Using unsupervised learning for Network Alert Correlation

Reuben Smith[1], Nathalie Japkowicz[1], Maxwell Dondo[2], and Peter Mason[2]

[1] School of Information Technology and Engineering (SITE), University of Ottawa ON Canada
[2] Defence Research and Development Canada (DRDC) Ottawa ON Canada

Abstract. Alert correlation systems are post-processing modules that enable intrusion analysts to find important alerts and filter false positives efficiently from the output of Intrusion Detection Systems. Typically, however, these modules require high levels of human involvement in creating the system and/or maintaining it, as patterns of attacks change as often as from month to month. We present an alert correlation system based on unsupervised machine learning algorithms that is accurate and low maintenance. The system is implemented in two stages of correlation. At the first stage, alerts are grouped together such that each group forms one step of an attack. At the second stage, the groups created at the first stage are combined such that each combination of groups contains the alerts of precisely one full attack. We tested various implementations of the system. The most successful one relies in the first stage on a new unsupervised algorithm inspired by an existing novelty detection system, and the EM algorithm in the second stage. Our experimental results show that, with our model, the number of alerts that an analyst has to deal with is significantly reduced.

1 Introduction

Intrusion detection systems (IDSs) are computer programs or hardware that attempt to detect attacks against a computer network. IDSs are deployed to inform administrators of the threats against their network services and data. Attacks against networks are common and firewalls are suitable for stopping only certain types of attacks, so IDSs are important in protecting networks.

The output of IDSs, however, is considered low level since a single attack can be represented by several alerts. This makes the work of intrusion analysts quite difficult, if not, virtually, impossible since they have to try to reconstruct the entire pattern of potential attacks from the alerts they received, many of which, incidentally, correspond to false alarms.

Tools that could be of great help to analysts are alert correlation systems, that would automatically find correlations between IDS alerts that represent the same attack. More specifically, such tools would find relationships between alerts that indicate the motives and methods of a particular attack attempt against the network. Previously designed alert correlation systems are either based on machine learning techniques or non-machine learning techniques, including statistical methods, logical rules and graph

theory algorithms. In both cases, these methods require a high level of human involvement in creating the system and/or maintaining it, as patterns of attacks change as often as from month to month.

This paper introduces an unsupervised machine learning approach for network alert correlation which does not require any kind of human involvement once the system is installed. The system we present takes as input the output of an IDS which it processes in two stages of correlation. After the first stage of correlation, individual steps of the attacks in the dataset will be clustered. This means that, for example, all similar reconnaissance probes for each of the attacks in the dataset of alerts will be clustered. After the second stage of correlation, different steps of the attacks in the dataset will be clustered. So after the second stage of the correlation system each cluster will represent one specific attack in the dataset.

This work makes three contributions. The first is of an applied nature: our work introduces an alert correlation system that is easy to deploy and maintain on a computer system, and that is more effective than the simple rule-based systems which are commonly used by analysts because of their availability and low costs. The next two contributions are of a more theoretical nature. First, we demonstrate that pure unsupervised learning can be an effective approach for this difficult but practical task and second, we introduce a new clustering algorithm and demonstrates its competitiveness with EM, and SOMs, on a subset of our task. One of our more minor contributions, briefly discussed in the text, consists of extending the two attribute sets previously proposed for this type of task.

The remainder of the paper is divided into four sections. Section 2 presents background research in the area of network event correlation and situates our research within this work. Section 3 introduces our model in greater detail, discussing each correlation stage carefully. The new clustering algorithm we proposed is introduced in this section. Section 4 presents the experimental set up we used to select an optimal implementation for our model and to test it effectively on new data. It also presents and discusses the results we obtained. Section 5 concludes the paper and proposes future extensions of this work.

2 Background Research

The majority of the existing correlation tools use elementary approaches to correlate attacks. For example, Shadow [1] and ACID [2], use the IP addresses to correlate attacks. However, IP addresses may be spoofed, therefore using them alone may not provide a sufficient measure to classify the threat posed by an alert.

Recent work reported by Haines et al. [3], details some of the common correlation tools and approaches. The majority of these approaches use one alert metric at a time to correlate with other possible attacks. More sophisticated approaches use statistical methods on multiple alert metrics [4]. Hatala et al. [5] also analyses various alert correlation efforts by different groups. They give the details of a number of correlation systems, most of which are not based on machine learning techniques. Other approaches that do not involve machine learning are reported in [6, 7].

Machine-learning techniques in this area include the work by Julisch et al. [8], Dain and Cunningham [9], Zanero and Savaresi [10], and Laskov et al [11]. The method by Julisch et al. relies on clustering algorithms rather than classification algorithms, and requires an administrator's input to reflect network changes. Dain et al. use machine learning algorithms such as neural networks and decision trees to recognise attacks based on a list of features. Hatala et al. notes that this work uses a simplistic dataset which does not cover a wide range of attack scenarios. The Dain et al. research was tested with a defence conference (DEFCON) dataset. The use of this dataset simplified the problem of alert correlation because attackers were motivated by points in the competition and no points were awarded for stealthy attacks.

Our approach is related to that of Valdes and Skinner [4] who also proceeds by discovering attack step correlations, once lower level correlations have been established. However, their work uses statistical methods, and, thus, depends on the underlying attribute behavioural distributions (such as Gaussian distribution) of deviations from what is expected; by using a machine learning approach, our work was able to avoid the restrictions imposed by such parametric models. Our work is also related to that of Julisch et al. [8] in that it uses unsupervised learning and to that of Dain and Cunninghan [9] in its choice of a data representation, as will be discussed in the next section.

3 The Proposed System

The system is based on the idea that attacks can be decomposed into attack steps, which correspond to one action in the attackers greater plan, and, further, that single attack steps are made up of large numbers of IDS alerts. For instance, one step by an attacker might be running the security scanning tool *nmap*, once, against a network to discover what services are available. This step would likely generate many IDS alerts. With no correlation tool, a systems operator is faced with thousands of IDS alerts and has no way of knowing, easily, which alerts represent the same attack step nor which attack steps should be considered together as parts of the same attack.

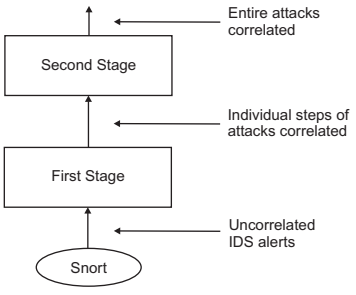

Fig. 1. Architecture of our alert correlation system

We recall that the system we propose is not an IDS, itself, but rather a post-processing module for the output of an IDS. It takes as input a set of uncorrelated IP packets that have been flagged as alerts by an IDS, specifically the Snort IDS in our experiments, and passes the alerts to the First Stage of Correlation module that outputs clusters of similar IP packets supposed to represent the same attack step. These clusters are subsequently input into the Second Stage of Correlation module which outputs *super-clusters* (clusters of clusters) representing a specific attack.

At the first stage of correlation, we cluster the dataset of alerts by constructing features that differentiate Snort alerts at the IP packet level. We are interested in clustering alert IP packets that are numerically similar in their protocol attributes, so we construct our set of attributes to use for clustering in such a way that numerically similar Snort alerts are clustered together. At the second stage of correlation we are interested in the trends between groups of similar Snort alerts. For example, if two separate sets of Snort alerts are comprised of alerts with the same identical IP source address, this might be an indicator that the sets of alerts are related. Our second set of features were constructed to reflect this goal. The next two subsections describe the kind of features that were used in our system while the following one describes the clustering algorithms used in both stages of correlation: AA, a new clustering approach based on a particular feedforword neural network architecture called an autoassociator and the EM algorithm.

Both the final choice of features and clustering algorithm were determined experimentally. This paper will describe a subset of the experiments that lead to our final choices in Section 4. The remainder of the experiments are described in [6, 7].

3.1 Features used in the First Stage of Correlation

The set of features extracted from Snort for the first stage of Correlation are listed in Table 1.[3] This set of features was created to fully represent an IP packet flagged as an alert by an IDS sensor. In this representation, however, we tried to ensure that only pro-

Table 1. First stage 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

[3] We used Snort's acronyms and shorthands to name our features.

tocol features were used. This means that no information specific to Snort was used and that, therefore, our approach is portable to other IDS systems. Another observation of interest is that we did not include the IP source address or IP destination address protocol fields in the list of features. This is because these features tended to impede correct clustering. We also reasoned that if we wish to cluster a number of packets together that make up a distributed denial of service or similar attack, the IP address fields could impede correct clustering since source IP address is easily forged for certain types of attacks, and the destination IP address can be masked by attacking many computers at once even if only one computer is the target.

3.2 Features used in the Second Stage of Correlation

For the second correlation stage, we take the output clusters of the first stage and represent them, each, as new data items, encoded using a new set of features. Feature construction for the data ultimately determines what is learned, therefore, we intentionally constructed 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.

The set of features constructed for the second stage of correlation are listed in Table 2. This list was inspired from the feature list of Dain et al [9]. Our list includes some of their features as well as some that they did not consider. All the features were meant to

Table 2. A cluster encoded for the second stage

Feature	Feature
numAlerts	ipSrcAddrCommonPart
ipSrcAddrCommonBits	ipDestAddrCommonPart
ipDestAddrCommonBits	modePortSrc
modePortDest	avgIpHdrLen
avgPayloadLen	avgReconsErr
avgSeqNumDiff	avgTimeSig
varTimeSig	avgTcpFlagsSet

capture the type of reasoning an attacker may go through or the kind of behaviour that can be observed when considering groups of alerts. We illustrate this kind of reasoning on a few of our features. Further explanation appear in [6, 7].

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

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 more common than the rest. Accordingly, we created two features: one feature to encode the most common TCP source port in a cluster of alerts and a feature to encode the most common destination port.

Please note that while the IP source and destination addresses were not used during the first stage of correlation, they are used during the second stage, since they were shown to bring about performance gains. This is because such features are valuable in detecting an attack source for many types of attacks. For those attacks in which IP spoofing (i.e., forging a source IP address) can be used, we found that allowing alerts with dissimilar IP addresses to be clustered together at the first stage of correlation adequately solved the problem without any need to repeat this approach at the second stage.

3.3 Clustering Algorithms used in both stages of Correlation

Along with the selection of features for each stage of correlation, clustering algorithms and their parameters also needed to be chosen. We experimented with three clustering systems: the EM algorithm, self-organizing maps and AA, a new algorithm based on the autoassociator, a particular feedforward neural network architecture. In the early stages of our research, we also experimented with k-means, but it was quickly dismissed given its low performance levels. This section begins by describing AA and then discusses the choices that were made in our final implementation. The EM algorithm [12] and self-organizing maps [13] will not be described here because of lack of space.

An autoassociator (AA) is a fully-connected three-layer, feedforward neural network whose objective is to reproduce an input vector x_i made up of alert attributes, at the output through weights w_i. However, the reproduction is not perfect and a reconstruction error, RE, is a measure of this imperfection. Similar attack scenarios have very similar attributes, and we expect their REs to be very close. Similar alerts are thus expected to be clustered together based on their RE. This idea is an extension of the work by [6, 7] on novelty detection. In that work, it was shown that the AA was good at clustering previously seen events, differentiating them from new events. This kind of clustering, however, was coarse-grained as the AA was only trained on the positive class and was only required to recognize data from that class and reject any other. Our purpose, here, is to use the AA in a similar capacity, but as a finer-grained clustering system. Specifically, we will train the AA with all the data available (not simply positive data, but rather, data that may belong to some indeterminate number of clusters) and cluster it according to their RE.

We will now describe this algorithm in more detail. A three-layer fully connected feedforward neural network with N input nodes, N output nodes and $J(< N)$ hidden nodes (unoptimised) was used. Although such feedforward architectures are usually used in a supervised fashion, in this work, we use it strictly in an unsupervised fashion: no instance labels are used. In place of the labels usually indicated at the output layer, the attribute vector used at the input layer is repeated at the output layer for each example. The neural network is, thus, expected to learn the identity function. In other words,

the output at node i of each layer is given by:

$$o_i = f(\sum_{k=1}^{K_i} w_{ik} y_k) \tag{1}$$

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 AA was trained using the error-backpropagation algorithm with the objective of reconstructing the input space at the output. The training objective is to minimise the square errors E, as given by:

$$E = \sum_{i=1}^{N} ||x_i - y_i||^2 \tag{2}$$

The neural network's weights were iteratively updated until convergence was achieved. Once training was completed, cluster barriers were created to separate adjacent clusters. This was achieved by comparing a predetermined threshold value to the resultant RE e_i produced by the difference between the alert vector $\mathbf{x}_i$ and the "new" alert vector $\mathbf{y}_i$. This difference is given by:

$$e_i = ||\mathbf{x}_i - \mathbf{y}_i|| \tag{3}$$

for each input vector $\mathbf{x}_i$. The predetermined threshold value was set experimentally. By varying the cluster barrier and recording the errors, we concluded that the best cluster barrier to use would be 0.0017. Equation 3 essentially reduced a 42-dimensional vector into a 1-dimensional threshold metric e used for clustering. The result of the clustering was multiple alert groups that have similar or closely similar attributes.

Note that an important difference between this algorithm and other more traditional ones is its reliance on imperfection. Indeed, we are exploiting the fact that the autoassociator is not able to fully reconstruct the input layer at the output level, to cluster the data, and that it will make different kinds of reconstruction "errors" on different categories of data, thus, allowing us to cluster them accurately. This kind of reasoning worked well in the case of novelty detection and we wanted to test whether this result would extend to the case of clustering.

There were three other reasons underlying our decision to explore the performance of the AA rather than content ourselves exclusively with off-the shelf algorithms. First, unlike other clustering algorithms like the *EM algorithm* or SOMs, the AA does not require knowledge of the number of output clusters in advance. It determines it through a self-optimisation process in the training algorithm. Second, unlike SOMs and traditional uses of the EM algorithm, the AA naturally performs soft-clustering that is then interpreted by the simple cluster barrier setting algorithm just discussed into hard clusters. More specifically, the AA can assign a number to each data point—the RE—such that data points corresponding to the same alert are roughly assigned similar numbers. This feature corresponds loosely to the notion of a ranking classifier in the supervised case. We believed that such flexibility could enhance clustering performance. Last but not least, we noticed that the RE of an alert was quite stable in that it is mostly independent of the data set that contains it. This is particularly important in our application

given that we separated the first correlation stage data into windows in order to make the results more manageable for the clustering algorithm as well as to include a rough form of correlation based on alert generation time. This means that the same alert appears surrounded by different alerts in various training scenarios and that its outcome should remain the same from one scenario to the next. Through its RE, initial experiments revealed that AA appears to obtain similar REs for the same alert in different context, a feature that the other algorithms do not provide.

4 Experiments Set-Up and Results

As previously mentioned, a large experimental study was performed prior to settling on the model presented in the previous section. It will be impossible for us to discuss the entire study in this paper. Therefore, we had to focus on a subset of our experiments. We chose to present the experiments which allowed us to select an optimal clustering approach for the first stage of correlation since these experiments highlight the usefulness of our new clustering algorithm, at least on one type of practical problem. Our other experiments were designed to select an optimal clustering approach for the second stage of correlation, to select the best parameters for all clustering algorithms (beyond the few results that are presented here), to choose an optimal cluster barrier setting approach for the AA (the one described above is the best one we identified), to select the best attribute sets for both stages of correlation, and to select an optimal method for scaling our attribute values in both stages (the optimal scale we settled on was $[-1, 1]$). These experiments are all detailed in [6, 7].

In order to validate our system's overall utility, we conclude this section by presenting the results obtained by our final model and by comparing them to a simple rule-based algorithm similar to those usually employed in real-world settings. The section starts with a presentation of the data sets used in both the development and testing parts of our study.

4.1 Data Sources

The system was tested using two alert data sets. We tested our system with Snort [14] alerts from incidents.org. We also used labelled alerts from the 1999 DARPA [15] IDS evaluation data. Since the labelled DARPA data was in packet form, we ran the data through Snort using commonly used configuration filters. We developed some Perl scripts to read the text-based Snort alerts into numeric data for use with our system.

In the incidents.org data set, we selected a training set of 10,000 alerts and a validation set of 500 alerts. In the DARPA data set, we selected a training set of 10,000 alerts, a validation set of 100 alerts and a testing set of 500 alerts. The incidents.org data set did not require a testing set since it was only used during the development of our system. We chose to develop our system using the incidents.org data set because it contains real and complex sets of attacks in contrast with the DARPA data which is simulated and, thus, of lesser value for the development stage. The DARPA data was set aside for formal testing of the completed system.

The data in the validation and testing sets of both domains were manually labeled for our task: based on our knowledge and experience, we placed alerts into clusters, each representing an attack step and super-clusters, each representing an attack attempt. This was a tedious and time consumming process, which is why the size of our labeled data sets is quite small.

4.2 Clustering Algorithm Selection Experiments

We examined three algorithms: the autoassociator introduced in the previous section, self-organization maps and the EM algorithm. All the experiments reported in this section were conducted on the incidents.org data set since they are part of the design of our system. The results obtained by each clustering method were compared against the benchmark clusters to determine performance. As mentioned above, we only report on the selection process in the first stage of correlation because of lack of space in the paper. The selection of a clustering system for the second stage of correlation was conducted in a similar fashion (see [6, 7]).

For the first stage of correlation, rather than training SOMs and the autoassociator with 10,000 training examples, we trained the two algorithms with only 1,017. This is because there were difficulties in training SOMS on the 10,000 alert set and we wanted the two algorithms to be trained on the same data for fair comparison purposes. EM was not trained at all, since it is not designed to be trained with unlabelled data. Rather, it builds clusters directly on the testing data. 1,017 training alerts comprises roughly the first 10% of each type of alert available from the 10,000 alert training set. We tested the system on the 500 alerts from the validation set.

The results of these experiments are presented in Figures 2, 3 and 4. In Figure 2, we

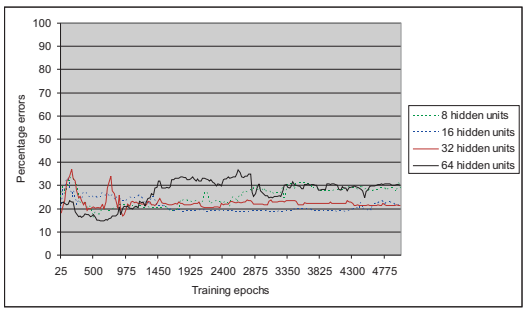

Fig. 2. First stage performance with the autoassociator

tested the AA by simultaneously varying the number of hidden units and the number of epochs. The learning rate was fixed and set at 0.4. From the graph, it is clear that the autoassociator attains the best performance if constructed with 64 hidden units and if trained for between 500 and 1000 epochs. Its error rate, in this region, falls below the

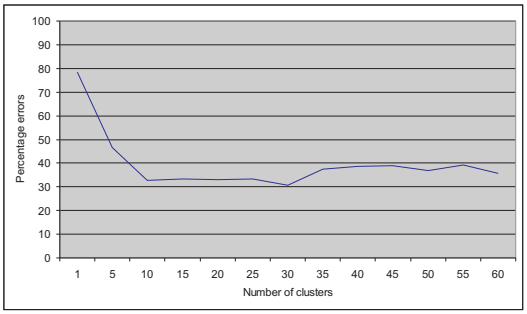

Fig. 3. First stage performance with the EM algorithm

20% mark. For the performance graph of the EM algorithm in Figure 3, we varied the number of clusters. We can see that if the number of clusters chosen is 10 or more, the performance results are consistently in the range of 30% to 40% errors. The best performance for the EM algorithm is reached at 30 clusters where it obtains a 30% error rate, or a 10% worse error rate than the AA. In Figure 4, we present the results of the

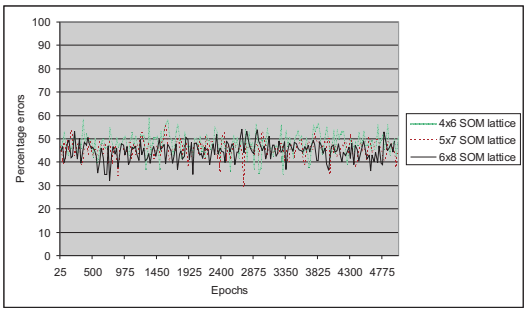

Fig. 4. First stage performance with the SOM algorithm

self-organizing maps for which we varied the lattice configuration and epochs simultaneously. The results for the SOM algorithm are noticeably worse than the results with the autoassociator and the EM algorithm. The results for all three lattice configurations vary significantly with no discernable range of best performance. From this we see that the performance of SOMs is universally poor on our problem, especially if compared with the autoassociator, which does not generate an error percentage greater than 40% in this evaluation.

We, thus, conclude that the autoassociator performs better than the EM algorithm and SOMs for this dataset against the incidents.org benchmark evaluation dataset.

4.3 Testing the Overall System

We conducted our final tests on the fresh data set provided by DARPA. It is important to note, though, that although we did not use the DARPA data to design the system, we still needed to tune and train the autoassociator on the 10,000 training and 100 validation alerts, respectively, prior to testing the overall system on the 500 testing alerts.

The system clustered the DARPA data set into 21 clusters, i.e., 21 attacks. From the benchmark labels, however, only 13 attacks were expected. This suggests that our system produced a number of "separation errors", in which alerts of the same kind were clustered into different groups. Indeed, we found that 54 out of 500 alerts were separated from the cluster they should have been in, but were nonetheless clustered along with similar alerts. Conversely, we found that very few "clustering errors" took place since we found that only 4 out of 500 alerts were clustered along with unrelated alerts. This is quite encouraging since separation errors are much less serious than clustering errors which may cause and IDS analyst to miss important events. Please see [6, 7] for tables and more complete descriptions of our results.

If considering both kinds of errors together, we can conclude that our system obtained an accuracy of 88.4%. On its own, this figure is not terribly meaningful, so we compared the results to those obtained using a simple rule-based correlation system based on the correlation descriptions of Northcutt [16] and the web-based ACID alert console [2]. Such a correlation algorithm is quite often used in practice. Using the Simple Correlator system, we obtained an accuracy of 79.4% on the DARPA data set. This means that our system achieved a 9% improvement.[4]

5 Concluding Remarks

In this work, we designed and implemented a two-stage alert correlation model and demonstrated how this model was able to cluster similar alerts with better accuracy than what we achieved with a simple rule-based correlation algorithm of the type often used in practice. The model we created is of a purely unsupervised nature and does not require any maintenance, once all the initial parameters are determined. This should provide intrusion analysts with a simple and effective solution to their alert flooding problem.

The paper also introduced a new clustering algorithm based on a neural network architecture called an autoassociator (AA). The main advantages of this algorithm is that it does not require knowledge of the number of output clusters in advance, that it performs soft-clustering and that its output remains stable in various contexts. Of particular interest here, the AA was also shown to be quite accurate on the first of our two clustering tasks.

Many avenues for future work stem from this study. First, we would like our results to be examined by a seasoned IDS analyst who could give us feedback on how

[4] Out of curiosity, we compared the two systems on the incidents.org domain. Our approach obtained an accuracy of 67.4% while the simple Correlator only reached an accuracy of 41.6%. This result, however, is optimistically biased given its reliance on the validation data set as well as the fact that it is based on internal, rather than external validation (i.e., testing within the same data set rather than on a separate one.)

to improve the system. Second, we recognize that our system still requires a lot of tuning, both in terms of algorithm and feature selection. In particular, we would like to experiment with other clustering algorithms and new features. Finally, we would like to examine the AA more carefully to establish its strengths and weaknesses.

References

1. Northcutt, S., *et al*: SHADOW: Second heuristic analysis for defensive online warfare
2. Danyliw, R.: ACID: Analysis console for intrusion detections
3. Haines, J., Ryder, D.K., Tinnel, L., Taylor, S.: Validation of sensor alert correlators. IEEE Security and Privacy (2003) 46–56
4. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001), LNCS #2212, Davis, CA, Springer-Verlag (2001) 54–68
5. Hätälä, A., Särs, C., Addams-Moring, R., Virtanen, T.: Event data exchange and intrusion alert correlation in heterogeneous networks. In: Proceedings of the 8th Colloquium for Information Systems Security Education (CISSE), Westpoint, NY, CISSE (June 2004 2004) 84–92
6. Smith, R., Japkowicz, N., Dondo, M.: Clustering using an autoassociator: A case study in network event correlation. In: Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems, Phoenix, AZ, ACTA Press (November 2005) 613–618
7. Japkowicz, N., Smith, R.: Autocorrel ii: Unsupervised network event correlation using neural networks. Contractor Report CR 2005-155, DRDC Ottawa, Ottawa, ON (October 2005)
8. Julisch, K., Dacier, M.: Mining intrusion detection alarms for actionable knowledge. In: Proceedings of SIGKDD '02, the 8th International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, ACM Press (July 2002 2002) 366–375
9. Dain, O., Cunningham, R.K.: Fusing a heterogeneous alert stream into scenarios. In: Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications, Philadelphia, PA, ACM Press (November 2001 2001) 1–13
10. Zanero, S., Savaresi, S.M.: Unsupervised learning techniques for an intrusion detection system. In: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, ACM (2004) 412–419
11. P. Laskov, P. Dussel, C.S., Rieck, K.: Learning intrusion detection: Supervised or unsupervised? In: Proceedings of the International Conference on Image Anaylsis and Processing (ICIAP 2005). 13th International Conference, Cagliari, Italy, Springer (2005) 50–57
12. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incoming data via the EM algorithm. J. Royal Stat. Soc., Series B **39**(1) (1977) 1–36
13. Kohonen, T.: Self-Organizing Maps. Volume 30 of Springer Series in Information Sciences. Springer-Verlag, Berlin, Germany (1995) (Second Extended Edition 1997).
14. Roesch, M.: Snort—lightweight intrusion detection for networks. In: Proceedings of LISA '99: 13th Systems Administration Conference, Seattle, Washington, The USENIX Association (November 7–12 1999) 229–238
15. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 darpa off-line intrusion detection evaluation. Computer Networks **34**(4) (2000) 579–595
16. Northcutt, S.: Network Intrusion Detection: An Analyst's Handbook. New Riders Publishing, Indianapolis, IN (1999)