Clustering using an Autoassociator: A case study in Network Event Correlation

Reuben Smith and Nathalie Japkowicz
University of Ottawa, School of Information Technology and Engineering
Ottawa ON Canada K1N 6N5
{nat,rsmith}@uottawa.ca

Maxwell Dondo
Defence R&D Ottawa
Ottawa ON Canada K1A 0Z4
maxwell.dondo@drdc-rddc.gc.ca

Abstract

An autoassociator is a feedforward neural network that has the same number of input and output units. The goal of the autoassociator is very simple; to reconstruct its input at the output layer. Despite their simplicity, autoassociators have previously been shown to be quite successful on the task of Novelty Detection applied to industrial and military domains. The purpose of this paper is to test their utility on the more general task of clustering. In particular, we apply a clustering version of the autoassociator to the domain of Network Event Correlation. The results suggest that autoassociators are indeed useful as clustering systems. They were able to successfully correlate similar types of network alerts and have the added advantage of being fast once trained, a crucial feature when used for Network Event Correlation.

1 Introduction

An autoassociator [1, 2] is a very simple feedforward network whose goal is to reproduce its input at the output layer. More specifically, an input vector $\mathbf{x}$, is fed through an autoassociator through connection weights $\mathbf{w}$. The objective is for the autoassociator to reproduce the vector $\mathbf{x}$ at the output.

Autoassociators were previously used successfully for Novelty detection [2, 3]. The approach is based on the fact that the reproduction of the input layer is not perfect, and a measure of this imperfection, the reconstruction error, can be used to distinguish inputs that deviate from the norm. In more detail, at training time, the autoassociator was only fed positive instances of the concept at hand. The expectation was that testing examples of the class the autoassociator was trained on (positive examples) would be well reconstructed (because the autoassociator already knows how to reconstruct examples of this kind) while negative examples would not be (since examples of this kind were never encountered by the system). This expectation proved correct and the autoassociator produced very good results.

In our previous work [2, 3], we noticed but did not exploit, an interesting feature of the autoassociator used as a novelty detector; in addition to separating all the positive from all the negative examples, with respect to the reconstruction error, they produced subgroupings of the input. This seemed to suggest that the network may be performing some sort of clustering and we wondered whether such clustering could be of any benefit.

In order to test this hypothesis, we applied a clustering version of the autoassociator to the difficult problem of Network Event Correlation. The autoassociator was used in a different manner than previously since this time, instead of being trained only on a subset of the data (the positive class), it was trained on all the available data (which is considered multi-class).

The remainder of this paper is organized in 5 sections. Section 2 introduces the domain of Intrusion Detection, focusing particularly on the issue of Network Event Correlation. Section 3 describes the architecture of the autoassociator used in our work and discusses its training parameters. Section 4 discusses the evaluation measures we devised to test our systems. Section 5 presents our results along with a discussion of their practical significance. Section 6 concludes the study.

2 Intrusion Detection and Network Event Correlation

Intrusion detection analysts are tasked with detecting attacks on their network. To do so, they use Intrusion Detection Systems (IDSs) which alert on unacceptable or suspicious events. Intrusion Detection Analysts are often overwhelmed by a multitude of alerts that they receive on a daily basis. The majority of these alerts are not new, but the analyst must go through each one of them to determine the threat each one poses. In some cases, analysts use multiple IDS rules to classify alerts or use some of the generic correlation tools that come bundled together with IDSs.

The majority of the existing correlation tools, however, use simplistic approaches to correlate attacks. For example, Shadow [4] and ACID [5], use the IP addresses to correlate attacks. However, it is known that IP addresses may be spoofed, therefore using IP addresses alone does not provide a sufficient measure to classify the threat possed by an alert. Work reported by Haines *et al* [6], details some of the common correlation tools and approaches. The majority of these approaches take one alert metric at a time to correlate with other possible attacks. More sophisticated approaches use statistical methods on multiple alert metrics [7]. How-

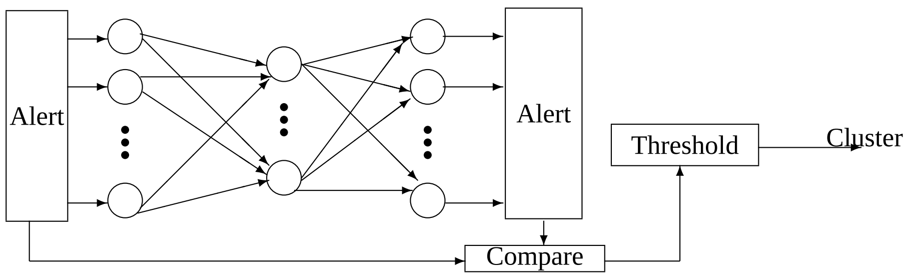

Figure 1. Autoassociation model.

ever, statistical approaches require assumptions to be made on the data beforehand.

IDS researchers and developers are actively working on new and better methods to implement automated alert correlation from both homogeneous and heterogeneous sensors. Current work on IDS reporting standards [6–8] will facilitate the sharing of alert information within the IDS community. We, therefore model our approach as closely as possible towards a model that would be capable of receiving and processing alert content from different homogeneous and heterogeneous IDS sensors. However, in this first generation model, we extract alert features that are common to many IDS sensors, while using Snort [9] based alerts. Future work will harden this approach by looking at a broader spectrum of IDS sensors as opposed to just considering one. This includes any standards that IETF [8] might have developed at that time.

In this work, we extract variables information from the alert content. This consists of the TCP/IP header information associated with the alert. To maximize correlation and to ensure that all the important factors of a possible attack are taken into consideration, as much information as possible is extracted from the alert content. Some attributes of the alerts were dropped for various reasons; for example, the source IP address was dropped because this may give some misleading clusters in cases where the IP address is spoofed; like in DoS attacks (for example). We also felt that the variable IP addresses on different alerts could be used as a strong determining factor in alert clustering, when we know that different alerts may originate from the same IP address. So it was our contention that the remaining variables should be able to identify and correlate any attack based on the remaining input variables.

The alert variable input vector $\mathbf{x}$, extracted from the TCP/IP content of the alert, is independent of the sensor used for the IDS. The same information may be equally extracted from any type of IDS sensor which gives similar amount of data in its alert content.

3 Details of our Autoassociation Scheme

To address the various shortcomings of the network correlation tools currently in use (i.e. information loss and lack of speed), our work undertakes to design an autoassociation-based event correlation engine that takes into account the majority of the alert attributes. We have incorporated a neural network whose speed is excellent for near real-time applications. A reduced dimension clustering algorithm ensures that very little additional computational effort is expended in performing the final decision making process. Neural networks are also adaptive and do not lose information as with classical correlation techniques that use data stored in TCP Quad format [10] and perform a multi-stage query on reduced information or data.

In our approach, all previously seen or collected attack data is used to train an ANN without prior analysis by experts. Once trained, this information is retained in the ANN as connection weights and does not need as much storage as the original alert data. In more detail, a three layer fully connected feedforward ANN with N input nodes, N output nodes and $J < N$ hidden nodes (unoptimised) was used. The output at node i of each layer is $y_i = f(\sum_{k=1}^{K_i} w_{ik} y_k)$, where y_i is the output of neuron i after receiving K_i signals from the neurons of the preceding layer, and $y_i = x_i$ for the input layer.

The network is trained using the error-backpropagation algorithm with the objective of reconstructing the input space at the output. The training objective is to minimize the square errors $E = \sum_{i=1}^{n} ||x_i - y_i||^2$. The ANN weights are iteratively updated as $\mathbf{w}^2 = \mathbf{w}^1 + \eta \Delta \mathbf{w}^1$, until convergence is achieved, where η is the training constant. Once training has been completed, a threshold level is used to compare with the resultant reconstruction error $e_i = ||x_i - y_i||$ for each input vector x_i during ANN recall. This is illustrated in Fig. 1. The result of the clustering is expected to be multiple alert groups that have similar or closely similar attributes.

4 Experimentation Effectiveness Measures

To analyse the performance of our approach and decide whether or not the clustering version of the autoassociator is indeed worthwhile, we have developed measures of success criteria. Our approach will take raw alert data, and use part of it for training. Another part of the data is used for testing the trained autoassociator. We expect to get crisp clusters of similar, closely similar and identical alerts.

Since this is an experiment, errors are inevitable. To understand and analyse the outcome of our experiments, we have used basic performance measures to characterise the effectiveness of the system.

If our approach puts $\mathbf{B}$ alerts into a cluster, and we determine that $\mathbf{A}$ alerts do not belong to this cluster, then the alert placement precision (APP) for this cluster is $\frac{\mathbf{B}-\mathbf{A}}{\mathbf{B}}$. In a similar manner, if $\mathbf{C}$ is the total number of alerts to be clustered, and $\mathbf{D}$ is the number of alerts that were misclustered, then the overall alert placement precision (OAPP) would be $\frac{\mathbf{C}-\mathbf{D}}{\mathbf{C}}$.

If our test data has $\mathbf{E}$ alert types, we expect to get $\mathbf{E}$ crisp clusters for each alert type. Now, if $\mathbf{F}$ alert types don't have clusters of their own, then the cluster placement precision (CPP) is $\frac{\mathbf{E}-\mathbf{F}}{\mathbf{E}}$.

5 Results

The model was tested in two steps. First, the model was tested with labeled DARPA [11] alerts. This portion served as a way to validate the ANN model. The main part of the model testing were carried out with unknown alerts from `www.incidents.org`. In each case, $10\,000$ alerts were used for training. An autoassociator with 42 inputs, 10 hidden layers and 42 outputs was trained, using the error backpropagation algorithm with a training constant η of 0.4. Training was initially fixed to $5\ 000$ epochs, but could be changed if desired.

5.1 Labeled DARPA Alerts

The ANN was trained in $90\,000$ epochs (an MSE of 0.003) using $20\,000$ alerts generated from *week 5* of the 1999 DARPA IDS Evaluation data set [11]. Forty-eight labeled alerts not used in the training were used to test the ANN. The reconstruction errors for the individual alerts are plotted for each alert as shown in Figure 2.

The test alerts represented six attack categories. We expected to get six crisp clusters of these 48 alerts; namely *Telnet Access, DOS Winnuke, FIN Scan, Tiny Fragments, Splice attack, and DOS Land attack.* Our experiment produced six (CPP $= 100\%$) clusters as indicated in Figure 2. However, our model correctly clustered 45 of these alerts into six clusters (OAPP $= 93.75\%$). Three alerts were placed into a cluster (cluster 2) where they shouldn't be (APP $= 30\%$). This translated to 1 erroneous cluster, and three misplaced alerts. As shown in Figure 2, three *DOS Winnuke* alerts were wrongly placed in cluster 2 (*Telnet Access*).

To explain the three stray *DOS Winnuke* alerts in cluster 2 instead of cluster 4 where the other 17 appeared, we compared the values of the input attributes to see where there are significant differences with alerts in cluster 4. Unlike alerts in cluster 2, all alerts in cluster 4 have the TCP FIN flag set. In addition, the alerts in each cluster use different values

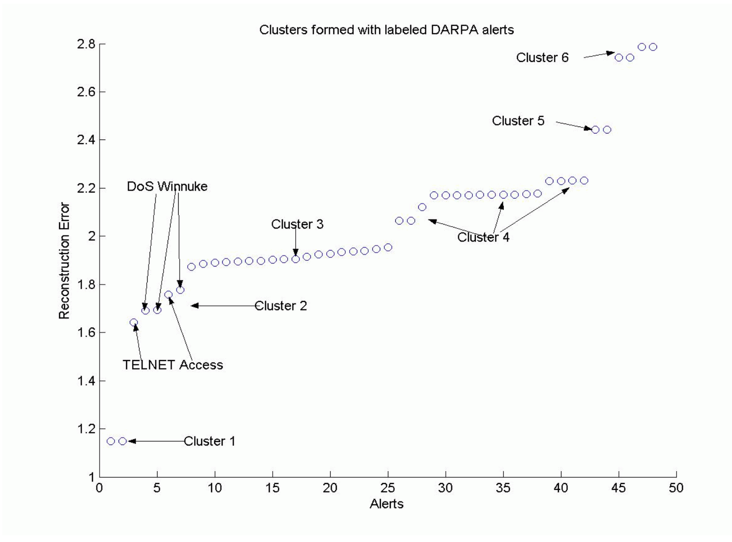

Figure 2. Clusters generated by labeled DARPA alerts.

Table 1. DARPA alert clusters.

Alert	Alerts in cluster					
	1	2	3	4	5	6
Splice attack	2	0	0	0	0	0
Telnet Access	0	2	0	0	0	0
FIN Scan	0	0	18	0	0	0
DOS Winnuke	0	3	0	17	0	0
DOS Land attack	0	0	0	0	2	0
Tiny Fragments	0	0	0	0	0	4
Clustering Error (1-APP)	0	60%	0	0	0	0

of the TCP urgent pointer; `0x31` for cluster 2 and `0xf5` for cluster 3. Despite these minor attribute differences, we expected the ANN to be able to "see through" these differences and still cluster the alerts in one cluster. After all, the *Telnet Access* attributes are different from the *DOS Winnuke* alerts.

The clustering results are summarised in Table 1. The table shows the six clusters formed and the individual alerts that fall into each cluster. It also shows the clustering errors for each of the six clusters. The table also shows the number of alert types that were erroneously clustered, e.g. the *DOS Winnuke* alert that was clustered into clusters 2 and 4.

The labeled DARPA data were clustered with very good results; a 100% CPP and a 93.75% OAPP. We conclude that the ANN clustering was successful in placing alerts into relevant clusters. Only one cluster contained erroneously placed alerts. These statistics were very encouraging and we decided to take a close look at unlabeled alerts that are available in the wild (an unsimulated environment).

5.2 Unlabeled `incidents.org` Alerts

The approach was tested with a sample of 514 Snort alerts from `www.incidents.org`. A total of $20\,000$ alerts were used for training in $80\,000$ epochs until the MSE was 0.05. Similarly matching alerts were grouped into the same

Table 2. A sample of cluster 4 alerts.

```
[**][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
[Xref=> http://www.whitehats.com/info/IDS28]

[**][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
[Xref=> http://www.whitehats.com/info/IDS28]

[**][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
[Xref=> http://www.whitehats.com/info/IDS28]

[**][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
[Xref=> http://www.whitehats.com/info/IDS28]
```

clusters. In Table 2, we show an example of *nmap scan* alert clustering produced by our approach. These closely related alerts belong to one cluster. As expected, the clustered alerts have closely matching attributes as defined earlier. We also note that the clustered alerts display other similar attributes that were not used in the clustering algorithm as variables; namely source and destination IP addresses.

The reconstruction errors plot of the individual alerts are shown in Figure 3. From Figure 3, we note that there are no clear cluster boundaries in some cases. It is evident that there are some cluster overlaps. We decided to carry out

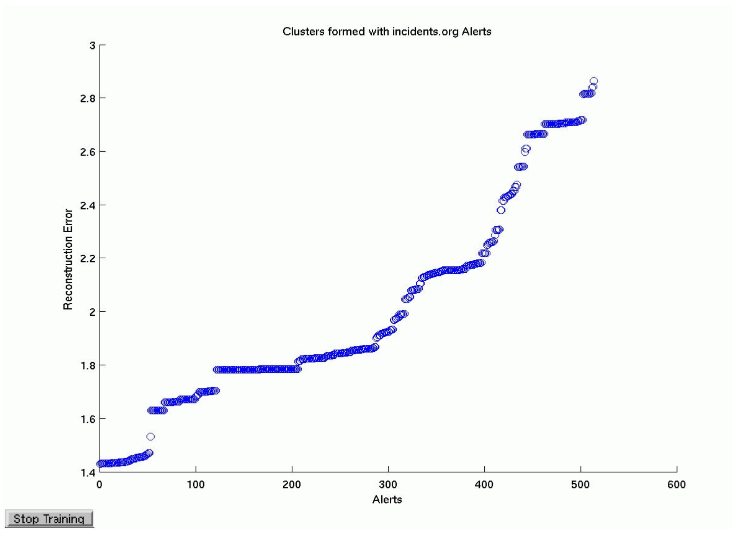

Figure 3. Clusters generated by unlabeled alerts from `incidents.org`.

the analysis of these results in two stages; namely regular clustering and variable clustering by inspection.

5.2.1 Regular Clustering

This approach uses a regular threshold spacing between different clusters. The clusters are separated by a fixed change of 0.025 in reconstruction errors. The results are summarised in Table 3, where the ND (last column) represents an error value that cannot be determined because the clustering is not very clear. The bold figures represents the alert (column) that owns the cluster (row). The error value in the last column represents the error associated with placing alerts into each cluster ($1-$ APP).

Using the regular clustering approach, we clustered the 514 test alerts into 27 clusters. Eighteen clusters accurately placed alerts into relevant clusters without errors (APP $=$ 100%); although some alert groups were clustered into two or more clusters. Seven alert groups did not have clusters of their own. Twenty-one clusters had an APP of 85% or better. The OAPP for this approach was 82% and the CPP was 59%.

From these results and from Figure 3, some clusters have overlaps that cannot be separated by a fixed threshold value. The worst affected cluster is 12. Other overlaps are characterised by low APP values as in clusters 6, 14, 18, and 19. We also have seven alerts that span two or more clusters, and seven alerts that don't have clusters of their own. The possible sources of these errors will be presented in Section 5.3.

Table 3. Alert placement matrix for regular clustering.

Cluster	Alerts																	
	NULL SCAN	TCP Port 0 Traffic	Squid Proxy SCAN	BACKDOOR Q Access	TCP nmap SCAN	Proxy Port SCAN	Short UDP packet	ICMP ISS Pinger	TCP Data Offset	ICMP Superscan	SOCKS Proxy SCAN	Land Attack	GNUTella Outbound	HTTP INSPECT	SHELLCODE x86	IP Reserved bit	TCP Header length	Error (1-APP)
1	**52**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	**14**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	**50**	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6%
5	0	0	0	**85**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	**75**	1	4	0	0	0	0	0	0	0	0	0	0	6%
7	0	0	0	0	**14**	0	1	3	0	0	0	0	0	0	0	0	0	22%
8	0	0	0	0	**12**	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	**5**	0	0	0	0	0	0	0	0	16%
10	0	0	0	0	0	0	0	0	**9**	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0
12	0	0	0	0	1	1	0	0	5	16	14	17	6	3	0	0	0	ND
13	0	0	0	0	0	0	0	0	0	0	**5**	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	2	0	0	0	**6**	0	0	0	25%
15	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	**5**	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0
18	0	0	0	0	0	0	0	0	**9**	0	0	0	0	4	0	0	0	30%
19	0	0	0	0	0	0	0	0	1	0	0	0	0	**2**	0	0	0	33%
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**7**	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**18**	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**40**	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**8**	0	1	11%
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0

5.2.2 Variable Clustering by Inspection

Looking at Figure 3, we were also able to cluster the alerts through inspection. This resulted in clusters with no fixed threshold spacing. The threshold value varied form cluster to cluster. However, there were still some overlaps in the clusters.

In Table 4, we show that we were able to cluster the 514 alerts into fifteen clusters. The clusters had varying clustering successes. We were able to place 477 alerts into thirteen clusters with an APP of 80% or better. This translates to an OAPP 92%. Clusters 6, 10 and 11 had low APP values.

Four of the seventeen alert groups did not have clusters of their own (CPP $= 76\%$). There were only two alert groups that fell into two different clusters, namely *TCP data offset* (clusters 5 and 11) and *Shellcode x86*(clusters 12 and 14). Table 5 shows the summary of the two methods. The variable clustering method is a marked improvement from the fixed threshold clustering. With varying success, our autoassociation model was able to differentiate between subtle differences in different alerts and cluster them accordingly.

Table 5. Clustering effectiveness for the incidents.org alerts

Clustering Method	Group overlaps	OAPP	CPP
Regular clustering	8	82%	59%
Variable Clustering	2	92%	76%

5.2.3 Cluster Grouping

The results also show that some alerts can be grouped together because of the similarities in their signatures. To further assist an analyst with making crucial decisions, it is possible to group the clusters based on their similarities and the analyst's preference. For example, the six *scan* alerts can be grouped into one. This would reduce the number of clusters to eleven.

In a similar manner, all alerts spanning more than one cluster can be grouped together to form one *super-cluster*. For example, clusters 1,4,6, and 8 could be grouped into *super-cluster* A, and clusters 5 and 11 into *super-cluster* B, and clusters 12 and 14 into *super-cluster* C. Depending on the analyst's preference, this grouping may be different, thus we suggest that a configurable decision visualisation engine be coupled to our autoassociation model. This reduces the number of alert categories the analyst has to deal with.

5.3 Practical Impact of Clustering Errors

While the clustering process was overally successful, there were some clustering errors encountered. We will use the variable clustering results of Figure 4 to explain the varying levels errors.

Closely following the six perfect clusters (1, 3, 7, 8, 12, and 15), we had the following finer grained cluster divisions:

Table 4. Alert placement matrix for variable clustering.

Cluster	Alerts																	
	NULL SCAN	TCP Port 0 Traffic	Squid Proxy SCAN	BACKDOOR Q Access	TCP nmap SCAN	Proxy Port SCAN	Short UDP packet	ICMP ISS Pinger	TCP Data Offset	ICMP Superscan	SOCKS Proxy SCAN	Land Attack	GNUTella Outbound	HTTP INSPECT	SHELLCODE x86	IP Reserved bit	TCP Header length	Error (1-APP)
1	**53**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	**64**	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5%
3	0	0	0	**85**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**101**	1	5	3	0	0	0	0	0	0	0	0	0	8%
5	0	0	1	0	0	0	0	0	**16**	0	0	0	0	0	0	0	0	5%
6	0	0	0	0	1	1	0	0	4	**16**	1	0	0	0	0	0	0	30%
7	0	0	0	0	0	0	0	0	0	0	0	**17**	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	**7**	0	0	0	0	0
9	0	0	0	0	0	0	0	0	1	0	**18**	0	0	3	0	0	0	18%
10	0	0	0	0	0	0	0	0	1	2	0	0	0	**13**	0	0	0	20%
11	0	0	0	0	0	0	0	0	**10**	0	0	0	0	6	0	0	0	37%
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**8**	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	**18**	0	10%
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**48**	0	1	2%
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**3**	0
Error(1-APP)	0	0	ND	0	1%	ND	ND	ND	18%	11%	5%	0	0	45%	0	0	25%	

1. The emergence of two or more distinct clusters for a given alert group as exemplified by the *Shellcode x86* alert group (clusters 12 and 14), and the *TCP data offset* alert group(clusters 5 and 11).

2. The "spilling over" of some of the alerts from the main clusters to the neighbouring clusters. This is exemplified in the *HTTP inspect* alert group (clusters 9–11) and *TCP data offset* (clusters 5–6 and 9–11).

Both errors 1 and 2 are very minor errors resulting from cluster overlaps. Although very close, the TCP/IP attributes for both alert categories were not "strong" enough to produce crisp clusters of their own. Some of the *TCP data offset* alerts are shown in Table 6. The TCP/IP attributes of these three alerts closely match those in cluster 5, but for reasons stated above, they were placed in cluster 6.

We also had three cases of "stray" alerts in *TCP nmap Scan, Socks Proxy Scan* and *ICMP Superscan*. There isn't any significant difference between the clustered alerts and these stray alerts. These are likely the result of the cumulative effect of the ANN errors and the minor TCP/IP differences in these alerts and the clustered alerts.

The main errors encountered were in the four cases in which the alert groups failed to form individual clusters. The alerts for three of them are distributed inside other clusters (clusters 2 and 4). The overall reconstruction error for these alerts was indistinguishable with that of the clusters they are embedded in. The main cause of this type of error is attributable to the initial assumption of using a one-dimensional reconstruction error as the clustering attribute.

Some of these errors suggest the use of a supervised ANN to force the autoassociator to classify certain alerts into certain clusters. While the performance of our autoassociation model is very good, the few errors encountered could be possibly reduced by conducting further research into the clustering algorithm of the autoassociator. For example, the threshold value dividing the different cluster boundaries was shown to produce better results if made variable rather than fixed. In addition, collapsing a 42 dimensional TCP/IP input vector into a one-dimensional decision-metric (reconstruction error) may also introduce some errors in the final results.

Table 6. Sample of *TCP data offset* alerts.

```
[**][116:46:1] WARNING: TCP Data Offset is less than 5!
11/12-20:25:11.826507217.209.183.235:0 ->
207.166.252.249:0TCP TTL:236 TOS:0x0 ID:0 IpLen:20 DgmLen:40
*****R**Seq:  0x24A1C4C Ack:  0x24A1C4C Win:  0x0 TcpLen:  0

[**][116:46:1] WARNING: TCP Data Offset is less than 5!
09/05-07:34:31.984488 172.20.10.199:0 ->
138.97.150.9:0 TCP TTL:237 TOS:0x0 ID:0 IpLen:20 DgmLen:40
*****R**Seq:  0x58092C9A Ack:  0x58092C9A Win:  0x0 TcpLen:  12

[**][116:46:1] WARNING: TCP Data Offset is less than 5!
10/10-15:12:45.92650780.128.206.166:0 ->
32.245.166.119:0 TCP TTL:117 TOS:0x0 ID:25125 IpLen:20 DgmLen:40
DF*******F Seq:  0x720049 Ack:  0xD655185A Win:  0x5010 TcpLen:
```

6 Conclusions

A first generation model of an autoassociator correlation model was presented with very good results. Its ability to cluster similar alerts with good accuracy is a vital tool for the intrusion detection analyst. It also showed that our autoassociation model was often sensitive enough to small differences to set slightly different alerts apart; something that a human analyst could have missed (especially when swamped with the many alerts that are typical of networks today). The processing speed produced by the ANN classifier makes this approach a good candidate for implementation in real time. The model could be used as a correlation engine for an IDS that collects events from other Snort–based sensors. Future work enhancements could expand our autoassociation model to handle inputs from multiple heterogeneous sensors.

The errors associated with this approach may be attributable to the 42 to 1 mapping of the input variables to one dimensional decision metric–the reconstruction error. This may require expanding the output of our autoassociation model to cluster the alerts based on weighted values of the input metrics. This would also provide a possible ranking system for the alerts using the analyst's prior knowledge of the different alerts. Another possibility would be separating rare events from the common ones, by passing the unclustered alerts into another autocorrelator for reclassification.

All in all this demonstrates that the autoassociator used as a clustering system, rather than a novelty detector, is a worthwhile approach. This is an important conclusion given two attractive features of autoassociators. First, they issue soft custers in the sense that, unlike a number of other clustering systems, they output raw signals that can be interpreted by users as they wish. This is of great practical interest since it is often desirable to have a human being make final decisions rather than having it imposed by a piece of software. Second, the basic autoassociator used in this work can be upgraded in various ways (e.g. by adding extra hidden layers or coupling them with Kernel functions) in order to improve performance. In future work, we plan to try some of these potential upgrades as well as compare the system to other highly performing clustering devices such as Self-Organizing Maps or the EM-Algorithm.

References

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

[2] N. Japkowicz, "Supervised versus unsupervised binary-learning by feedforward neural networks," *Machine Learning*, vol. 42, no. 1/2, pp. 97–122, January 2001.

[3] N. Japkowicz, C. Myers, and M. Gluck, "A novelty detection approach to classification," in *The Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995, pp. 518–523.

[4] SANS Institute, "Shadow," *SANS Institute*, 2001. [Online]. Available: http://www.nswc.navy.mil/ISSEC/CID/

[5] Southcott Patrick, "Snort & Acid," *Online*, 2002. [Online]. Available: http://www.patricksouthcott.com/projects/saclug_snort_and_acid/SACLUG_SnortandACID.ppt

[6] J. Haines, D. K. Ryder, L. Tinnel, and S. Taylor, "Intrusion alert correlation: Validation of sensor alert correlations," *IEEE Security & Privacy*, pp. 46–56, Jan/Feb 2003.

[7] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Recent Advances in Intrusion Detection (RAID 2001)*, ser. Lecture Notes in Computer Science, no. 2212. Springer-Verlag, 2001. [Online]. Available: http://www.sdl.sri.com/papers/raid2001-pac/

[8] Erlinger Michael and Staniford-Chen Stuart, "Intrusion detection exchange format (IDWG)," *IETF*, 2003. [Online]. Available: http://www.ietf.org/html.charters/idwg-charter.html

[9] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *USENIX* 13^{th} *Systems Administration Conference (LISA 99)*, 1999. [Online]. Available: http://www.usenix.org/publications/library/proceedings/lisa99/roesch.html

[10] S. Northcutt and J. Novak, *Network Intrusion Detection : An Analyst's Handbook*, 2nd ed. Indianapolis, Indiana: New Riders, 2000.

[11] DARPA, "1999 darpa intrusion detection evaluation data set overview," *MIT: DARPA Intrusion Evaluation*, 1999. [Online]. Available: http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html