Multistep Attack Detection and Alert Correlation in Intrusion Detection Systems

Fabio Manganiello, Mirco Marchetti, and Michele Colajanni

Università degli Studi di Modena e Reggio Emilia,
Department of Information Engineering
Via Vignolese 905, Modena, Italy
{fabio.manganiello, mirco.marchetti, michele.colajanni}@unimore.it

Abstract. A growing trend in the cybersecurity landscape is represented by *multistep* attacks that involve multiple correlated intrusion activities to reach the intended target. The duty of reconstructing complete attack scenarios is left to system administrators because current Network Intrusion Detection Systems (NIDS) are still oriented to generate alerts related to single attacks, with no or minimal correlation.
We propose a novel approach for the automatic analysis of multiple security alerts generated by state-of-the-art signature-based NIDS. Our proposal is able to group security alerts that are likely to belong to the same attack scenario, and to identify correlations and causal relationships among them. This goal is achieved by combining alert classification through Self Organizing Maps and unsupervised clustering algorithms. The efficacy of the proposal is demonstrated through a prototype tested against network traffic traces containing multistep attacks.

Keywords: Network security, neural networks, alert correlation

1 Introduction

The presence of a Network Intrusion Detection System (*NIDS*) is a cornerstone in any modern security architecture. A typical NIDS analyzes network traffic and generates security alerts as soon as a malicious network packet is detected. Alert analysis is then performed manually by security experts that parse the NIDS logs to identify relevant alerts and possible causal relationships among them. While log analysis can be aided by NIDS management interfaces, it is still a manual, time consuming and error prone process, especially because we are experiencing a growing number of *multistep* attacks that involve multiple intrusion activities.

This paper proposes a novel alert correlation and clustering approach that helps security analysts in identifying multistep attacks by clustering similar alerts produced by a signature-based NIDS (such as Snort [10]), and by highlighting strong correlations and causal relationships among different clusters. This goal is achieved through multiple steps: alerts are preprocessed by a hierarchical clustering scheme; then, they are processed using a Self-Organizing Map [5]; the alerts that likely belong to the same attack scenarios are clustered by using the *k*-means algorithm over the SOM output layer; finally, a correlation index is computed among the alert clusters to identify causal relationships that are typical of multistep attacks. The final output of our framework for multistep attack detection is a set of oriented graphs. Each graph describes an attack scenario in which the vertices represent alert clusters belonging to the same attack scenario, and the directed links denote alert clusters that are tied by causal relationships. In such a way, a security administrator can immediately identify correlated alerts by looking at the graphs, avoiding to waste time on checking false positives and irrelevant alerts.

This paper presents several contributions with respect to the state of the art. The application of the k-means clustering algorithm to the output layer of a SOM allows us to perform robust and unsupervised clustering of correlated NIDS alerts. To the best of our knowledge, this solution has never been proposed in network security. Moreover, several original (for the security literature) heuristics for the initialization of the SOM and for the definition of the number of clusters produced by the k-means algorithm reduce the number of configuration parameters and allow our solution to autonomously adapt to different workloads. This is an important result, because security alerts are heterogeneous and present high variability. We demonstrate the feasibility and efficacy of the proposed solution through a prototype that was extensively tested using the most recent datasets released after the 2010 Capture the Flag competition [3].

2 Related work

The application of machine learning techniques, such as neural networks and clustering algorithms, for intrusion detection has been widely explored in the security literature. Several papers propose clustering algorithms, support vector machines [6] and neural networks as the main detection engine for the implementation of anomaly-based network intrusion detection systems. In particular, the use of *Self-Organizing Maps* (SOM) for the implementation of an anomaly-based IDS was proposed in [13], [8] and [1]. Unlike previous literature mainly oriented to anomaly detection, in this paper we propose SOM and clustering algorithms for the postprocessing of security alerts generated by a signature-based NIDS. Hence, our proposal relates to other papers focusing on techniques for NIDS alert correlation [2]. According to the comprehensive framework for NIDS alert correlation proposed in [12], the solution proposed in this paper can be classified as a *multistep correlation* component. In this context, two related papers are [7] and [4].

Our work differentiates from [7] because our evaluation is not limited to the computation of alert similarity. Indeed, mapping security alerts on the output layer of the SOM is only one intermediate steps of our framework, that uses the SOM output as the input of the alert clustering algorithms. Moreover, with respect to [7] we propose an innovative initialization algorithm for the SOM and an adaptive training strategy that makes the SOM more robust with respect to perturbations in the training data (see Section 4 for details on the design and implementation of the SOM). Finally, instead of relying just on a commutative correlation index based on the distance between two alerts, our correlation index depends also on the type of alerts and on their detection time (see Section 6). The resulting correlation index expresses the causality relationships among alerts much better than the previous one.

In [4] security alerts generated by a NIDS are grouped through a hierarchical clustering algorithm. The classification hierarchy, that is defined by the user, aggregates alerts of the same type targeting one host or a set of hosts connected to the same subnet. We use a similar hierarchical clustering scheme as a preprocessing step. We then use the alert clusters generated by this hierarchical clustering algorithm as an input for the SOM. Hence, we take advantage of the ability of the algorithm presented in [4] to reduce the number of alerts to process, and to group a high number of false positives. All the subsequent processing steps are novel.

3 Software architecture

The architecture of the framework proposed in this paper consists of a preprocessing phase and of three main processing steps.

The preprocessing phase, called *hierarchical clustering* in Figure 1, takes as its input the intrusion alerts generated by a signature-based NIDS. In our reference implementation, we refer to the well known Snort. Alerts are grouped according to a clustering hierarchy in a way similar to [4]. This preprocessing phase has two positive effects. It reduces the number of elements that have to be processed by the subsequent steps, with an important reduction of the computational cost of the three processing phases. Moreover, it groups many false positives in the same cluster, thus simplifying human inspection of the security alerts. Clustered alerts are then processed by the SOM, that is able to reduce the dimensionality of the input dataset by mapping each multidimensional tuple to one output neuron. Each output neuron is identified by its coordinates on the output layer of the SOM, hence clustered alerts are mapped to two-dimensional coordinate sets on the output layer. Moreover, a SOM has the ability to map similar tuples to neurons that are close in the output layer. In particular, the distance between two output neurons on the output layer of the SOM is inversely proportional to the similarity of the related inputs.

The output of the SOM is then analyzed by the second processing step, that is a k-means clustering algorithm. The basic idea is that similar alert clusters are mapped on close neurons of the SOM, hence it is possible to group similar alert clusters by executing a k-means clustering on the output layer of the SOM.The choice of the parameter k is critical. Our proposal includes a heuristic that computes the best k on the basis of the data analyzed so far, as illustrated in Section 5. This approach allows our clustering algorithm to automatically adapt its parameters without setting a static value for k that risks to be unsuitable in

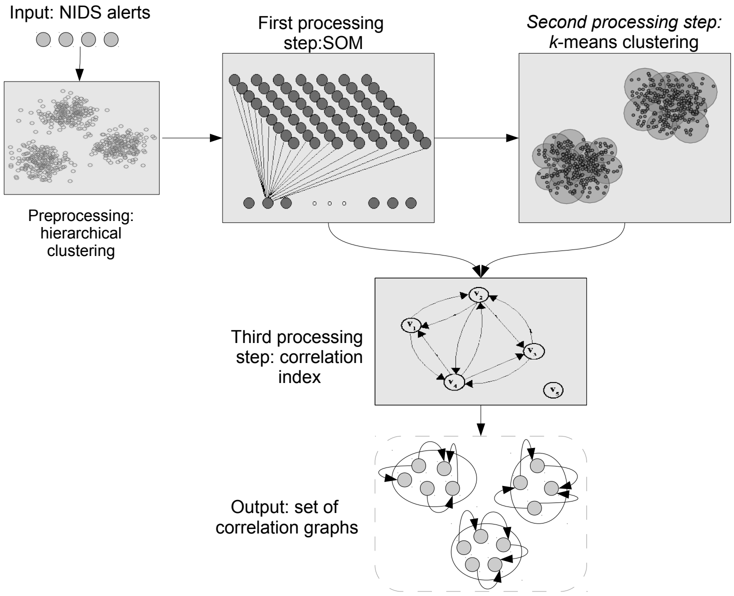
Input: NIDS alerts
First processing step:SOM
Second processing step: *k*-means clustering
Preprocessing: hierarchical clustering
Third processing step: correlation index
Output: set of correlation graphs

Fig. 1. Main functional steps of the software architecture

most instances. The output of the proposed k-means clustering algorithm, described in Section 5, is a set of clusters each representing a likely attack scenario.

The third (and last) processing step takes as its input the output layer of the SOM and the results of the k-means, and then computes a correlation index between alert clusters that have been grouped within the same cluster by the k-means. The correlation index is based on the distance between the neurons on the output layer of the SOM, on the timing relationships between alerts and on their type. This algorithm can identify causal relationships between alerts by determining which of the two alerts occurred first, and whether historical data show that alerts of the same type usually occur one after another.

The final output of the proposed algorithm for multistep alert correlation is represented by a set of directed graphs. Each graph represents a different attack scenario, whose vertices are clusters of similar alerts. The directed edges represent relationships between different alert clusters that belong to the same scenario.

4 Self-Organizing Map

A Self-Organizing Map is an auto-associative neural network [5], that is commonly used to produce a low-dimensional (typically two-dimensional) representation of input vectors that belong to a high-dimensional input space, since input vectors are mapped to coordinates in the output layer by a neighborhood function that preserves the topological properties of the input space. Hence, input vectors that are close in the input space are mapped to near positions in the

output layer of the SOM. Given a SOM having K neurons in the input layer, and $M \times N$ on the output layer, the SOM is a completely connected network having $K \cdot M \cdot N$ links, so that each input neuron is connected to each neuron of the output layer. Each link from the input to the output layer has a weight that expresses the strength of that link. In the described implementation, each alert is modelled as a numerical normalized tuple provided to the input layer. As an example, the $i - th$ alert x_i is modelled as
$\mathbf{x}_i = (alertType_i, srcIP_i, dstIP_i, srcPort_i, dstPort_i, timestamp_i)$
where $alertType_i$ is the alert type as identified by Snort, $srcIP_i$ and dst_IP_i are its source and destination IP addresses, $srcPort_i$ and $dstPort_i$ are the source and destination port numbers, and $timestamp_i$ is the time at which the alert has been issued.

The first step initializes the weights of the links between the input and the output layer. Since the training algorithm is unsupervised, it is important to have an optimal weight initialization instead of a random initialization. The weight initialization algorithm used in this model is similar to that proposed in [11]. It involves a heuristics that aims to map on near points the items which are dimensionally or logically close, and on distant points the items which are dimensionally or logically distant. Let us consider the training space $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_h\}$ for our network. We pick up the two vectors $\mathbf{x}_a, \mathbf{x}_b \in \mathbf{X}$, with $\mathbf{x}_a = \{x_{a1}, ..., x_{aK}\}$ and $\mathbf{x}_b = \{x_{b1}, ..., x_{bK}\}$ having the maximum K-dimensional Euclidean distance. The values of $\mathbf{x}_a$ and $\mathbf{x}_b$ are used for initializing the vectors of weights on the lower left, $\mathbf{w}_{M1} = \mathbf{x}_a$, and the upper right corner, $\mathbf{w}_{1N} = \mathbf{x}_b$. The idea is to map the most distant items on the opposite corners of the output layer. The values of the upper left corner weights vector $\mathbf{w}_{11}$ are then initialized by picking up the vector $\mathbf{x}_c \in \mathbf{X} - \{\mathbf{x}_a, \mathbf{x}_b\}$ having the maximum distance from $\mathbf{x}_a$ and $\mathbf{x}_b$. Finally, the vector $\mathbf{x}_d \in \mathbf{X} - \{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c\}$ having the maximum distance from $\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c$ initializes the values of the bottom right corner $\mathbf{w}_{MN}$. The weights of the remaining neurons on the four edges are initialized through linear interpolations. This heuristic-based initialization strategy for the SOM reduces the number of steps to reach a good precision, and improves the accuracy of the network with respect to a random initialization scheme [11].

After the initialization of the weights, the network undergoes unsupervised and competitive training by using the same training set $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_h\}$ used for the initialization. For each training vector $\mathbf{x}_i \in \mathbf{X}$, $\forall i = 1, ..., h$, the algorithm finds the neuron that most likely "represents" it, that is, the neuron having the weights vector $\bar{\mathbf{w}}$ with the minimum distance from $\mathbf{x}_i$. At the t-th learning step, the map weights are updated according to the following relation:

$$\mathbf{w}_{jk}(t) = \mathbf{w}_{jk}(t-1) + \delta(\bar{\mathbf{w}}, \mathbf{w}_{jk})\alpha(t)\left(\mathbf{x}_i - \mathbf{w}_{jk}(t-1)\right) \tag{1}$$

for $j = 1, ..., M,\ k = 1, ..., N$. The value of the function δ is inversely proportional to the distance between the two neuron weights taken as their arguments. Considering two neurons with coordinates (x, y) and (i, j), for the purposes of this paper we use the following δ function:

$$\delta(\mathbf{w}_{xy}, \mathbf{w}_{jk}) = \frac{1}{(|x-j| + |y-k|)^4 + 1} \tag{2}$$

The following step consists in the definition of a *learning rate* for the network expressed in function of the learning step t. In many SOM applications, this value is high at the beginning of the learning phase, when the network is still more prone to errors, and it decreases monotonically as the learning phase continues. However, as discussed in [14], this approach makes the learning process too much dependent on the first learning vectors, and not suited to the high variable context of NIDS alert analysis.

To mitigate this issue, in this paper we use a learning rate function $\alpha(t)$ close to that proposed in [14]:

$$\alpha(t) = \frac{t}{T} \exp\left(1 - \frac{t}{T}\right) \tag{3}$$

T is a parameter that expresses *how fast* the learning rate should tend to 0. A low value of T implies a faster learning process on a smaller training set, while a high value implies a slower process on a larger training set. The parameter T is computed as a function of two user-defined parameters: C and t_c. C represents the value under which the learning phase feedback becomes negligible (cutoff threshold). t_c is computed by solving the equation $\alpha(t_c) = C$. It represents the number of learning steps before the cutoff threshold is reached, i.e. the number of steps after which the SOM network becomes nearly insensitive to further learning feedbacks. The parameter T in $\alpha(t)$ representation given t_c, is computed by solving $\alpha(t_c) = C$. The SOM is re-trained at regular intervals. The training phase, that is computationally expensive, is executed in a separate thread, as discussed in Section 7.

5 Clustering algorithm

The next step is to apply a k-means clustering algorithm on the SOM output layer to extract possible information over distinct *attack scenarios*. Hence, we need to initialize k centers, for a fixed value of k. As the first two centers we choose the two points in the data set having the maximum euclidean distance. The third center is chosen as the point having the maximum distance from these two centers, the fourth center as the point having the maximum distance from the three centers chosen so far, and so on. After the initialization step, each element in the data set D chooses the centers closest to it as the identifier of its cluster. A well known drawback of the k-means algorithm consists in the necessity to fix the value of k before clustering. To limit this risk, we use the *Schwarz criterion* [9] as our heuristics for computing the best k. In particular, for $1 \leq k \leq |D|$, we compute several heuristic indexes that express, for that value of k, how large is the distortion value for that k, computed as the average distance between each point and its associated center. The best value of k, k^*, is the one having the minimum distortion. This approach is able to autonomously adapt to the heterogeneity of the data set at the price of a higher computational cost.

6 Correlation index

In the last processing step we compute correlations among alert clusters belonging to the same scenario. The correlation index between two alert clusters A_i and A_j, whose output neurons have coordinates $(x[A_i], y[A_i])$ and $(x[A_j], y[A_j])$ is computed as a function of the Euclidean distance between these two neurons. The correlation value between two alerts is always normalized in $[0, 1]$.

This definition of correlation is commutative, and it does not express any causality correlation. As we want to express the *direction* of the causality between two alerts, a of type A and b of type B with a relatively high correlation coefficient, we pursue the following approach. If $t[b] > t[a]$, i.e. the alert b was raised *after* the alert a, and we have no historical information over time relations between alerts of type A and alerts of type B, then $a \longrightarrow b$, that is the alert a was likely to generate the alert b in a specific attack scenario. If we have logged information over alerts of both types A and B, and the number of times that an alert of type B occurred *after* an alert of type A (in a specified time window) is greater than the number of times when the opposite event occurred, then $a \longrightarrow b$.

The weight of this correlation value (denoting its accuracy), is computed as a function of the number of alerts in the alert log $\mathcal{L}$ used to train the SOM. In particular, we want to keep a relatively low weight (close to 0) when the size of the alert log used for training the network is small, and to increase it when the system acquires new alerts. Given an alert log of size $x = |\mathcal{L}|$, we compute the weight $w(x)$ of the SOM-based index through a monotonically increasing function in $[0, 1]$. In particular, we will use the *hyperbolic tangent* tanh as the weight function:

$$w(x) = \tanh\left(\frac{x}{K}\right) = \frac{e^{\frac{x}{K}} - e^{-\frac{x}{K}}}{e^{\frac{x}{K}} + e^{-\frac{x}{K}}} \tag{4}$$

The parameter K determines how fast we want the algorithm to tend to 1. It is set as a function of the parameter x_M denoting the size of the alert log $\mathcal{L}$ in order to have $w(x_M) = M$. In this paper we use $M = 0.95$.

After having computed the correlation coefficients for each pair of alerts, we consider two alerts A_i, A_j actually correlated only if $Corr(A_i, A_j) \geq \mu_{corr} + \lambda\sigma_{corr}$ where μ_{corr} is the average correlation value and σ_{corr} is its standard deviation. $\lambda \in \mathbb{R}$ denotes how far from the average we want to shift in order to consider two alerts as correlated. Feasible values of λ for our purposes are in the interval $\lambda \in [0, 2]$. For $\lambda \simeq 0$ we consider as correlated all the pairs of alerts having a correlation value higher than the average one. This usually implies a relatively large correlation graph. A value of $\lambda \simeq 2$ brings to a smaller correlation graph, that only contains the most strongly correlated pairs of alerts. Higher values for λ could result in empty or poorly populated correlation graphs, since the correlation constraint could be too strict.

7 Experimental results

We carried out several experiments using a prototype implementation of the proposed multistep alert correlation architecture. The software has been mainly developed in C, as a module for Intrusion Detection System software *Snort*. A preliminary set of experiments, aiming to verify all the funcionalities of our software, were conducted using small-scale traffic traces, including attack scenarios performed within controlled network environment. Extensive experimental evaluations have then been carried out against the *Capture the Flag* 2010 dataset, that includes 40 GB of network traffic in PCAP format and is publicly available [3]. Our goals are to demonstrate that the computational cost of the proposed solution is compatible with live traffic analysis and to verify the capability of the system to recongize and correlate alerts belonging to the same attack scenario.

To achieve high performance, several different processing steps have been implemented as concurrent threads, that run in parallel with traffic analysis. In particular, alert clustering, training of the SOM and the periodic evaluation of the best value of k of the k-means algorithm, are performed in the background. As soon as the new weight of the SOM or the new best value of k are computed, they are substituted to the previous values. This design choice allows us to leverage multi-core CPUs to perform these expensive operations with no impacts on the performance of Snort. The two lines in Figure 2 show the memory usage of Snort while analyzing a 200MB traffic trace with and without or multistep alert correlation module. When our module is active, the memory usage of Snort is slightly higher. However, in both the cases the analysis of the traffic dump requires about 42 seconds, hence or module has no noticeable impact on the time required to analyze the traffic trace. Thanks to the self-adaptive choices of the parameters, our framework can easily adapt to heterogeneous traffic traces without the need for user-defined static configurations. The first alert clustering phase [4] is performed using the average heterogeneity of the alerts as grouping measure. The SOM distances depend on the size of the SOM network itself, but a greater size means a higher average distance, that anyway does not impact on the relative normalized distance values. The k-means clustering uses Schwarz criterion as heuristic for computing the best number of data clusters.

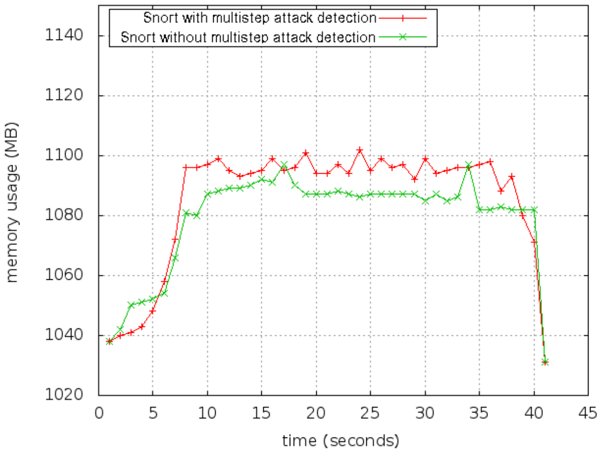

Fig. 2. Memory usage of Snort with and without multistep attack detection

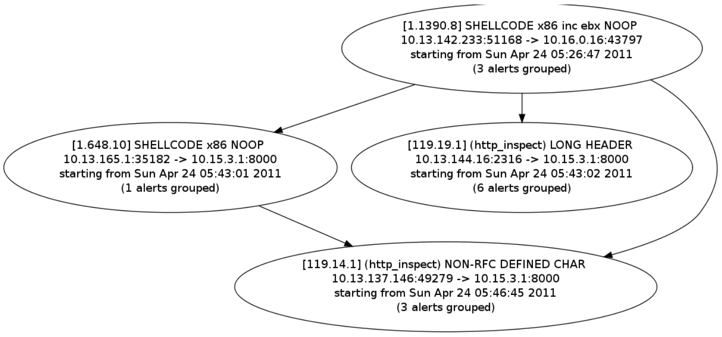
[1.1390.8] SHELLCODE x86 inc ebx NOOP
10.13.142.233:51168 -> 10.16.0.16:43797
starting from Sun Apr 24 05:26:47 2011
(3 alerts grouped)
[1.648.10] SHELLCODE x86 NOOP
10.13.165.1:35182 -> 10.15.3.1:8000
starting from Sun Apr 24 05:43:01 2011
(1 alerts grouped)
[119.19.1] (http_inspect) LONG HEADER
10.13.144.16:2316 -> 10.15.3.1:8000
starting from Sun Apr 24 05:43:02 2011
(6 alerts grouped)
[119.14.1] (http_inspect) NON-RFC DEFINED CHAR
10.13.137.146:49279 -> 10.15.3.1:8000
starting from Sun Apr 24 05:46:45 2011
(3 alerts grouped)

Fig. 3. Example of a correlation sub-graph

The correlation index among clustered alerts, is sensitive to the choice of the parameter λ. Feasible values are in the interval $\lambda \in [0, 2]$. A value of λ close to zero produces a graph that contains many alert correlations (all the ones having a correlation value greater than the average one). This is useful when the user wants to know all the likely correlations. A value closer to 2 highlights the few "strongest" correlations, having high correlation values. For example, by setting $\lambda = 1.7$, we obtain 4 different scenarios. Due to space limitation, only one of them is shown in Figure 3. It contains a likely shellcode execution (the grouping of several similar alerts inside of the same graph node is performed through Julisch method described in [4]) linked to an HTTP LONG HEADER alert and to a NON-RFC DEFINED CHAR alert raised by the `http_inspect` module towards the same host and port in the same time window.

On the other hand, by setting $\lambda = 1.2$ we obtain a larger number of likely scenarios, that can also be connected among them. The XML file[1] containing the 37 alert clusters that represent different attack scenarios and the corresponding correlation graphs [2] cannot be included in this paper for space limitations and are available online.

8 Conclusion

This paper presents a novel multistep alert correlation algorithm that is able to group security alerts belonging to the same attack scenario and to identify correlations and casual relationships among intrusion activities. The input of the proposed algorithm is represented by security alerts generated by a signature-based NIDS, such as Snort. Viability and efficacy of the proposed multistep alert correlation algorithm is demonstrated through a prototype, tested against recent and publicly available traffic traces. Experimental results show that the proposed multistep correlation algorithm is able to isolate and correlate intrusion activities belonging to the same attack scenario, thus helping security administrator in the

[1] Available online at `http://cris.unimore.it/files/attack_scenarios.xml`

[2] Available online at `http://cris.unimore.it/files/correlation_graph.png`

analysis of alerts produced by a NIDS. Moreover, by leveraging modern multi-core architectures to perform training in parallel and non-blocking threads, the computational cost of our prototype is compatible with live traffic analysis.

Acknowledgments

The authors acknowledge the support of MIUR-PRIN project DOTS-LCCI "Dependable Off-The-Shelf based middleware systems for Large-scale Complex Critical Infrastructure".

References

1. Chen, Z.G., Zhang, G.H., Tian, L.Q., Geng, Z.L.: Intrusion detection based on self-organizing map and artificial immunisation algorithm. Engineering Materials 439(1), 29–34 (2010)
2. Colajanni, M., Marchetti, M., Messori, M.: Selective and early threat detection in large networked systems. In: Proc. of the 10th IEEE International Conference on Computer and Information Technology (CIT 2010) (2010)
3. Capture the flag traffic dump, available online at `http://www.defcon.org/html/links/dc-ctf.html`
4. Julisch, K.: Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security 6, 443–471 (2003)
5. Kohonen, T.: The self-organizing map 78(9) (1990)
6. Mukkamala, S., Janoski, G., Sung, A.: Intrusion detection using neural networks and support vector machines. In: Proceedings of the 2002 International Joint Conference on Neural Networks (2002)
7. Munesh, K., Shoaib, S., Humera, N.: Feature-based alert correlation in security systems using self organizing maps. In: Proceedings of SPIE, the International Society for Optical Engineering (2009)
8. Patole, V.A., Pachghare, V.K., Kulkarni, P.: Article: Self organizing maps to build intrusion detection system. International Journal of Computer Applications 1(7), 1–4 (February 2010)
9. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. In: Proc. of the 17th International Conference on Machine Learning. pp. 727–734. Morgan Kaufmann (2000)
10. Snort home page, available online at `http://www.snort.org`
11. Su, M.C., Liu, T.K., Chang, H.T.: Improving the self-organizing feature map algorithm using an efficient initialization scheme. Tamkang Journal of Science and Engineering 5(1), 35 – 48 (2002)
12. Valeur, F., Vigna, G., Kruegel, C., Kemmerer, R.A.: A comprehensive approach to intrusion detection alert correlation. IEEE Transactions on Dependable and Secure Computing 1, 146–169 (2004)
13. Vokorokos, L., Baláz, A., Chovanec, M.: Intrusion detection system using self organizing map 6(1) (2006)
14. Yoo, J.H., Kang, B.H., Kim, J.W.: A clustering analysis and learning rate for self-organizing feature map. In: Proc. of the 3rd International Conference on Fuzzy Logic, Neural Networks and Soft Computing (1994)