Int. J. Advance. Soft Comput. Appl., Vol. 1, No. 1, July 2009
ISSN 2074-8523; Copyright © ICSRS Publication, 2009
www.i-csrs.org

Intelligent Alert Clustering Model for Network Intrusion Analysis

Maheyzah Md Siraj, Mohd Aizaini Maarof, and Siti Zaiton Mohd Hashim

Faculty of Computer Science and Information Systems
Universiti Teknologi Malaysia
81310 Skudai Johor, MALAYSIA
e-mail: {maheyzah, aizaini, sitizaiton}@utm.my

Abstract

As security threats change and advance in a drastic way, most of the organizations implement multiple Network Intrusion Detection Systems (NIDSs) to optimize detection and to provide comprehensive view of intrusion activities. But NIDSs trigger a massive amount of alerts even for a day and overwhelmed security experts. Thus, automated and intelligent clustering is important to reveal their structural correlation by grouping alerts with common attributes. We propose a new hybrid clustering model based on Improved Unit Range (IUR), Principal Component Analysis (PCA) and unsupervised learning algorithm (Expectation Maximization) to aggregate similar alerts and to reduce the number of alerts. We tested against other unsupervised learning algorithms to validate the performance of the proposed model. Our empirical results show using DARPA 2000 dataset the proposed model gives better results in terms of the clustering accuracy and processing time.

Keywords: *alert clustering, alert correlation, Expectation Maximization, Principal Component Analysis, unsupervised learning.*

1 Introduction

Despite more than 20 years' efforts on intrusion detection, no part of the Intrusion Detection System (IDS) is currently at a fully reliable level. Even though researchers are concurrently engaged in working on both detection and respond

sides of the system, a major problem in the IDS is the guarantee for the intrusion detection. This is the reason why in many cases IDS are used together with human experts (or analysts). In this way, IDS is actually helping the human security expert (SE) and it is not reliable enough to be trusted on its own.

Network Intrusion Detection Systems (NIDSs) have been extensively used by researchers and practitioners to maintain trustworthiness in systems [1]. However, NIDSs usually generated thousands of alerts even for a day. Worse, those alerts are in low quality because they mixed with false positives, and repeated warnings for the same attack, or alert notifications from erroneous activity [2]. Therefore, manually analyze those alerts are tedious, time-consuming and error-prone [3].

A promising technique to automatically analyze the intrusion alerts is called *correlation*. In specific, Alert Correlation System (ACS) is post-processing modules that provide high-level insight on the security state of the network and filter false positives as well as redundant alerts efficiently from the output of NIDSs. The analyses from ACS actually become an important guidance for SE to plan and develop the responsive and preventive mechanisms. Moreover, automation of ACS is crucial not only because to achieve a reliable and accurate response plans, but to reveal the continuously changing attack strategies.

Generally, correlation can be of two types: *structural* correlation and *causal* correlation. In this paper, we address the structural correlation (or alert clustering) aspect of NIDSs data to aggregate alerts with similar attributes. The main problem in existing ACSs is they 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 [4]. Our goal is to minimize the intervention (i.e., to ease the burden) of SE as much as possible, but not to replace them. In this paper we propose new, automated and intelligent hybrid clustering model called Improved Unit Range and Principal Component Analysis with Expectation Maximization (IPCA-EM) to aggregate similar alerts as well as to filter the low quality alerts.

The following section presents the overview of some related researches and necessary background information in the area of intrusion alert correlation. Section 3 describes each component involved in our proposed approach. Section 4 explains the dataset, experiments conducted followed by discussions of the results. Lastly, we conclude the paper and present potential future work.

2 Related Work

Most of the previous works [2], [3], [5], [6], [7] of alert clustering for finding structural correlation required strong dependencies on SE in developing and/or maintaining their correlation system. They either need pre-defined rules or human expert knowledge to manage and analyze the intrusion alerts. As a result, rules or

knowledge for such systems need to be updated periodically as patterns of attacks change drastically.

In [3], Aggregation and Correlation Component (ACC) is proposed to group the alerts into situations based on any combination of the three attributes: *source*, *target* and *alert class*. ACC relies on a set of rules to cluster the alerts. Whilst in CRIM [6] and Rule-Based Temporal ACS [7], they implemented a knowledge-based database to correlate and filter false positives alerts. Such database stored predicate logics to support logical reasoning in finding similarity between incoming alerts and existing alerts. In both cases, these approaches were time-consuming since they required a large number of predefined rules/knowledge in order to correlate alerts.

There are few works that cluster alerts based on supervised machine learning. For instance, algorithm introduced by [5] required a significant amount of alerts to be managed manually (i.e., hand-clustered) beforehand. Likewise, system by [2] required manual tuning periodically. Moreover, in their first system deployment, it needs to encode network properties to assist the clustering algorithm. Again, these approaches were time-consuming since regular setup and maintenance are significantly required for their system. Therefore, those constraints make the development of supervised learning-based correlation system less practical.

The closest work to ours was by [4] which used Expectation Maximization (EM) clustering algorithm as well in their second stage of correlation. A major different is that we implemented Principal Component Analysis (PCA) to obtain better performance. Detail justifications on the implementation of PCA in our work are presented in the next section.

3 Our Approach

The goal of this work is to find the best integration of PCA and unsupervised learning algorithm for clustering intrusion alerts. Our system architecture composed of six main components as illustrated in Fig. 1 (i.e., *alert normalization, alert preprocessing, dimension reduction*, *alert clustering, alert ranking and verification,* and *alert reduction*). In the first component, alerts that were generated by multiples NIDSs were collected and stored in database before they were modeled and converted into a standard format called Intrusion Detection Message Exchange Format (IDMEF). The formatted alerts were represented in numerical value and scaled to produce a balanced dataset. Since the number of alerts was huge and the alerts information was massive, we reduced the dimensionality of data using PCA. There were four unsupervised learning clustering algorithms tested. Among them, the EM gave better performance. Alerts in each cluster were ranked based on their severity level in order to discover the high and low risks of alerts. Based on the sensor's signatures file, alerts were verified to determine the false positives and invalid alerts. In the last

component, the system automatically merged redundant alerts, and discarded false positives and invalid alerts.

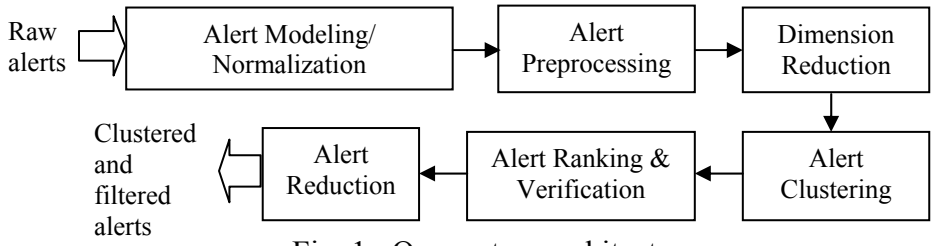

Fig. 1 : Our system architecture

3.1 Alert modeling/normalization

Recently, organizations use cooperative NIDSs to provide a better detection and global view of intrusion activities. This contributes to the diversity of output formats. In order to correlate alerts such diversified formats have to be converted into a unified standard representation. We applied IDMEF [8] to define the common data formats for the alerts. The IDMEF data model is implemented using a Document Type Definition (DTD) to describe Extensible Markup Language (XML) documents. IDMEF is also an object-oriented representation and a Unified Modeling Language (UML) model, as shown in Fig. 2 [8]. Whilst Table 1 shows the descriptions of each classes in IDMEF [8].

A sample of an alert in IDMEF is illustrated in Fig. 3. Referring to Fig. 3, the alert is uniquely identified by the '*Alert ident*' attribute. The *service* section describes network services on targets. In this case, it contains two attributes, namely *protocol* (tcp) and *port* (22). The target node address is specified by the *target* element and the alert message is given by the *Classification name* attribute. This alert simply reports a *stealth scan* on port 22 from 135.013.216.191 to 172.016.112.149. Note that s*tealth scan* attack is a kind of scan that is designed to go undetected by auditing tools. So scanning very slowly becomes a stealth technique.

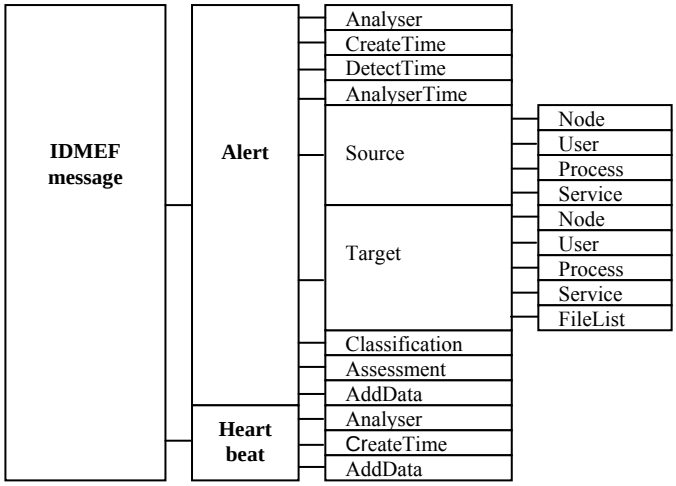

Fig. 2 : IDMEF data model

Table 1: The IDMEF-Message classes

Class	Description
Alert	Depending on the sensor, an *Alert* message may correspond to a single detected event or multiple detected events. Alerts occur asynchronously in response to outside events.
Analyser	Identification information for the analyzer (i.e., sensor) that originated the alert.
CreateTime	The time the alert was created. Of the three times that may be provided with an alert, this is the only one that is required.
DetectTime	The time the event(s) leading up to the alert was detected. In the case of more than one event, the time the first event was detected. In some circumstances, this may not be the same value as *CreateTime*.
AnalyserTime	The current time on the analyzer.
Source	The source(s) of the event(s) leading up to the alert.
Target	The target(s) of the event(s) leading up to the alert.
Classification	The 'name' of the alert.
Assessment	Information about the impact of the event, actions taken by the analyzer in response to it, and the analyzer's confidence in its evaluation.
*AddData	AdditionalData - Information included by the analyzer that does not fit into the data model. This may be an atomic piece of data, or a large amount of data provided through an extension to the IDMEF.
Heartbeat	Indicate analyzer current status. *Heartbeats* are intended to be sent in a regular period, say, every ten minutes or every hour. The receipt of a *Heartbeat* message from an analyzer indicates that the analyzer is up and running.
Analyser	Identification information for the analyzer that originated the heartbeat.
CreateTime	The time the heartbeat was created.
AddData	Similar description to *.

We extracted nine attributes for each alert. Thus, a vector for an alert A = {*SensorID, AlertID*, *SourceIPAddress, DestinationIPAddress, SourcePort, DestinationPort, ServiceProtocol, DetectTime, AlertType*}. To manage all attributes in more manageable way each attribute is stored in a field. An example of an alert attributes in a database is illustrated in Table 2 and they are extracted from an XML document (as shown in Fig. 3).

Table 2 : An example of an alert in database

Sensor ID	Alert ID	SrcIPAddress	DestIPAddress	Src Port	Dest Port	Serv	Time	AlertType

109	289	135.013.216.191	172.016.112.149	22	22	tcp	2007-11-24 17:42:31	STEALTH ACTIVITY

```
<IDMEF-Message/>
<?xml version="1.0"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX
IDMEF v1.0//EN" "/usr/local/etc/idmef-message.dtd">
<IDMEF-Message version="1.0">
   <Alert ident="289">
      <Analyzer analyzerid="109" model="snort" version="2.0.5">
            <Node>
               <name>tcpdump_dmz</name>
            </Node>
      </Analyzer>
      <CreateTime ntpstamp="0xc36cc187.0xd3aa9b49">2007-11-
            24T17:42:31Z</CreateTime>
            <Source>
            <Node>
               <Address category="ipv4-addr">
                  <address>135.013.216.191</address>
               </Address>
            </Node>
                <Service>
                  <port>22</port>
                     <protocol>tcp</protocol>
                </Service>
            </Source>
            <Target>
            <Node>
               <Address category="ipv4-addr">
                  <address>172.016.112.149</address>
               </Address>
            </Node>
                <Service>
                  <port>22</port>
                     <protocol>tcp</protocol>
                </Service>
            </Target>
      <Classification origin="vendor-specific">
            <name>msg=(spp_stream4) STEALTH ACTIVITY
            (NULL scan) detection</name>
               <url>none</url>
      </Classification>
   </Alert>
</IDMEF-Message>
```

Fig. 3 : IDMEF representation of an alert in an XML document

3.2 Alert preprocessing

Alert attributes are in the form of numerical and non-numerical values. Attributes that contain numerical values are *AlertID, SensorID, SourcePort*, *DestinationPort*, and *DetectTime*. The rest are non-numerical values (i.e., *SourceIPaddress*, *DestinationIPaddress, ServiceProtocol* and *AlertType*) and have to be mapped into numerical values. For instance to convert a 32-bit IP address (IP_{addr}) which in *X*1.*X*2.*X*3.*X*4 format, mapping as (1) was used.

$$IP_{addr} = ((X1 \times 256 + X2) \times 256 + X3) \times 256 + X4 \quad (1)$$

We scaled all values in the range of [0,1]. We tested against two scaling methods to find the best result in the alert clustering component. They were Unit Range (UR) and Improved Unit Range (IUR) as in (2) and (3) respectively, where x' is the scaled value, x is raw value, x_{max} is maximum value and x_{min} is minimum value.

$$x' = \frac{(x - x_{min})}{(x_{max} - x_{min})} \quad (2)$$

$$x' = 0.8 \times \frac{(x - x_{min})}{(x_{max} - x_{min})} + 0.1 \quad (3)$$

3.3 Dimension reduction using PCA

PCA has proven to be a useful technique for dimension reduction and multivariate analysis [9]. An important virtue of PCA is that the extracted components are statistically orthogonal to each other. This produces speedup training and robust convergence as shown in [10]. We expect that the unsupervised learning algorithm can work much better with PCA. According to [9], PCA is for a set of observed vectors $\{v_i\}$, $i \in \{1,2,...,N\}$, the q principle axes $\{w_j\}$, $j \in \{1,2,..,q\}$ are those orthonormal axes onto which the retained variance under projection is maximal. It can be shown that the vectors w_j are given by the q dominant eigenvectors (i.e. those with largest associated eigenvalues) of the covariance matrix $C = \sum_i \frac{(v_i - \bar{v})(v_i - \bar{v})^T}{N}$ such that $Cw_j = \lambda_i w_j$, where $\bar{v}$ is the simple mean. The vector $u_i = W^T(v_i - \bar{v})$, where $W = (w_1, w_2, ..., w_q)$, is thus a q-dimensional reduced representation of the observed vector $\{v_i\}$.

For the intrusion alerts in the dataset, the purpose of performing PCA is to find the principal components of the alerts, i.e., the attributes vector that can describe the alerts exactly and sufficiently, but not redundantly. In mathematical terms, we wish to find the principal components of the distribution of the alerts, or the eigenvectors of the covariance matrix of the set of the alerts [9], [11].

3.4 Alert clustering with unsupervised learning

Besides EM, we tested against other three unsupervised learning algorithms namely Self-organizing maps (SOM), K-means, and Fuzzy c-means (FCM) for performance comparison. Noted that, in this clustering component, we did not include the *AlertID*, *SourceIPAddress,* and *DestinationIPAddress* attributes

because as mentioned in [12] IP address tended to impede correct clustering since they are easily forged. However these attributes will be used for the next stage of correlation in our future work.

3.4.1 SOM

SOM [13] is a competitive learning algorithm that reduces the dimensions of data by mapping high dimensional data onto a set of units set up in a 2-dimensional lattice. An n-dimensional weight vector is associated with each unit, having the same dimension of the input space. At each step, the Euclidean distances between a randomly selected input vector *x* and all the units weight vectors w_i is calculated. The unit having the shortest distance to the input vector is identified to be the best matching unit *c* for *x*. As a result, the winner index *c* or best matching unit (BMU) for input vector *x(t)* is identified. Then, the input is mapped to the location of the BMU. We updated the weight vectors of the units neighboring the BMU(*c)* and of BMU itself according to (4), for $i = c$ and its neighbours.

$$w_i(t+1) = w_i(t) + \delta(t)[(x(t) - w_i(t))] \tag{4}$$

3.4.2 K-means

K-means [14] follows a simple and easy way to cluster a given data set through a certain number of clusters (assume *k* clusters) fixed a priori. The main idea is to define *k* centroids (or center), one for each cluster. The centroid is the average of all the points in the cluster i.e., its coordinates are the arithmetic mean for each dimension separately over all the points in the cluster. The better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point, *k* new centroids are re-calculated as barycenters of the clusters resulting from the previous step. With these *k* new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop, the *k* centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function is as (5):

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2 \tag{5}$$

where $\left\| x_i^{(j)} - c_j \right\|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j, is an indicator of the distance of the *n* data points from their respective cluster centres. The algorithm is also significantly sensitive to the

initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect. The main advantages of this algorithm are its simplicity and speed which allows it to run on large datasets [14].

3.4.3 FCM

Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. This method (developed by [15] and improved by [16]) is frequently used in pattern recognition. It is based on minimization of the following objective function as (6):

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^{m}\left\|x_i - c_j\right\|^2 \quad ,1 \le m < \infty \tag{6}$$

where m is any real number greater than 1, u_{ij} is the degree of membership of x_i in the cluster j, x_i is the ith of d-dimensional measured data, c_j is the d-dimension center of the cluster, and $||x_i - c_j||$ is any norm expressing the similarity between any measured data and the center. Fuzzy partitioning is carried out through an iterative optimization of the objective function shown above, with the update of membership u_{ij} and the cluster centers c_j by (7) and (8) respectively:

$$u_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{\left\|x_i - c_j\right\|}{\left\|x_i - c_k\right\|}\right)^{\frac{2}{m-1}}} \tag{7}$$

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{8}$$

This iteration will stop when $max_{ij}\left\{\left|u_{ij}^{(k+1)} - u_{ij}^{(k)}\right|\right\} < \varepsilon$, where ε is a termination criterion between 0 and 1, whereas k are the iteration steps. This procedure converges to a local minimum or a saddle point of J_m in (6).

3.4.4 EM

The EM algorithm [17] consists of two repeated steps, Expectation and Maximization. It uses a statistical model called Gaussian 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 (e.g, number of iteration and log likelihood difference between two iterations) 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 are reached.

In particular, it attempts to find the parameters θ that maximize the log probability logP(x;θ) of the observed data. It reduces the difficult task of optimizing logP(x;θ) into a sequence of simpler optimization subproblems, whose objective functions have unique global maxima that can often be computed in closed form. These subproblems are chosen in a way that guarantees their corresponding solutions $\phi^{(1)}\phi^{(2)}$,... and will converge to a local optimum of logP(x;θ). More specifically, the Expectation step (E-step) of the algorithm estimates the clusters of each data instance given the parameters of the finite mixture. During the E-step, the algorithm chooses a function g_t that lower bounds logP(x;θ) everywhere, and for which $g_t(\phi^{(1)})$=logP(x; $\phi^{(t)}$).

The Maximization step (M-step) of the algorithm tries to maximize the likelihood of the distributions that make up the finite mixture, given the data [12]. During the M-step, the algorithm moves to a new parameter set $\phi^{(t+1)}$, that maximizes g_t. As the value of the lower-bound g_t matches the objective function at $\phi^{(t)}$, it follows it follows (9), so the objective function monotonically increases during each of the iterations in EM [18].

$$\log P(x;\phi^{(t)}) = g_t(\phi^{(t)}) \leq g_t(\phi^{(t+1)}) = \log P(x;\phi^{(t+1)}) \quad (9)$$

3.5 Alert ranking and verification

Alerts that issued by NIDSs were not all in the same level of severity and importance. It would be great if the system can identify which alerts are highly important and which are not, so that the number of alerts that need to be deal with can be reduced. The algorithm for alert ranking and verification component is shown in Fig. 4. As shown in Fig. 4, we automatically cross-checked each alerts with the sensor's signatures file [19] to determine the priority of alerts and to verify the false positive and invalid alerts. In alert ranking, we introduced three level of severity: (1) *High-risk*, (2) *Medium-risk* and (3) *Low-risk*. For each level, we associate a numerical weight of priority in order to distinguish significant alarms from the others.

3.6 Alert reduction

Given the clustered alerts from previous component, redundant alerts (i.e., alerts that have equal values in all attributes) in each cluster were merged into a hyper-alert. In specific, repeated alerts for each cluster were represented as one. Moreover, with the reduction of invalid, false positive and low risk alerts, the total

number of alerts left for future analysis is significantly reduced. The alert reduction algorithm is shown in Fig. 5.

```
Cj : Cluster, ci : alert in cluster Cj, ci.rank : weight for severity level, ci.verify :
weight for false/true intrusion.

global ranking (alert) {
  while each alert ci ∈ Cj , do
    if (ci.AlertType is null)
        ci.rank ← -1
        invalid++
    else :
        if (ci.AlertType matched LOW severity level)
            ci.rank ← 0              %Low-risk%
            low++
        else :
            if (ci.AlertType matched MEDIUM severity level)
            ci.rank ← 0.5            %Medium-risk%
            medium++
            else :
                ci.rank ← 1          %High-risk%
                high++
  outputs invalid, low, medium, high
  call verify(alert)       }

global verify (alert) {
  while each alert ci ∈ Cj , do
    if (ci.AlertType is reported as FALSE POSITIVES)
        ci.verify ← 1
        false++
    else :
        ci.verify ← 0
  outputs false
  call reduction(alert)     }
```

Fig. 4 : Algorithm for alert ranking and verification

```
mode: automatic delete for low risk alerts (1) need permission, (2) no need

global reduction (alert) {
  while each alert ci ∈ Cj , do
    if (all attributes values in ci is EQUAL to all attributes values in ci+1 )
        delete ci
        merge++
    else :
        if (ci.verify is '1')
            delete ci
        else :
            ask mode
                if (ci.rank is '0' AND mode is '0')
                    delete ci
                else :
  outputs sum of merge, invalid, false, low  }
```

Fig. 5 : Algorithm for alert reduction

4 Results and Discussions

Performing real attacks in real networks to produce NIDS alerts as datasets are not realistic [20] and our work therefore shares the weaknesses with other published research works in the area whom using publicly available benchmark data. The lack of publicly available and representative datasets hinders ACS research and makes the comparison of different ACS and algorithms difficult. Most of the research community of IDS evaluated their works with DARPA's datasets. These datasets are, nonetheless, the only publicly available datasets in evaluating IDSs.

4.1 Dataset and Experiments

The experiments were conducted with MIT Lincoln's Lab's DARPA 2000 Scenario Specific Dataset [21]. The dataset contain simulated multi-staged attack scenarios in a protected environment: the intruder probes, breaks-in, installs the Distributed Denial-of-Service (DDoS) daemon and launches a DDoS attack against an off-site server. Since we are dealing with the sensor data, alerts reported by RealSecure network sensor Version 6.0 [22] which were provided by [23] were used to evaluate the effectiveness of our model.

The alerts data represents two kinds of attack scenarios (i.e., scenario 1.0 and scenario 2.0.2) in two types of networks (i.e., *inside* and *dmz* network). Attacks in scenario 2.0.2 were stealthier than scenario 1.0. For this paper, we only used alerts data for scenario 2.0.2 in *dmz* network. For implementation of the model, we used MATLAB Software [24]. We have five set of experiments as illustrated in Table 3 : (1) clustering with UR only (i.e., labeled as UR), (2) clustering with IUR only (i.e., labeled as IUR), (3) clustering with PCA only (i.e., labeled as PCA), (4) clustering with UR and PCA (i.e., labeled as UPCA), and (5) clustering with IUR and PCA (i.e., labeled as IPCA).

Table 3 : Clustering performance

Model	FCM				K-means				SOM [25]				EM			
	CE	*ER (%)*	*AR (%)*	*Time (sec)*	*CE*	*ER (%)*	*AR (%)*	*Time (sec)*	*CE*	*ER (%)*	*AR (%)*	*Time (sec)*	*CE*	*ER (%)*	*AR (%)*	*Time (sec)*
UR	78	18.40	81.60	1.30	62	14.62	85.38	4.23	139	32.78	67.22	4.22	47	11.08	88.92	1.90
IUR	74	17.45	82.55	1.27	57	13.44	86.56	4.40	135	31.84	68.16	4.21	45	10.61	89.39	1.85
PCA	133	31.37	68.63	3.56	141	33.25	66.75	5.20	170	40.09	59.91	6.52	86	20.28	79.72	2.67
UPCA	70	16.51	83.49	4.80	52	12.26	87.74	6.12	127	29.95	70.05	7.44	43	10.14	89.86	4.64
IPCA	67	15.80	84.20	4.81	46	10.85	89.15	6.18	112	26.42	73.58	7.42	**41**	**9.67**	**90.33**	**4.59**

Table 4 : Total of ranked alerts

Priority	Type of Alerts	Total
High-risk	Admind, HTTP_ActiveX, HTTP_Cisco_Catalysts_Exec, Sadmind_Amslverify_Overflow	10
Medium-risk	Email_Almail_Overflow, FTP_Pass	43
Low-risk	Email_Ehlo, FTP_Put, FTP_Syst, FTP_User, HTTP_Java, SSH_Detected, TCP_Urgent_Data, TelnetEnvAll, TelnetTerminalType, TelnetXdisplay	371

Table 5 : Total reduction of alerts

Merged	Invalid	FP	Low-risk	SUM	Reduction (%)
3	2	1	371	377	87.67

4.2 Data Analysis

The number of alerts tested was 430. The results obtained were compared against the benchmark clusters (i.e., 16 clusters are expected) to determine the performance of the proposed model. As in Table 3, we used four measurements: (1) *Clustering Error* (CE) is the number of alerts that are wrongly clustered. (2) *Error rate* (ER) is the percentage of wrongly clustered alerts, ER = (CE ÷ Total number of alerts observed) x 100, (3) *Accuracy Rate* (AR) is the percentage of alerts that are accurately clustered as they should be, AR = 100 – ER, and (4) *Time* is the algorithm processing time in seconds.

We varied the number of clusters in FCM, K-means, and EM to find the optimal results. Similarly, we tested the SOM by simultaneously varying the epochs and lattice configuration. Two third of the dataset was used for training and the rest was for testing. The best result on SOM (i.e., 73.58% with IPCA) was attained after it was trained for 2500 epochs using hexagonal 4 by 6 lattice type. It produced 12 clusters. It term of time costs, the overall processing time for training and testing was 7.42 seconds. The processing time might be longer if the dataset, epochs and/or lattice type are larger.

Overall, the best performance was with EM (i.e., 90.33% with IPCA) which was reached at 14 clusters and the processing time was 4.59 seconds. In each cluster, similar types of alerts were grouped together to represent an attack step. Since FCM, K-means, and SOM have a larger value of CE, it means that they put a large number of alerts that should belong together in one cluster into another clusters. Therefore, we summarized that the proposed model (i.e., IPCA-EM) is

effective and performed better than the rest of the algorithms tested for this dataset in terms of clustering accuracy and processing time.

Table 4 presented the type of alerts with their level of rank/priority. It shows that the majority of the alerts are most probably not serious at all. SE might found such alerts inappropriate to be analyzed and correlated. But, others may feel they are appropriate. Because of that, our system provided two kind of mode to automatically delete the low risk alerts: (1) need permission from SE, or (2) no need. If (2) was chose, then the total reduction of alerts was significant (see Table 5). Table 5 illustrated the total alerts in each category (i.e., merged redundant alerts, invalid, false positives and low risk alerts) which the system considers in order to reduce the amount of alerts. The original input data was 430, thus total reduction of unwanted alerts was 87.67%.

5 Conclusion and Future Work

Automation of alert management and analysis is crucial because alerts are in low level information and the volume is very large that make them tedious and hardly to be analyzed manually. Since alerts are not significant if they are isolated, thus finding the relationships between them is an important stage.

Grouping and clustering the alerts based on their feature similarities actually can reveal the attack steps launched by the attackers. Moreover, redundant alerts can be detected and merged easily. Therefore, the novelty of this work is the new integration of IUR, PCA and EM algorithm (which we called it IPCA-EM) as a solution to cluster multi sensors' intrusion alerts and to filter out the unwanted alerts. To the best of our knowledge, this is the first attempt for such integration and produces better results.

Altogether, the results are encouraging in terms of clustering accuracy rate and processing time compared to other unsupervised learning algorithms tested in this paper. Noted that a successful network attack consists of multi-stages attack, and an attack stage may comprise of one/more attack steps. Thus, we need a secondary clustering component to aggregate similar attack types to reveal the stages of attack. This becomes our main future work besides testing the proposed model with larger dataset. In the near future, we would like to develop a collaborative multi-stages correlation system to determine known and unknown attack scenarios.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Higher Education (MOHE), Malaysia.

References

[1] A. Siraj, and R. B. Vaughn, "Multi-level alert clustering for intrusion detection sensor data," *Proc. of the North American Fuzzy Information Processing Society*, (2005), pp. 748-753.

[2] K. Julisch, and M. Dacier, "Mining intrusion detection alarms for actionable knowledge," *Proc. of the 8th ACM Int. Conf. on Knowledge Discovery and Data Mining*, (2002), pp. 366–375.

[3] H. Debar, and A. Wespi, "Aggregation and correlation of intrusion detection alerts," *Proc. of the 4th Int. Symp. on Recent Advances in Intrusion Detection*, (2001), pp. 87–105.

[4] R. Smith, N. Japkowicz, M. Dondo, and P. Mason, "Using unsupervised learning for network alert correlation," *Springer-Verlag LNAI 5032*, (2008), pp. 308-319.

[5] O.M. Dain, and R. K. Cunningham, "Fusing a heterogeneous alert stream into scenarios," *ACM Workshop on Data Mining for Security Applications*, (2001), pp. 1-13.

[6] F. Cuppens, and A. Miege, "Alert correlation in a cooperative intrusion detection framework," *Proc. of the IEEE Symp. on Security and Privacy*, (2002), pp. 202-215.

[7] P. Kabiri, and A. A. Ghorbani, "A rule-based temporal Alert Correlation System," *Int. J. of Network Security*, Vol. 5, No. 1, (2007), pp. 66–72.

[8] H. Debar, D. Curry, and B. Feinstein. (2007). The Intrusion Detection Message Exchange Format (IDMEF) [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc4765.txt

[9] I. T Jolliffe, *Principal Component Analysis* (3rd ed.). New York: Springer Verlag, (2002).

[10] E. Oja, "Neural networks, principal components, and subspaces," *Int. Journal of Neural Systems*, Vol. 1, No. 1, (1989), pp. 61-68.

[11] J. X. Wang, Z. Y. Wang, and K. Dai, "Intrusion alert analysis based on PCA and the LVQ neural network," *Proc. of the13th Int. Conf. on Neural Information Processing*, (2006), Vol. 4234, pp. 217-224.

[12] N. Japkowicz, and R. Smith, "Autocorrel II: Unsupervised network event correlation using neural networks," *Contractor Report*, CR2005-155, DRDC Ottawa, (2005).

[13] T. Kohonen, *Self-Organizing Maps: Series in Information Sciences* (3rd ext. ed.). Berlin: Springer, (2001).

[14] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proc. of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley: University of California Press, Vol. 1, (1967), pp. 281-297.

[15] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *J. of Cybernetics*, Vol. 3, (1973), pp. 32-57.

[16] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York: Plenum Press, (1981).

[17] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood from Incoming data via the EM algorithm," *J. Royal Stat. Soc.*, Series B, Vol. 39, No. 1, (1977), pp. 1–36.

[18] C. B. Do, and S. Batzoglou, "What is the Expectation Maximization algorithm?," *Nature Biotechnology*, Vol. 26, (2008), pp. 897-899.

[19] RealSecure Signatures Reference Guide. Internet Security Systems [Online]. Available: http://xforce.iss.net

[20] T. Pietraszek, *Alert Classification to Reduce False Positives in Intrusion Detection*. PhD Thesis. Germany: Albert-Ludwigs-Universit¨at Freiburg im Breisgau, (2006).

[21] MIT Lincoln Lab. (2000). DARPA 2000 Intrusion Detection Evaluation Datasets [Online]. Available: http://ideval.ll.mit.edu/2000index.html

[22] Internet Security Systems. RealSecure Network 10/100 [Online]. Available: http://www.iss.net/products_services/enterprise_protection/rsnetwork/sensor.php

[23] P. Ning. (2002). TIAA: A Toolkit for Intrusion Alert Analysis [Online]. Available: http://discovery.csc.ncsu.edu/software/correlator

[24] The MathWorks. MATLAB: The Languange of Technical Computing [Online]. Available: http://www.mathworks.com

[25] A. Faour, P. Leray, and B. Eter, "Automated filtering of network intrusion detection alerts," *Proc. 1st Joint Conf. on Security in Network Architectures and Security of Information Systems*, (2006), pp. 277-291.