly shows that self-organizing maps perform better with all the features than with just the Dain and Cunningham features.

For the pair of graphs representing the performance of the EM algorithm (Figure E.3 and Figure E.6) we see that the overall shapes of the graphs are very similar, and, indeed, the performance shown in the two graphs is similar as well. The latter graph shows that the best performance happens at 15 clusters and 30 clusters. The most interesting results are seen at the 30 cluster mark. The results at this point seem to balance clustering errors versus separation errors well.

At this point we'd like to delve a bit deeper into the results to perform more thorough analysis. Some of the best results we've seen are using the EM algorithm with all new features selected and with the number of clusters fixed at 30. With the number of clusters set to 30, the system produced 58 clustering errors and 105 separation errors, for a total of 163 errors.

A significant portion of the errors occur because of the problems of two super-clusters. One super-cluster is composed of two separate super-clusters from the gold standard. The two super-clusters from the gold standard are homogenously composed of `nmap` alerts. These two optimal super-clusters are pushed together by the EM algorithm, and this accounts for 32 of the clustering errors. 43 separation errors occur because a large super-cluster from the gold standard (composed of "BAD-TRAFFIC tcp port 0 traffic" alerts) is split into two large clusters. Another 21 separation errors occur because another super-cluster of the same type of alerts is split from its optimal super-cluster.

From this analysis we can conclude that many of the errors that we've seen in the results are not critical or dangerous. We have seen that some of the errors from the level of clustering have propagated upwards and cause clustering errors in the larger super-clusters where there would otherwise be no errors as well.

E.5 Conclusions

We've seen that each of the algorithms, while varying the chosen parameters for each, produced very characteristic graphs. The results produced by the autoassociator seemed stable but were very poor. The results produced by the self-organizing maps varied significantly, but the algorithm seemed to produce acceptable results on average if all of our new features were used. For the EM algorithm, we found that the algorithm did very well, especially with the number of clusters fixed at 30 or 35. The EM algorithm did quite well regardless of which feature set we used.

We found in our testing that the algorithms tended to perform better with all of the features we constructed for this report, rather than for just the features adapted from the paper by Dain and Cunningham [27].

Accordingly, we recommend using the EM algorithm with all of the new features, fixing 30 to be the number of clusters to produce. We also found, after analyzing the results of this configuration in depth, that most of the errors produced by the system are acceptable.

E.6 Appendix: Gold Standard Super-clusters

This section contains an editted version of the gold standard super-clusters for the incidents.org dataset. This section is editted because it was too long (about 50 pages) to include here otherwise. It has been editted to give an overview of how the gold standard was created.

```
Supercluster 1/25
------------------
Cluster 1/6 (sc:1)
-------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/09-20:51:11.676507 62.13.27.29:0->207.166.33.145:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40
*****R** Seq:0x81F9750 Ack:0x81F9750 Win:0x0 TcpLen:0


[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:12:17.866507 172.20.10.199:0->207.166.119.62:0
TCP TTL:235 TOS:0x0 ID:0 IpLen:20 DgmLen:40
*****R** Seq:0xBDD2D468 Ack:0xBDD2D468 Win:0x0 TcpLen:16


[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/10-01:28:34.556507 62.13.27.29:0->207.166.78.44:0
TCP TTL:234 TOS:0x0 ID:0 IpLen:20 DgmLen:40 DF
*****R** Seq:0x91D8C02 Ack:0x91D8C02 Win:0x0 TcpLen:12


Cluster 2/6 (sc:1)
-------------------
[**][116:46:1] (snort_decoder) WARNING:TCP Data Offset is less than 5!  [**]
11/12-18:38:17.846507 203.80.239.162:0->207.166.182.137:0
TCP TTL:107 TOS:0x0 ID:35119 IpLen:20 DgmLen:48 DF
1*UA**** Seq:0x7930005 Ack:0xD80A04D1 Win:0x64BA TcpLen:0 UrgPtr:0x800
...

Supercluster 2/25
------------------
Cluster 1/6 (sc:2)
-------------------
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:38.316507 170.129.50.120:63362->159.153.199.24:80
```

```
TCP TTL:125 TOS:0x0 ID:38908 IpLen:20 DgmLen:436 DF
***AP*** Seq:0x99AD8FC7 Ack:0x732FBFCD Win:0x4230 TcpLen:20


[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-09:54:47.286507 170.129.50.120:63387->159.153.199.24:80
TCP TTL:125 TOS:0x0 ID:38984 IpLen:20 DgmLen:436 DF
***AP*** Seq:0x99C8B003 Ack:0x733D8EE2 Win:0x4230 TcpLen:20


Cluster 2/6 (sc:2)
-------------------
[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/14-13:03:42.666507 170.129.50.120:63598->64.4.22.250:80
TCP TTL:124 TOS:0x0 ID:56064 IpLen:20 DgmLen:932 DF
***AP*** Seq:0x94851318 Ack:0x9AB18054 Win:0x43E1 TcpLen:20


Cluster 3/6 (sc:2)
-------------------
[**][119:12:1] (http_inspect) APACHE WHITESPACE (TAB) [**]
11/15-02:46:28.446507 170.129.50.120:64749->216.130.211.11:80
TCP TTL:124 TOS:0x0 ID:40697 IpLen:20 DgmLen:1332 DF
***AP*** Seq:0x1601F507 Ack:0xF2FBCCD5 Win:0x2058 TcpLen:20


[**][119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**]
11/15-02:48:02.606507 170.129.50.120:64868->216.130.211.11:80
TCP TTL:124 TOS:0x0 ID:29178 IpLen:20 DgmLen:1332 DF
***AP*** Seq:0x1772D6D5 Ack:0x4E05CFF6 Win:0x2058 TcpLen:20


Cluster 1/5 (sc:3)
-------------------
[**][1:184:4] BACKDOOR Q access [**]
11/14-09:29:14.826507 255.255.255.255:31337->170.129.172.186:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20


[**][1:184:4] BACKDOOR Q access [**]
11/14-09:32:53.016507 255.255.255.255:31337->170.129.132.79:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
```

```
***A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20


[**][1:184:4] BACKDOOR Q access [**]
11/14-09:49:26.156507 255.255.255.255:31337->170.129.129.188:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20
...

Cluster 2/5 (sc:3)
-------------------
[**][1:184:4] BACKDOOR Q access [**]
11/14-23:47:53.416507 255.255.255.255:31337->170.129.19.28:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20


[**][1:184:4] BACKDOOR Q access [**]
11/14-23:55:50.456507 255.255.255.255:31337->170.129.161.133:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq:0x0 Ack:0x0 Win:0x0 TcpLen:20
...

Supercluster 4/25
------------------
Cluster 1/6 (sc:4)
-------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:03.816507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30084 IpLen:20 DgmLen:40
***A**** Seq:0x134 Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:08.786507 61.218.161.202:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30366 IpLen:20 DgmLen:40
***A**** Seq:0x198 Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-10:10:13.826507 61.218.161.210:80->170.129.19.170:80
TCP TTL:48 TOS:0x0 ID:30662 IpLen:20 DgmLen:40
***A**** Seq:0x20A Ack:0x0 Win:0x578 TcpLen:20
...
```

```
Cluster 2/6 (sc:4)
-------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:49.906507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28299 IpLen:20 DgmLen:40
***A**** Seq:0x11B Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:54.916507 61.218.161.202:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28601 IpLen:20 DgmLen:40
***A**** Seq:0x18D Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-12:51:59.916507 61.218.161.210:80->170.129.14.62:80
TCP TTL:48 TOS:0x0 ID:28911 IpLen:20 DgmLen:40
***A**** Seq:0x209 Ack:0x0 Win:0x578 TcpLen:20
...

Supercluster 8/25
------------------
Cluster 1/4 (sc:8)
-------------------
[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:44.536507 61.218.15.126:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:9312 IpLen:20 DgmLen:40
***A**** Seq:0x1BF Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-20:33:54.676507 61.221.88.198:80->170.129.69.49:80
TCP TTL:50 TOS:0x0 ID:10358 IpLen:20 DgmLen:40
***A**** Seq:0x286 Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/14-20:34:00.056507 192.192.171.251:80->170.129.69.49:80
TCP TTL:44 TOS:0x0 ID:10868 IpLen:20 DgmLen:40
***A**** Seq:0x2EA Ack:0x0 Win:0x578 TcpLen:20
...

Cluster 2/4 (sc:8)
-------------------
```

```
[**][1:628:3] SCAN nmap TCP [**]
11/15-15:36:25.236507 192.192.171.251:80->170.129.105.7:80
TCP TTL:43 TOS:0x0 ID:10388 IpLen:20 DgmLen:40
***A**** Seq:0x260 Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/15-15:36:17.916507 61.221.88.198:80->170.129.105.7:80
TCP TTL:49 TOS:0x0 ID:9868 IpLen:20 DgmLen:40
***A**** Seq:0x1FC Ack:0x0 Win:0x578 TcpLen:20


[**][1:628:3] SCAN nmap TCP [**]
11/15-15:36:07.886507 61.218.15.126:80->170.129.105.7:80
TCP TTL:49 TOS:0x0 ID:8842 IpLen:20 DgmLen:40
***A**** Seq:0x134 Ack:0x0 Win:0x578 TcpLen:20
...
Cluster 1/1 (sc:15)
-------------------
[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.096507 170.129.50.120:61121->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22720 IpLen:20 DgmLen:158 DF
***AP*** Seq:0x5A0CA92C Ack:0x53A65413 Win:0x4038 TcpLen:20


[**][1:556:5] P2P Outbound GNUTella client request [**]
11/14-15:43:37.316507 170.129.50.120:61122->24.65.114.32:6003
TCP TTL:123 TOS:0x0 ID:22766 IpLen:20 DgmLen:62 DF
***AP*** Seq:0x5A129557 Ack:0x53A7A3BA Win:0x4038 TcpLen:20


Supercluster 16/25
------------------
Cluster 1/2 (sc:16)
-------------------
[**][1:620:6] SCAN Proxy Port 8080 attempt [**]
11/14-16:00:56.996507 66.159.18.66:43517->170.129.50.120:8080
TCP TTL:53 TOS:0x0 ID:59575 IpLen:20 DgmLen:60 DF
******S* Seq:0xBED8745 Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0


[**][1:618:5] SCAN Squid Proxy attempt [**]
```

```
11/14-16:00:56.996507 66.159.18.66:43518->170.129.50.120:3128
TCP TTL:53 TOS:0x0 ID:50174 IpLen:20 DgmLen:60 DF
******S* Seq:0xBF3AC0C Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:48656370 0 NOP WS:0
...

Cluster 2/2 (sc:16)
-------------------
[**][1:618:5] SCAN Squid Proxy attempt [**]
11/14-23:32:20.916507 66.159.18.49:55991->170.129.50.120:3128
TCP TTL:52 TOS:0x0 ID:16353 IpLen:20 DgmLen:60 DF
******S* Seq:0xB4E1FC5B Ack:0x0 Win:0x16D0 TcpLen:40
TCP Options (5) => MSS:1460 SackOK TS:51364794 0 NOP WS:0
...

Supercluster 21/25
------------------
Cluster 1/2 (sc:21)
-------------------
[**][1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
11/14-16:10:30.806507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:56986 IpLen:20 DgmLen:1420 DF
***A**** Seq:0x8217BFFC Ack:0x90CF9E29 Win:0x16D0 TcpLen:20


[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.566507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46490 IpLen:20 DgmLen:1420 DF
***A**** Seq:0xA074E240 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20


[**][1:648:6] SHELLCODE x86 NOOP [**]
11/14-21:55:36.576507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:46491 IpLen:20 DgmLen:1420 DF
***A**** Seq:0xA074E7A4 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
...

Cluster 2/2 (sc:21)
-------------------
[**][1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
11/14-21:56:03.856507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0x0 ID:48298 IpLen:20 DgmLen:1420 DF
***AP*** Seq:0xA09AF480 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
```

```
[**][1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
11/14-21:56:03.906507 129.118.2.10:57425->170.129.50.120:63414
TCP TTL:51 TOS:0xA0 ID:48302 IpLen:20 DgmLen:1420 DF
***AP*** Seq:0xA09B0A10 Ack:0x90CF9E29 Win:0x16D0 TcpLen:20
...

Supercluster 22/25
------------------
Cluster 1/13 (sc:22)
-------------------
[**][1:524:7] BAD-TRAFFIC tcp port 0 traffic [**]
11/15-07:36:26.406507 211.47.255.24:41866->170.129.195.40:0
TCP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq:0xD8010CF5 Ack:0x0 Win:0x16D0 TcpLen:32
TCP Options (6) => MSS:1460 NOP NOP SackOK NOP WS:0


[**][1:524:7] BAD-TRAFFIC tcp port 0 traffic [**]
11/15-07:36:29.296507 211.47.255.24:41866->170.129.195.40:0
TCP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq:0xD8010CF5 Ack:0x0 Win:0x16D0 TcpLen:32
TCP Options (6) => MSS:1460 NOP NOP SackOK NOP WS:0
...

Cluster 2/13 (sc:22)
-------------------
[**][1:524:7] BAD-TRAFFIC tcp port 0 traffic [**]
11/15-15:09:56.016507 211.47.255.23:47620->170.129.23.96:0
TCP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq:0x88C07AEB Ack:0x0 Win:0x16D0 TcpLen:32
TCP Options (6) => MSS:1460 NOP NOP SackOK NOP WS:0


[**][1:524:7] BAD-TRAFFIC tcp port 0 traffic [**]
11/15-15:10:02.006507 211.47.255.23:47620->170.129.23.96:0
TCP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq:0x88C07AEB Ack:0x0 Win:0x16D0 TcpLen:32
TCP Options (6) => MSS:1460 NOP NOP SackOK NOP WS:0
...

Supercluster 23/25
------------------
```

```
Cluster 1/3 (sc:23)
-------------------
[**][1:527:4] BAD-TRAFFIC same SRC/DST [**]
11/14-22:36:45.306507 170.129.215.99->170.129.215.99
IGMP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28


[**][1:527:4] BAD-TRAFFIC same SRC/DST [**]
11/14-22:36:45.306507 170.129.215.104->170.129.215.104
IGMP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:28
...

Cluster 2/3 (sc:23)
-------------------
[**][1:527:4] BAD-TRAFFIC same SRC/DST [**]
11/16-03:26:16.456507 170.129.71.37->170.129.71.37
IGMP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:28


[**][1:527:4] BAD-TRAFFIC same SRC/DST [**]
11/16-03:26:16.456507 170.129.71.42->170.129.71.42
IGMP TTL:46 TOS:0x0 ID:0 IpLen:20 DgmLen:28
...

Supercluster 24/25
------------------
Cluster 1/3 (sc:24)
-------------------
[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-11:21:09.916507 200.200.200.1->170.129.211.200
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014


[**][1:523:4] BAD-TRAFFIC ip reserved bit set [**]
11/14-14:37:18.296507 200.200.200.1->170.129.2.16
TCP TTL:242 TOS:0x0 ID:0 IpLen:20 DgmLen:40 RB
Frag Offset:0x0864 Frag Size:0x0014
...
```

E.7 Appendix: MSE graph for training data

We include the MSE graph produced by Matlab as an appendix. The data being trained here is the data used to produce Figure E.4, in Section E.4.2. The graph shows 500 training epochs for an autoassociator with 64 hidden units.

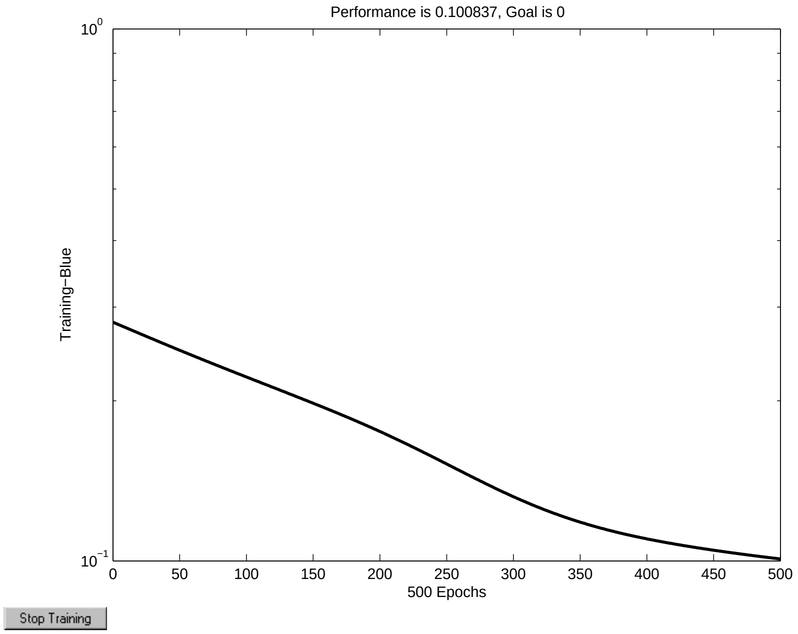

Figure E.7: *The graph produced by Matlab while training the autoassociator*

This page intentionally left blank.

Annex F: Fourth Report

F.1 Introduction

In this report we analyze the results of two experiments. In the first experiment we hope to show that the reconstruction error values produced by the autoassociator change less if we train the autoassociator with a separate training set. Minimizing change in the reconstruction error values is important because it allows the IDS operator to find correlation between different testing sets more easily. The system of super-cluster correlation we presented in the last report requires this trait of stability as well.

In the second experiment we'll evaluate some different scaling methods. In artificial neural networks it is often important to scale data so that computational errors are don't occur in neural network training. In our experiments we will evaluate different scaling methods with respect to system performance.

In our proposal for this research contract, we planned to report on a number of things for this fourth report. Specifically, we said that we would experiment with two-tier feature selection and create new features if the old ones weren't performing, and we proposed to experiment with different data scaling methods. We have completed the former task for the last report and we do not explore it any more here.

We wish to show stability in reconstruction error values because if reconstruction error values for a set of similar alerts are static, or vary little, then alerts are comparable between test sets, which bolsters our super-cluster creation system introduced in the previous report. This is also a property of the autoassociator we've reported to want since creating the Autocorrel I system, but we've never proven that using an unlabelled training dataset produces this desired stability. We show this in Section F.2.1.

The experiments we describe in Section F.2.2 are intended to do two things. The first thing we'd like to show is the importance of scaling the data. We show this by showing how our system performs with unscaled data. We compare the results to the results of our Autocorrel I system. The second thing we'd like to investigate is which scaling method performs best, if there are in fact differences in the performance between the algorithms being considered. If the scaling algorithms we consider perform differently then the old system may have different optimal parameter values, especially for the cluster barrier parameter. We vary this parameter for our experiments because it is the parameter most likely in need of adjustment.

In Section F.2 we discuss the experiments we'll perform, in Section F.3 we discuss the results of the experiments, and in Section F.4 we conclude our results.

F.2 Technical Overview

This section explains the experiments we're going to perform and what we want to learn from these experiments.

F.2.1 Reconstruction Error Stability

First we begin with a more rigourous definition of stability in the context of reconstruction error values changing between experiments. Stability for a particular alert or data item means that for a given training set that the reconstruction error value of that alert doesn't change much if it is part of a very different greater validation set. What this means practically is that the reconstruction error should be mostly independent of the dataset which it's a part of.

We set up an experiment to test the stability of a set alerts. To do this we use the parameters tuned in the previous reports. Namely, we train the autoassociator for 500 epochs and with 64 hidden units. We use the same 10,000 alert training set that we've used in the other reports. We contrast this with the reconstruction error values of a system that doesn't use the training data, a system that both trains on and clusters the validation data.

We should mention that we only use the first part of the system we discussed in the last report because only the first part uses the autoassociator, so only the first part will have reconstruction error values associated with the simulated data.

For the experiment we choose a set of 100 alerts for which we'll report the reconstruction error values. This set of 100 alerts contains 50 `Scan Proxy Port 8080` alerts and 50 `Short UDP Packet` alerts. For this set of 100 alerts we report on the reconstruction error values seen if clustered together with a greater set of 400 other alerts. For one test this set of 400 alerts will be more `Scan Proxy Port 8080` alerts and for another test the additional set of 400 alerts will be composed of different `Short UDP Packet` alerts.

We expect the reconstruction error values of the 100 validation alerts will exhibit less stability if trained with the complete set of alerts they're clustered with than if the training set of 10,000 alerts is used. By changing the distribution of the validation set we hope to show that using the validation set as the training data leads to unstable reconstruction error values.

F.2.2 Scaling Algorithms

To talk about how we'd like to change the way we scale the datasets for input into our system, we need to review what scaling method we've used previously. The system of scaling we used previously linearly scaled data from an input range to the range $[0, 1]$.

Specifically, for every feature in the dataset we determined the range of values for that set. We called the highest value for feature f $high$ and the lowest value low. From these values we computed the linearly scaled value for f of a data item d_i as $(d_i - low)/(high - low)$. For the case when $high = low$, we computed f for d_i as $d_i - low + 0.5$. You can see that with this computation, as long as all the input values are in the interval $[low, high]$ then the linearly mapped values will be in the interval $[0, 1]$.

For our previous system we meant to take these values for $high$ and low from the training set and apply the scaling with these values to all of the validation and testing sets. But because of a bug in our software, we were effectively determining new values for $high$ and low for the validation and testing sets, which happened to have $high$ and low values close to those of the training set. Because of this bug we want to report how our system would perform without the bug. (We chose to keep this bug in the system for quite a while after discovering it because the system seemed to function well enough with it present, and we'd already reported results with it present.)

We also want to try scaling our system to a set of predefined ranges, which was originally suggested by our DRDC liason, Dr. Dondo. We believe this type of scaling is intuitive, so we present results for that change as well. This type of change is a good idea because of the domain we're experimenting in. For IP addresses, as well as all of the other features we've selected for the first level of our Autocorrel system, we know that the addresses are 32 bits by definition in the IP header, so the values of the addresses are always in the range of encoded values $[0, 2^{32} - 1]$. We can assign $low = 0$ and $high = 2^{32} - 1$ and remove the step for determination of $high$ and low altogether when running the scaling algorithm, leaving the system with more predictable sets of values produced. For each of the features we were able to determine a correct range of values from domain literature such as Stevens [2]. We do not need to reproduce the ranges here because the format of the IP, TCP, etc. headers are so well known.

We also wanted to try linearly scaling to the interval $[-1, 1]$ instead of $[0, 1]$ because this range is recommended in some neural network literature (Jain *et al* [47]). To do this we simply took our previously scaled values for feature f at data item d_i, say $scaled([0, 1], d_i^{(f)})$, and performed the following transformation to get $scaled([-1, 1], d_i^{(f)})$: $scaled([-1, 1], d_i^{(f)}) := 2 * scaled([0, 1], d_i^{(f)}) - 1$.

Lastly, we wanted to try a Gaussian method of scaling, as well, that relies on accurate (and Gaussian) distributions in the training data. To compute the Gaussian scaling for $d_i^{(f)}$, we first have to determine $\mu^{(f)}$ and $\sqrt{\omega}^{(f)}$. To do this, we simply used the mean and standard deviation calculations, respectively, of Matlab. Once these values are found, we could compute the new value for $d_i^{(f)}$ as $(d_i^{(f)} - \mu^{(f)})/\sqrt{\omega}^{(f)}$, where the formula includes the unscaled values for $d_i^{(f)}$. This gives the training data a new mean of 0 and a new standard deviation of 1. The values of $\mu^{(f)}$ and $\sqrt{\omega}^{(f)}$ from the training set are used to scale the validation and testing sets.

For completeness we also discuss our results when trying out our system using only unscaled data.

F.3 Experiment Results

In this section we present the results of the experiments we presented in Section F.2.

F.3.1 Reconstruction Error Stability Results

To explore the impact on reconstruction error values we use our system to determine the reconstruction errors on the two datasets outlined in Section F.2.1. Both validation datasets have 500 alerts, but one dataset has 450 `Scan Proxy Port 8080` alerts with only 50 `Short UDP Packet` alerts, and the other dataset of 500 alerts has 450 of the latter type of alert and only 50 of the former. We report on the reconstruction error values for 100 of the 500 alerts from each dataset. The 100 alerts that we report with are the same in both datasets. See Figure F.1 for a graph of the results.

To measure the stability of the reconstruction error values we determined the reconstruction error values for both of the datasets for each different training set, then took the difference of the reconstruction errors for the data items that were in both datasets. The results of this experiment for the autoassociator trained with the 10,000 training set alerts are the solid line in Figure F.1. The results for the autoassociator trained on the validation set are the dotted line in the same figure.

As you can see from the graph the difference in reconstruction error values for the autoassociator trained on the validation set are much larger than those for the autoassociator with the 10,000 alert training set. This is especially true of the `Short UDP Packet` alerts which are represented by the high plateau on the right side of the graph. You can see for the `Scan Proxy Port 8080` alerts that the autoassociator trained with the 10,000 alert training set displays more stable reconstruction error values too. The differences for these alerts are not as dramatic, but it is still clear that the autoassociator trained with 10,000 alerts is uniformly more stable than if trained on the validation data.

From the previous experiment we can conclude that the reconstruction errors values produced by an autoassociator trained on the 10,000 alert training set are much more stable than if the autoassociator is not trained on this data and instead trained on the validation data as we've previously experimented with.

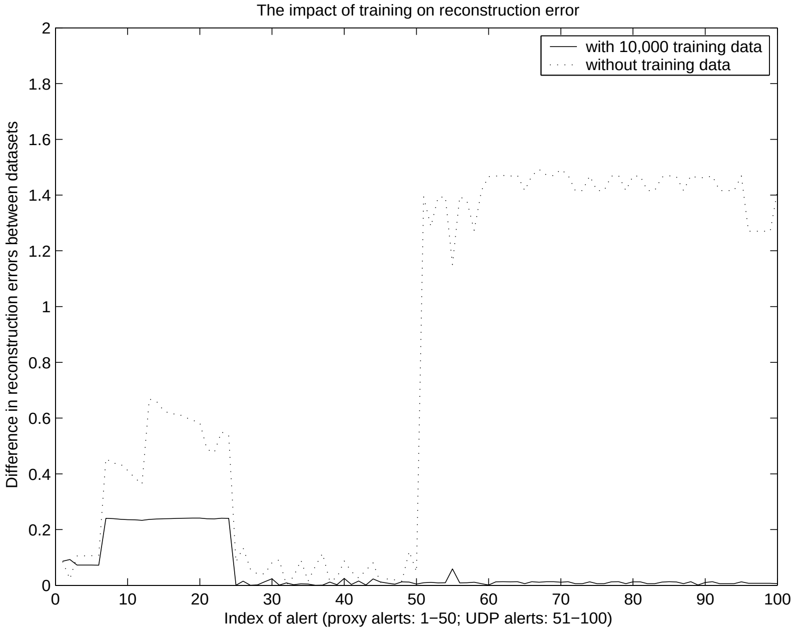

Figure F.1: *The impact of using a training set on the stability of reconstruction error values*

F.3.2 Scaling Algorithm Results

For the results of this section we use the five 100 alert validation clusterings from the *incidents.org* [43] that we used to report with in the previous report. For this report we determine the performance of our system on each of these five validation sets, and we sum the errors produced by each system to gain a better view of how the system performs. We report the number of errors as a percentage of the total possible number of errors, 500. The system with the lowest percentage of errors is considered the best.

As previously mentioned, we expected that we'd have to adjust the cluster barrier parameter for this experiment by necessity because this parameter is directly dependent on how the data is encoded and scaled. We do this graphically by showing the percentage of total errors, clustering errors and separations errors (see the last report for definitions of these types of errors) plotted against the value for the cluster barrier parameter for the system in question.

Figure F.2 shows how our previous system performs (with the bug where the training set scaling data wasn't used to scale the testing sets). Figure F.3 shows the exact same system as Figure F.2 except with this bug removed. As you can see from the these graphs, the best performance is attained with cluster barrier parameters 0.0017 and 0.0015 respectively. These optimal values are fairly close to the value used in our

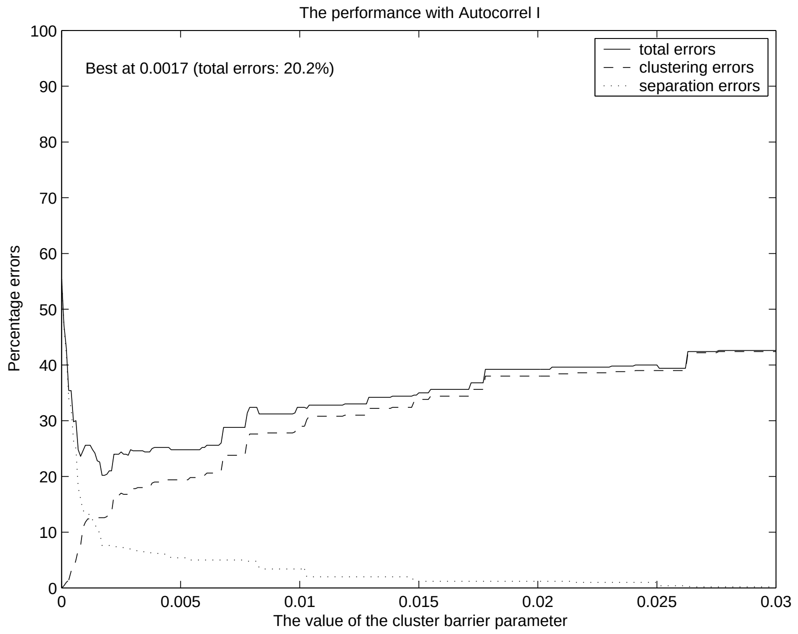

Figure F.2: *The performance of Autocorrel I system, varying the cluster barrier parameter*

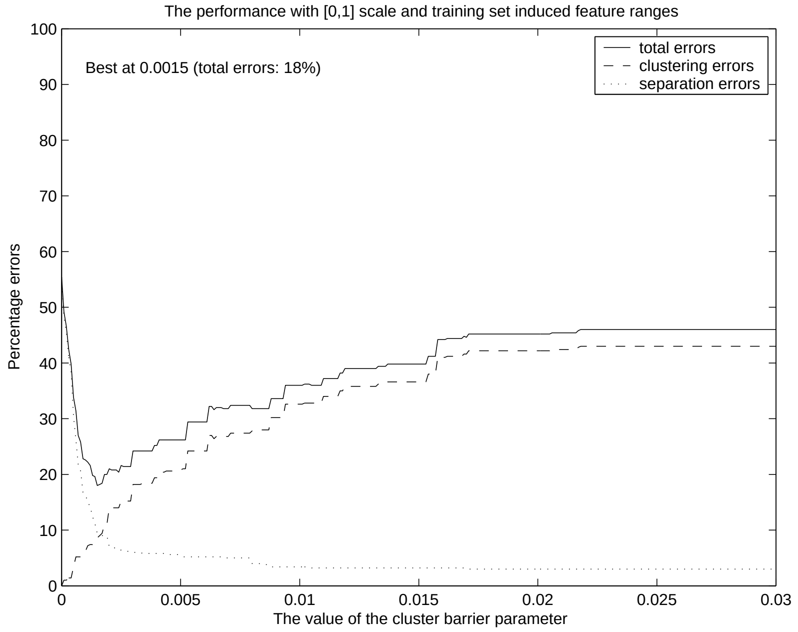

Figure F.3: *The performance of the system using a [0,1] scale with the training set to determine ranges*

Autocorrel I system, 0.0025. It is interesting to note that correcting the bug has given better optimal performance, but the performance in the second graph also worsens quicker, implying that it's more important to get the cluster barrier correct, which of course is impossible in a deployed system where there is not optimal correlation sets to reference.

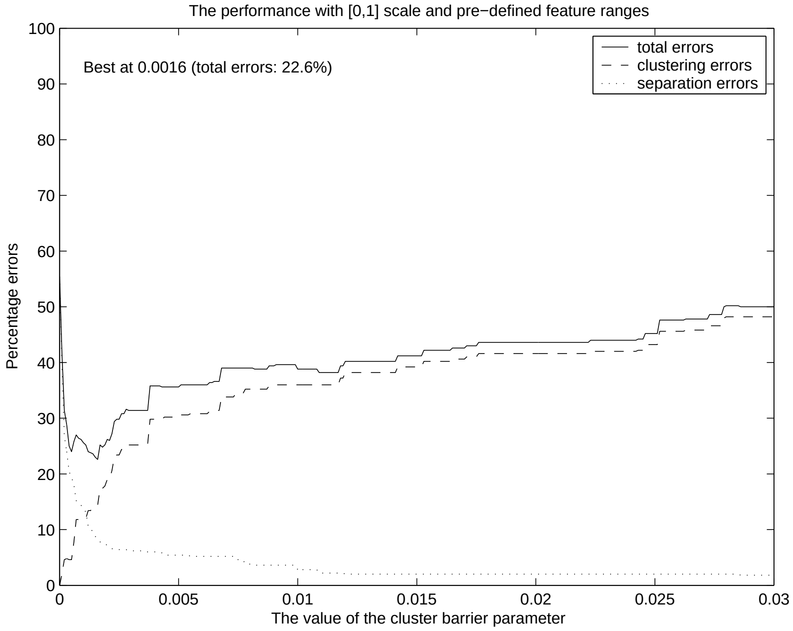

Figure F.4: *The performance of the system using a [0,1] scale and predefined feature ranges*

The last variation of a system scaling to the interval $[0, 1]$ is seen in Figure F.4. For this graph we used the predefined feature ranges we discussed in Section F.2.2. You can see from the graph that this system performs noticably worse than the other systems linearly scaling to this interval.

Figure F.5 shows the performance for the system scaling to the $[-1, 1]$ interval using *high* and *low* values from the training set. Figure F.6 shows the performance for this same interval but with the predefined *high* and *low* feature range values. Figure F.6 showed the lowest error rate of any of the systems we examined, and Figure F.5 was so close to this that the difference isn't statistically significant. The two graphs show essentially the same performance, and they show this with almost the same value for the cluster barrier parameter, 0.0028 and 0.0027.

Figure F.7 shows how well our system performed with the Gaussian scaling method from Section F.2.2. As you can see from this graph, the best results are not as good as the other methods, but it seems that the performance was less affected by

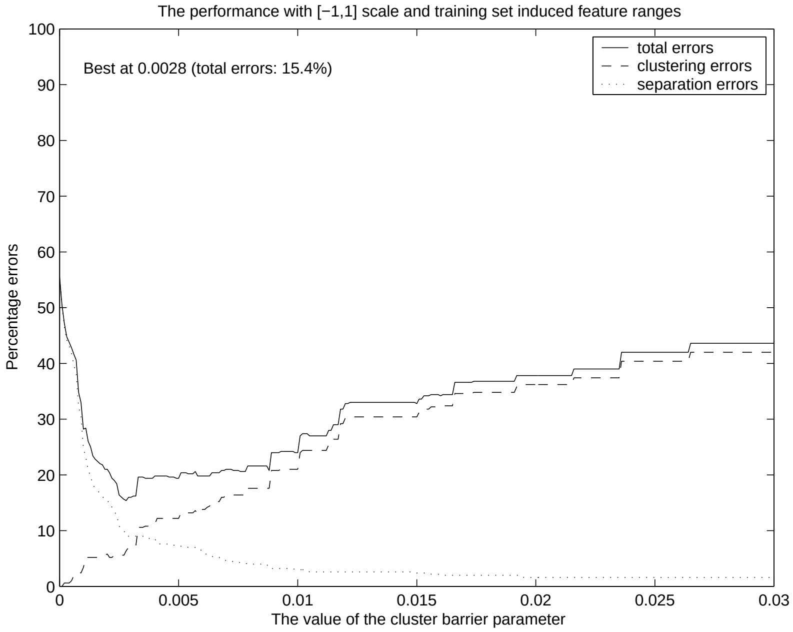

Figure F.5: *The performance of the system using a [-1,1] scale with the training set to determine ranges*

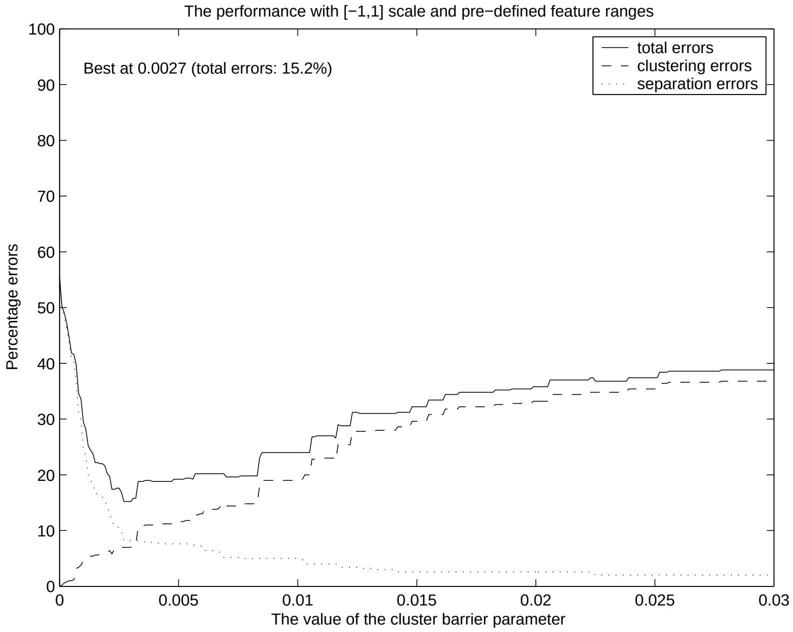

Figure F.6: *The performance of the system using a [-1,1] scale and predefined feature ranges*

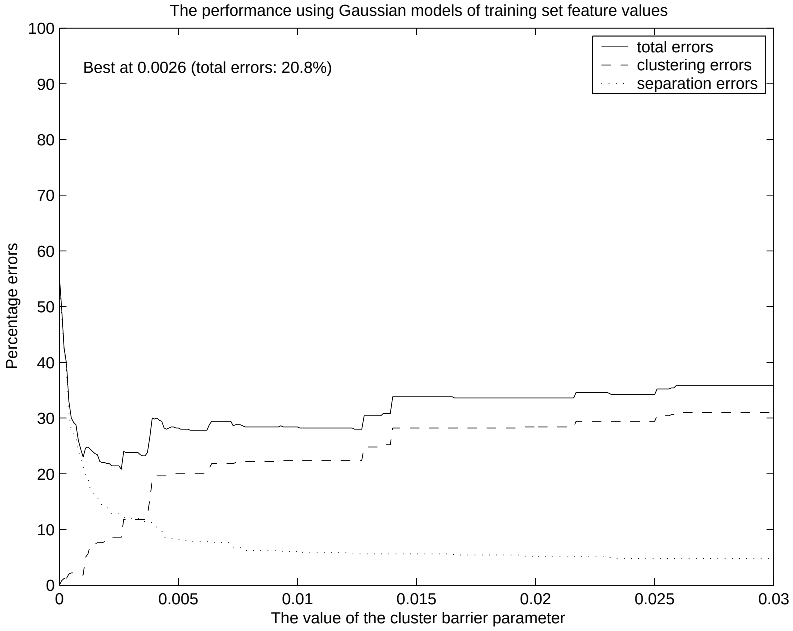

Figure F.7: *The performance of the system using Gaussian models for features determined from the training set*

the value for the cluster barrier parameter, because larger values for this parameter didn't produce significantly worse results, unlike in the other systems. We found this interesting, but we do not choose this system of scaling because we plan to fix the cluster barrier parameter at a lower value anyway.

Lastly we tried running our system on unscaled datasets. The results were so poor (55% error rate in the best system) that we do not present them in graphical form here. The results hardly varied at all with the cluster barrier parameter. We concluded with this result that selection of which scaling method to use is important. We also conclude that linearly scaling to the interval $[-1, 1]$ (using either of the methods we presented results for) at a cluster barrier parameter value of about 0.0025 produces the best results.

F.4 Conclusions

The results from the previous section are very clear. In Section F.3.1 we learned that for stability in the reconstruction error values we must train with the 10,000 alert training set. In Section F.3.2 we learned that scaling the data is indeed important for the autoassociator and that the choice of scaling algorithm is also important.

For the choice of the scaling algorithm, our tests indicate that either one of the algorithms we presented that linearly scales to the $[-1, 1]$ interval produces the best results. We also found that 0.0025 is probably a good choice for the cluster barrier parameter for this scaling method.

F.5 Appendix: MSE graph for training data

We include the MSE graph produced by Matlab as an appendix. The data being trained here is scaled with the best performing algorithm we presented here (the predefined input range algorithm) to the interval $[-1, 1]$. The graph shows 500 training epochs for an autoassociator with 64 hidden units.

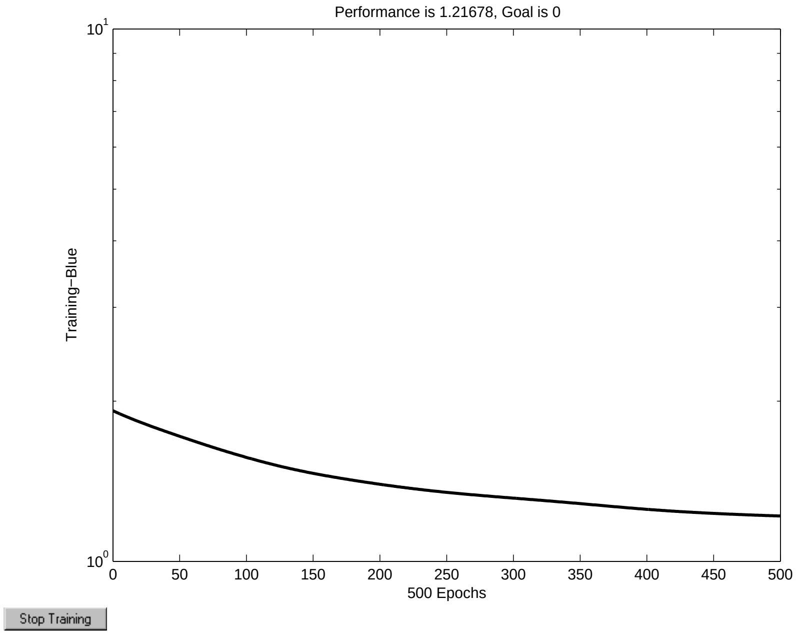

Figure F.8: *The graph produced by Matlab while training the autoassociator*

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

School of Information Technology and Engineering
University of Ottawa
Ottawa Ontario

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Autocorrel II: Unsupervised Network Event Correlation Using Neural Networks

4. AUTHORS (last name, first name, middle initial)

Japkowicz, Nathalie ; Smith, Reuben

5. DATE OF PUBLICATION (month and year of publication of document)

March 2005

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).

188

6b. NO. OF REFS (total cited in document)

52

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

Contract Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

DEFENCE R&D CANADA - OTTAWA
3701 Carling Avenue, Ottawa, Ontario, K1A 0Z4

9a. PROJECT NO. (the applicable research and development project number under which the document was written. Specify whether project).

15BF29

9b. GRANT OR CONTRACT NO. (if appropriate, the applicable number under which the document was written).

W7714-04-09128

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)

Defence R&D Canada – Ottawa CR 2005-155

10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Network event correlation is the process where relationships between intrusion detection system alerts or other network events are discovered and reported. An accurate network event correlation system can enable the intrusion detection analyst to find important events more easily. We present a system that uses unsupervised machine learning algorithms to create an effective and maintenance-free way to do network event correlation. The system uses the autoassociator, a type of neural network architecture, to find the relationships between related network events hidden in a collection of unrelated data. We demonstrate our system using intrusion alerts generated by a Snort intrusion detection system and discuss the overall performance of our system.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Neural Network, Intrusion Detection System, Network Event Correlation, Alert Correlation, Autoassociator

Defence R&D Canada – Ottawa DOCUMENT SUBMISSION RECORD

Revised 02/05/27

Work Unit # 15BF29 DRP # CR 2005-155

This form is to accompany all documents to be reviewed by the Document Review Panel (DRP)

TITLE & TITLE CLASSIFICATION Enter The title followed by classification in the form (U) for UNCLASSIFIED, (C) for CONFIDENTIAL, (S) for SECRET, and (TS) for TOP SECRET. For classified titles, page classification must also be changed in Header/Footer.
Autocorrel II: Unsupervised Network Event Correlation Using Neural Networks (U)

AUTHOR(S): Japkowicz, Nathalie ; Smith, Reuben

DOCUMENT CLASSIFICATION/WARNING TERMS: (U)

RESPONSIBLE SECTION: INFORMATION OPERATIONS

DOCUMENT INTENDED FOR:

□ DRDC Ottawa TECHNICAL REPORT

□ DRDC Ottawa TECHNICAL MEMORANDUM

□ DRDC Ottawa TECHNICAL NOTE
(for dist. outside DRDC Ottawa)

□ PAPER FOR OPEN LITERATURE
☒ CONTRACTOR REPORT (Name of Company)
School of Information Technology and Engineering

□ CRC REPORT
□ CRC TECHNICAL NOTE

□ ABSTRACT

□ ORAL PRESENTATION
□ WORKING PAPER FOR TTCP
□ WORKING PAPER FOR NATO

Use this space to identify the following:

1. Distribution of document to any agency not included on Section Standard Distribution List (name agency)
2. Name the publication to be printed in or location of the presentation Date

Each signature identifies that the attached document, forms, and distribution list are accurate and complete and ready for the next stage of review.

AUTHOR/SA: ____________________ **DATE** __________
REVIEWER: ____________________ **DATE** __________
EDITOR: ____________________ **DATE** __________
SECTION HEAD: ____________________ **DATE** __________
DRP REP: ____________________ **DATE** __________
DRP CHAIR: ____________________ **DATE** __________