# Framework and Models for Multistep Attack Detection

Mirco Marchetti, Michele Colajanni, Fabio Manganiello

*Department of Information Engineering*
*University of Modena and Reggio Emilia*
*{mirco.marchetti, michele.colajanni, fabio.manganiello}@unimore.it*

***Abstract***

*Cyber attacks are becoming increasingly complex, especially when the target is a modern IT infrastructure, characterized by a layered architecture that integrates several security technologies such as firewalls and intrusion detection systems. These contexts can be violated by a multistep attack, that is a complex attack strategy that comprises multiple correlated intrusion activities. While a modern Intrusion Detection System detects single intrusions, it is unable to link them together and to highlight the strategy that underlies a multistep attack. Hence, a single multistep attack may generate a high number of uncorrelated intrusion alerts. The critical task of analyzing and correlating all these alerts is then performed manually by security experts. This process is time consuming and prone to human errors. This paper proposes a novel framework for the analysis and correlation of security alerts generated by state-of-the-art Intrusion Detection Systems. Our goal is to help security analysts in recognizing and correlating intrusion activities that are part of the same multistep attack scenario. The proposed framework produces correlation graphs, in which all the intrusion alerts that are part of the same multistep attack are linked together. By looking at these correlation graphs, a security analyst can quickly identify the relationships that link together seemingly uncorrelated intrusion alerts, and can easily recognize complex attack strategies and identify their final targets. Moreover, the proposed framework is able to leverage multiple algorithms for alert correlation.*

***Keywords:*** *Multistep attacks, alert correlation, Self-Organizing-Maps, machine learning, network intrusion detection.*

## 1. Introduction

Nowadays, several technologies and architectures can be used to improve the security of networked information systems. Current best practices require layered defense infrastructures, where firewalls and Network Intrusion Detection Systems (NIDS) represent a cornerstone in any modern architecture for information security. These networked systems are vulnerable to multistep attacks, that are intrusion activities that leverage a chain of attacks carried out against different components of the same infrastructure. As modern information systems become more complex, network topologies are usually fragmented in different interconnected subnetworks. Hence, the target of an attack may be not directly reachable from the public Internet. In this scenario attackers are forced to compromise several elements of the network infrastructure. These elements are then used as stepping stones to reach the final target of the attack. As a consequence, a single multistep attack usually generates several intrusion alerts. The problem with existing technologies is that alert analysis and correlation require the

intervention of a security expert. This is a time consuming activity that is prone to human errors.

The first contribution of this paper is a new framework for the automatic analysis and correlation of security alerts generated by state-of-the-art NIDS. The goal of the proposed framework is to help a security analyst in recognizing and correlating the intrusion activities that are part of the same multistep attack. The output of the proposed framework is represented by a directed graph that represents a likely multistep attack. The nodes of the graph are the intrusion alerts generated by the NIDS, while the directed vertexes represent causal relationships among them. By looking at a correlation graph, a security expert can immediately recognize the relationships that link together different intrusion activities, and can easily identify the final target of the whole attack strategy without loosing precious time on the analysis of individual alerts and false positives.

A second contribution is represented by the inclusion in the proposed framework of two algorithms for alert analysis and correlation, based two different unsupervised machine-learning techniques: the former algorithm leverages Self-Organizing-Maps [1]; the latter algorithm, called *pseudo-Bayesian alert correlation*, is inspired to Bayes theorem of conditional probability[2]. These algorithms are included in the proposed prototype, and combined to produce correlated alert graphs.

This paper is organized as follows. Section 2 presents the architecture of the proposed framework for alert correlation. The two main algorithms for alert correlation on which our reference implementation is based, pseudo-Bayesian probability and Self-Organizing maps, are described in Section 3 and Section 4, respectively. Section 5 presents the experimental evaluation carried out through our prototype implementation. Related work is discussed in Section 6, while conclusions are outlined in Section 7.

## 2. Framework Architecture

The high-level architecture of the proposed framework for security alert correlation is shown in

Figure 1. Our framework comprises three main processing steps: *hierarchical clustering*, *alert correlation algorithms*, and *dynamic threshold*.

Hierarchical clustering is the first processing step, and operates directly on the unmodified security alerts that are generated by a NIDS. Alerts are grouped according to a clustering hierarchy in a way similar to [3]. 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 alert analysis and correlation process. Moreover, it groups many false positives in the same cluster, thus simplifying human inspection of security alerts. In our approach, clustered intrusion alerts are modeled as numerical normalized tuples, where each element represents a given property of the alert cluster (timestamp, type of alert, source and destination IP addresses, source and destination port numbers).

Clustered alerts are then analyzed by the alert correlation algorithms included in the framework. We remark that the proposed framework is modular and extensible, and makes it possible to easily insert or remove alert correlation algorithms. In the reference implementation of our framework, that has been used for experimental validation (see Section 5) we included two different algorithms for alert correlation. The first one, based on pseudo-Bayesian probability, is described in Section 3. The second one, based on Self-Organizing-Maps, is described in Section 4.

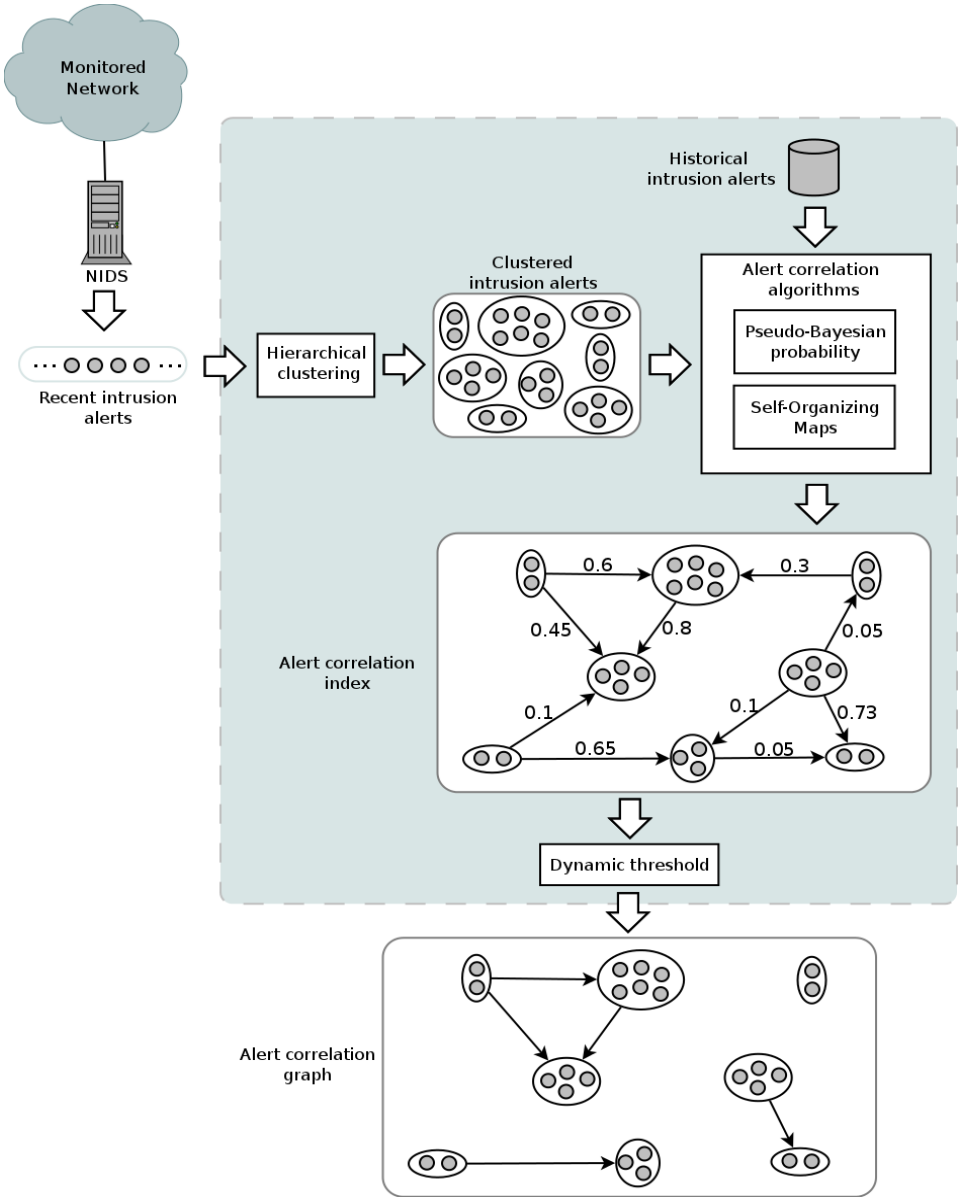



**Figure 1: Architecture of the Framework for Alert Correlation**

All the alert correlation algorithms need to expose the same programming interface towards the proposed framework. In particular, they require two different inputs: the clustered alerts as generated by the hierarchical clustering processing phase and past intrusion alerts generated by the NIDS. Past intrusion alerts are accessible through the database of historical intrusion alerts (see Figure 1). Each alert correlation algorithm analyzes these inputs and generates as its output a weighted and directed graph. The nodes are the intrusion alert clusters generated by the hierarchical clustering. Vertices link together clusters of intrusion alerts that are likely to belong to the same multistep attack scenario. The weight of each vertex is proportional to the strength of the correlation among the connected clusters, while the direction of the vertex expresses the causal relationship among the alert clusters. The absence of a vertex between two alert clusters means that they are not correlated (meaning

that they do not belong to the same multistep attack scenario) and is equivalent to the presence of a vertex having zero weight.

Since the alert correlation algorithms are different, in most cases they generate different correlation graphs. The graph produced as the output of all the alert aggregation algorithms (called *alert correlation index* in

Figure 1) is computed by merging the intermediate graphs obtained by all the alert correlation algorithms. In the current implementation, the weight of each vertex in the *alert correlation index* graph is computed as the average of the weights that the same vertex has in the graphs produced by the two correlation algorithms.

The last processing step, called *dynamic threshold* takes the *alert correlation index* as its input and prunes all the vertexes whose weight is relatively low with respect to the average weight of all the vertexes. This process has two purposes. First of all, it helps removing false correlations that are typically characterized by low correlation weights. Moreover, by preserving only the correlations that are characterized by a high weight, dynamic threshold highlights only the alert instances that are most likely to belong to the same multistep attack scenario. This process helps the security analyst to quickly identify a multistep attack among the multitude of alerts generated by the NIDS. The dynamic threshold algorithm removes from the alert correlation index graph all the vertexes whose weight *v* does not satisfy the following condition:

$$v \geq \mu_{corr} + \lambda\sigma_{corr} \qquad (1)$$

where $\mu_{corr}$ is the average value of the weight of all the vertexes in the graph, and $\sigma_{corr}$ is its standard deviation. The user-defined parameter $\lambda \in \mathbb{R}$ denotes how far from the average we want to shift in order to consider two alert clusters as correlated. For our purposes, the feasible values of $\lambda$ are in the range [0, 2]. For low values of $\lambda$ we consider as correlated all the pairs of alert clusters having a correlation value higher than the average. This usually implies a large correlation graph, composed by many interconnected alert clusters. Higher values of $\lambda$ bring to a smaller correlation graph that only contains the most correlated pairs of alert clusters.

## 3. Pseudo-Bayesian Algorithm

The pseudo-Bayesian algorithm for computing the correlation index between clusters of NIDS alert is inspired to the Bayes law of conditional probability. The correlation index is computed on the basis of the alert types, their timestamps and on the knowledge base of historical intrusion alerts. In particular, alert history is analyzed periodically, while clusters of recent intrusion alerts are analyzed as soon as they are received.

In the context of a signature-based NIDS, different signatures trigger alerts of different types, and each alert contains an identifier of the signature that has been triggered. Let $A$ and $B$ be two types of alert generated by two different attack signatures. They can be modeled as sets of homogeneous alert instances. Let $a$ and $b$ be two alert instances of type $A$ and $B$, respectively. The two alert instances $a \in A$ and $b \in B$ are considered as correlated if both conditions are verified:

1. Previous history shows that there is a high *pseudo-Bayesian probability* that an alert of type $A$ is triggered shortly after an alert of type $B$. We denote the pseudo-Bayesian probability of $A$ given $B$ as $P(A|B)$.

2. The two alert instances have been generated within a short timeframe, where this is a configurable parameter.

These conditions reflect the underlying assumption that two intrusion activities that belong to the same multistep attack exhibit a strong temporal locality. While the second condition can be verified easily by comparing the timestamps associated with the two alert instances $a$ and $b$, the pseudo-Bayesian probability $P(A|B)$ can only be computed by analyzing the intrusion alerts that have been generated by the same NIDS.

Given a finite set of historical intrusion alert (alert history) $H$, and any two alert types $A = \{a_1, a_2, a_3, \cdots, a_m\}$ and $B = \{b_1, b_2, b_3, \cdots, b_n\}$, such as $A \subseteq H$ and $B \subseteq H$, we define the set $Corr(A, B)$ containing all the possible correlated instances of alert types $A$ and $B$, as

$$Corr(A, B) = \{(a, b) \in A \times B : |t_a - t_b| \leq T_{max}\} \quad (2)$$

where $t_a$ and $t_b$ are the timestamps of alert instances $a$ and $b$, respectively, and $T_{max}$ is the highest time threshold for considering the two alert instances correlated. If $t_a$ and $t_b$ differ for more than $T_{max}$, then $a$ and $b$ are considered not correlated, independently of the alert history.

We now define the set $A_{corr}$ as the subset of $A$ containing only elements that appear in $Corr(A, B)$

$$A_{corr} = \{a \in A \mid \exists\, b \in B : (a, b) \in A \times B\} \quad (3)$$

Similarly, we define the set $A_{not}$ as the subset of $A$ containing only elements that do not appear in $Corr(A, B)$

$$A_{not} = \{a \in A \mid \nexists\, b \in B : (a, b) \in A \times B\} \quad (4)$$

It is now possible to define the *pseudo-Bayesian probability of having an intrusion alert of type A given an intrusion alert of type B*, or $P(A|B)$ as follows:

$$P(A|B) = P\big(Corr(A, B)\big) - \frac{|A_{not}|}{|A|} \quad (5)$$

The last term of equation 5 represents the ratio between the number of alerts of type $A$ that are not correlated to any alert of type $B$ and the total number of alerts of type $A$. This quantity ranges from 0 (if all alerts of type $A$ are correlated to alerts of type $B$) to 1 (if no alert of type $A$ is correlated to any alert of type $B$), and it is subtracted to the value of $P(Corr(A, B))$. Hence, if the number of alerts of type $A$ that are not correlated to alerts of type $B$ is much larger than the number of correlated alerts, the pseudo-Bayesian probability of $A$ given $B$ is significantly decreased.

The other term of the equation 5, $P(Corr(A, B))$, is defined as follows:

$$P(Corr(A, B)) = \frac{1}{|Corr(A,B)|} \sum_{(a,b) \in Corr(A,B)} e^{-\frac{(t_a - t_b)^2}{K}} \quad (6)$$

Each term of the sum represents the time correlation between two alert instances $a$ and $b$, where $(a, b) \in Corr(A, B)$. The time correlation value decays exponentially as the difference between the timestamps of $a$ and $b$ increases. The decay is modulated by the parameter $K$. For low values of $K$ the decay is faster, and even small time differences result in low

correlation values. The parameter $K$is computed as a function of the parameter $T_{max}$, that was introduced in equation 2. We want to compute $K$ so that the value of the function

$$f(x) = e^{-\frac{|t_a - t_b|^2}{K}} \quad (7)$$

reaches an arbitrarily small *cutoff value* $C$ when $x = |t_a - t_b| = T_{max}$ . In our implementation $C = 0.01$, hence the correlation between two alert instances $a$ and $b$ whose timestamps differ by $T_{max}$ will be 1%. To compute $K$ we have to solve the following equation:

$$f(T_{max}) = C \rightarrow e^{-\frac{{T_{max}}^2}{K}} = C \quad (8)$$

that is an exponential equation in which $K$ is the only variable. By solving it, we obtain

$$K = -\frac{{T_{max}}^2}{logC} \quad (9)$$

The pseudo-Bayesian algorithm proposed in this section is unsupervised, and does not rely on a-priori knowledge about the nature of intrusion alerts or about the topology of the protected network. However, useful information about the likelihood of alert correlation is extracted from analysis of the historical intrusion alerts. Hence, the reliability of the pseudo-Bayesian algorithm is influenced by the completeness of this knowledge base, that can be approximated by the number of alerts that are known in the alert history. If the number of past intrusion alerts (that is, the cardinality of the set $H$, or $|H|$) is low, then the correlation indexes are computed on the basis of incomplete and possibly biased historical data. As $|H|$ increases, the correlation indexes becomes more reliable and less sensitive to perturbations caused by anomalous alert clusters.

To embed these results in our model, we multiply the value $P(A|B)$ by a weight factor $w$, that is close to zero for small values of $|H|$ and approaches asymptotically 1 as $|H|$ increases. For a given set of historical alerts $H$, $w$ is defined as $w = W(|H|)$, where $W(x)$ represents a suitable weight function. In our implementation we use the hyperbolic tangent ($tanh$) as the weight function:

$$W(x) = tanh\left(\frac{x}{k}\right) = \frac{e^{\frac{x}{k}} - e^{-\frac{x}{k}}}{e^{\frac{x}{k}} + e^{-\frac{x}{k}}} \quad (10)$$

The parameter $k$ allows us to tune the weight function by letting the user choose how fast it approaches 1 as $|H|$ grows. It is computed as a function of two parameters, $M$ and $x_M$. $M$ is an arbitrary value close to 1. In the proposed implementation $M = 0.95$. $x_M$ represents the value for which $W(x_M) = M$. By choosing $x_M$ and imposing $w = W(x_M) = M$, we obtain the following equation:

$$w = W(x_m) = M \rightarrow \frac{e^{\frac{x_M}{k}} - e^{-\frac{x_M}{k}}}{e^{\frac{x_M}{k}} + e^{-\frac{x_M}{k}}} = M \quad (11)$$

where $k$ is the only variable. In particular, let $t = x_M/k$. For a fixed value of $M$, an approximated solution $t_{sol}$ can be computed through numerical approximation. For $M =$

$0.95$, $t_{sol} \cong 1.83178$. The parameter $k$ can then be computed by solving the equation $k = x_M / t_{sol}$.

The output of this correlation algorithm is a graph that denotes how strong is the correlation likelihood among different alert clusters. Each alert cluster is represented by a node in the graph. If two alert clusters contain alert instances of type $A$ and $B$ that were generated within the same timeframe (of length $T_{max}$) and if the pseudo-Bayesian probability $P(A|B)$ is higher than 0, then the two alert clusters are connected by a vertex whose weight is $wP(A|B)$. The direction of the vertex is inferred by the timestamps that are associated with alert instances that are included in the two clusters. The vertex originates from the cluster that contains the earliest alert instance and points toward the other cluster. This graph is then merged with the analogous graph produced by the other correlation algorithms as described in Section 2.

## 4. Self-Organizing Maps

The second alert correlation algorithm included in the proposed framework is based on the application of a neural network called Self-Organizing Map. It consumes two inputs: clustered intrusion alerts produced by the hierarchical clustering algorithm shown in

Figure 1, and information about past alerts retrieved from the historical alert database. The whole correlation algorithm consists of three processing phases: SOM, k-means clustering and correlation, that are described in Sections 4.1, 4.2 and 4.3 respectively.

### 4.1. First Processing Phase: SOM

A Self-Organizing Map (SOM) is an auto-associative neural network [4] that is used to produce a low-dimensional (typically two-dimensional) representation of input vectors belonging to a high-dimensional input space. 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 $l$ neurons in the input layer, and $o \times p$ on the output layer (that is, a two-dimensional output layer having a height of $o$ and a width of $p$), the SOM is a completely connected network having $l \times o \times p$ 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 its strength.

Before the use of SOM to classify input alert clusters, it is necessary to initialize the weights of the links between the input and the output layer. Since the training algorithm is unsupervised, it is important to have an optimized weight initialization instead of a random initialization. The weight initialization algorithm used in this model is similar to the one already proposed in [5]. It involves a heuristic that aims to map on near points the items which are close in the input space, and on distant points the items which are distant in the input space.

Let us consider the set of $h$ input vectors $X = \{x_1, \cdots, x_h\}$, whose elements are D-dimensional vectors. We choose the two vectors $x_a = \{x_1^a, x_2^a, \cdots, x_D^a\}$ and $x_b = \{x_1^b, x_2^b, \cdots, x_D^b\}$ among those belonging to $X$ that have the maximum D-dimensional Euclidean distance:

$$x_a, x_b \in X \mid d(x_a, x_b) \geq d(x_i, x_j) \; \forall \; 1 \leq i, j \leq h \qquad (12)$$

where

$$d(x_i, x_j) = \sqrt{\sum_{q=1}^{D} \left(x_q^i - x_q^j\right)^2} \quad (13)$$

The two vectors $x_a$ and $x_b$ are used to initialize the vectors of weights on the lower left, $n_1^1 = x_a$, and upper right $n_p^o = x_a$ corners of the output layer of the SOM. The idea is to map the two most distant input vectors on the two opposite corners of the output layer. The values of the upper left corner $n_1^o$ are initialized by picking the input vector $x_c \in X - \{x_a, x_b\}$ having the maximum distance from $x_a$ and $x_b$. Finally, the input vector $x_d \in X - \{x_a, x_b, x_c\}$ having the maximum distance from $x_a$, $x_b$ and $x_c$ initializes the values of the bottom right corner $n_p^1$.

We then initialize the neurons on the four edges of the output layer through the following linear interpolations:

$$n_j^1 = \frac{j-1}{p-1} n_p^1 + \frac{p-j}{p-1} n_1^1, \quad 2 \le j \le p-1 \quad (14)$$

$$n_j^o = \frac{j-1}{p-1} n_p^o + \frac{p-j}{p-1} n_1^o, \quad 2 \le j \le p-1 \quad (15)$$

$$n_1^i = \frac{i-1}{o-1} n_1^i + \frac{o-i}{o-1} n_1^1, \quad 2 \le i \le o-1 \quad (16)$$

$$n_p^i = \frac{i-1}{o-1} n_p^o + \frac{o-i}{o-1} n_p^1, \quad 2 \le i \le o-1 \quad (17)$$

The remaining neurons are initialized according to the following two-dimensional linear interpolation:

$$n_j^i = \frac{(j-1)(i-1)}{(p-1)(o-1)} n_p^o + \frac{(j-1)(o-i)}{(p-1)(o-1)} n_p^1 + \frac{(p-j)(i-1)}{(p-1)(0-1)} n_1^o + \frac{(p-j)(o-i)}{(p-1)(o-1)} n_1^1 \quad (18)$$

This heuristic-based initialization strategy for the SOM reduces the number of learning steps that are necessary to reach a good precision, and improves the accuracy of the network with respect to a random initialization scheme [5].

After the weight initialization, the network undergoes a phase of unsupervised and competitive training by using the same training set $X = \{x_1, \cdots, x_h\}$ as that of the initialization phase. For each input vector $x_l$, the learning process finds the output neuron $n_{best}$that has the minimum distance from $x_l$:

$$d(x_l, n_{best}) \le d(x_l, n_j^i) \; \forall \, i, j : \; 1 \le i \le o \, , 1 \le j \le p \quad (19)$$

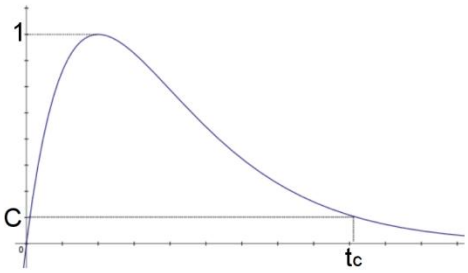



**Figure 2: Plot of the Learning Rate Function**

where $d$ is the Euclidean distance defined in equation 13, and $n_{best}$ is the "best" output neuron for the input vector $x_l$. At the *t-th* learning step, the weights of the SOM are updated according to the following relation:

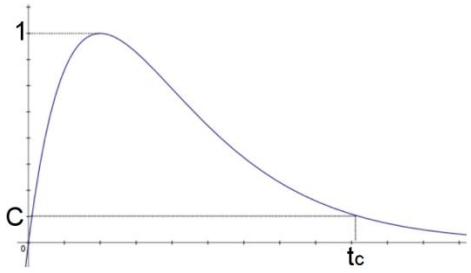
$$n_j^i(t) = n_j^i(t-1) + \delta\left(n_{best}, n_j^i\right)\alpha(t)\left[x_l - n_j^i(t-1)\right] \qquad (20)$$

The value of the function $\delta$ is inversely proportional to the distance between the two neurons taken as arguments. Considering two neurons $n_j^i$ and $n_m^l$ with coordinates $(i,j)$ and $(l,m)$, respectively, $\delta$ can be defined as:

$$\delta\left(n_j^i, n_m^l\right) = \frac{1}{(|i-l|+|j-m|)^4+1} \qquad (21)$$

The term in round parenthesis at the denominator of equation 21 is the *Manhattan distance* between the coordinates of the two neurons.

The next step is the definition of a *learning rate* for the SOM, expressed as a function of the learning step $t$. In many SOM applications, this value is high at the beginning of the learning phase, and it decreases monotonically as the learning phase continues. However, as discussed in [6], this approach makes the learning process too sensitive with respect to the first learning vectors, and not suitable to the high variable context of NIDS alert analysis. To mitigate this issue, in this paper we use a learning rate function $\alpha(t)$ that is close to that proposed in [6], and shown in Figure 2:




$$\alpha(t) = \left(\frac{t}{T}\right)^{\left(1-\frac{t}{T}\right)} \qquad (22)$$

$T$ is a parameter denoting *how fast* the learning rate should tend to 0. A small value of $T$ implies a faster learning process on a smaller training set, while a large value of $T$ implies a slower learning process on a larger training set. $T$ is computed as a function of two user-defined parameters: $C$ and $t_c$. $C$ represents the *cutoff* threshold for the learning process, that is

the value below which the feedback of the learning process becomes negligible. $t_c$ represents the number of learning steps that should be executed before the learning rate reaches the cutoff value $C$, that is the number of learning steps after which the SOM becomes nearly insensitive to further learning feedback.

For given values of $C$ and $t_c$, it is possible to compute $T$ in equation 22 by imposing $\alpha(t_c) = C$, which implies

$$\frac{t_c}{T} + log(T) = log\left(\frac{et_c}{C}\right) \tag{23}$$

This is a logarithmic equation having $T$ as the only variable. Its approximated solution is

$$T = \frac{et_c}{C} W_{-1}\left(-\frac{C}{e}\right) \tag{24}$$

The term $W_{-1}$ is the analytic continuation of the Lambert W function. An approximation of this function can be computed through the Taylor series, as proposed in [7]:

$$W_{-1}(z) = \sum_{k=0}^{+\infty} \mu_k \, r^k \tag{25}$$

where

$$r = -\sqrt{2(ez+1)} \tag{26}$$

$$\mu_k = \frac{k+1}{k+1}\left(\frac{\mu_{k-2}}{2} + \frac{\alpha_{k-2}}{4}\right) - \frac{\alpha_k}{2} - \frac{\mu_{k-1}}{k+1} \tag{27}$$

$$\alpha_k = \sum_{j=2}^{k-1} \mu_j \mu_k - j + 1 \tag{28}$$

$$\mu_0 = -1, \mu_1 = 1, \alpha_0 = 2, \alpha_1 = -1 \tag{29}$$

The first terms of the series of equation 25 are:

$$W_{-1}(z) = -1 + r - \frac{1}{3}r^2 + \frac{11}{72}r^3 - \frac{42}{540}r^4 + \frac{769}{17280}r^5 - \cdots \tag{30}$$

The SOM is retrained at regular intervals. The training phase, that is computationally expensive, can be executed in a separate thread. On multi-core machines this design choice allows us to execute training in parallel with analysis, and to retrain the SOM at short time intervals (in the order of 30 to 60 seconds). After the first learning phase, the SOM is ready to associate the clustered alerts provided as numerical normalized tuples by the hierarchical clustering algorithm shown in

Figure 1.

### 4.2. Second Processing Phase: k-means Clustering

The next step if the SOM correlation algorithm is to apply the k-means clustering to the output layer of the SOM. K-means is used to recognize different attack scenarios, under the assumption that alert clusters that belong to the same attack scenarios are mapped to nearby neurons in the output layer by the SOM. Hence, we need to initialize *k* centers, for a given value of $k$.

Let $o \times p$ be the size of the output layer of the SOM, and $D$ the dataset containing all the coordinate pairs $(i,j), 1 \leq i \leq o, 1 \leq j \leq p$, that identify output neurons to which is associated at least one alert cluster. Let $n_\mu = (i_\mu, j_\mu)$ and $n_\nu = (i_\nu, j_\nu)$ be the two neurons belonging to $D$ that have the maximum Euclidean distance. These two points of the output layer of the SOM identify the first two centers for the k-means clustering algorithm. The third center is chosen as the neuron belonging to $D$ that has the maximum distance from $n_\mu$ and $n_\nu$, the fourth center as the neuron having the maximum distance from the centers chosen so far, and so on.

After this initialization step, each element in the dataset $D$ "chooses" the closest center as the identifier of its cluster. Therefore, at the *t-th* iteration, the set $S_i^t$, that contains the elements in $D$ that are associated to the $i_{th}$center, for $1 \leq i \leq k$, is defined as:

$$S_i^t = \{ n_j \in D \mid d(n_j, m_i^t) \leq d(n_j, m_l^t), 1 \leq l \leq k \} \quad (31)$$

where $m_i^t$ represents the coordinates of the $i_{th}$ center at the step $t$. Once all the elements of $D$ have been clustered, the centers of all the clusters are updated as the average of the points that belong to the cluster:

$$m_i^{(t+1)} = \frac{1}{|S_i^t|} \sum_{x_j \in S_i^t} n_j \quad (32)$$

This process is iterated until $m_i^t = m_i^{(t+1)}, 1 \leq i \leq k$, that is until the clustering converge and the centers become stable.

A well known drawback of the k-mean clustering algorithm is the need to set a fixed value of k beforehand. To solve this issue, we apply the *Schwarz criterion* [8] as a heuristic to compute the optimal number of clusters for our dataset. For all the possible values of $k, 1 \leq k \leq |D|$, we compute the *distortion index*, defined as the average distance between each point and its center:

$$dist_k = k\ log(n) + \sum_{x_j \in D} (n_j - m_j) \quad (33)$$

where $m_j$ is the center associated to the neuron $n_j$. The best value for $k$, denoted as $k^*$, is the one having the smaller distortion: $dist_{k^*} \leq dist_k, 1 \leq k \leq |D|$.

The Schwarz heuristic allows us to build a clustering algorithm based on a near-optimal number of clusters without having to rely on fixed constants or on user-provided parameters. This approach is able to adapt to the heterogeneity of the dataset, at the expense of a higher computational cost.

### 4.3. Third Processing Phase: Correlation

Alert cluster correlation is the last phase of the SOM algorithm for alert correlation. It takes as input the results of both the previous phases, and generates as output the correlation value among all the alert clusters that have been grouped within the same multistep attack scenario by the k-means clustering algorithm.

The correlation index between two alert clusters $I$ and $J$, mapped by the SOM on the output neurons $n_i$ and $n_j$ respectively, is always zero if $n_i$ and $n_j$ have been assigned to different clusters by the k-means algorithm. If $n_i$ and $n_j$ belong to the same cluster, the

correlation index between them is computed on the basis of the distance between the two neurons.

Let $n_i$ be the output neuron whose coordinates on the output layer of the SOM are $(x[n_i], y[n_i])$. Similarly, the coordinates of the neuron $n_j$ are $(x[n_j], y[n_j])$. The correlation between $n_i$ and $n_j$, $Corr(I,J)$, can then be computed as a function of the Euclidean distance among them (as measured on the output layer of the SOM), normalized over the maximum possible distance, that is the distance between two opposite corners of the output matrix.

$$Corr(I,J) = \begin{cases} 0 & \text{(34a)} \\ \dfrac{1}{1 + \sqrt{\left(x[n_i] - x[n_j]\right)^2 + \left(y[n_i] - y[n_j]\right)^2}} & \text{(34b)} \end{cases}$$

The case of the equation 34a is verified when the Euclidean distance between $n_i$ and $n_j$ is the same as the diagonal of the output layer of the SOM, that is, if the two alerts are mapped to the two neurons that have the maximum possible distance. If this is not the case, correlation between $n_i$ and $n_j$ is computed according to equation 34b. Correlation values are always normalized in the range [0,1].

The definition of correlation given in equation 34 is commutative, and does not express any causality relation. On the other hand, we want to leverage the direction of a vertex between two correlated alert clusters to express possible causal relationships. The direction is computed on the basis of the timestamps of the alerts that are included in the two alert clusters that are connected by a vertex, according to the following approach:

- If the timestamps of the alert instances included in the alert cluster *I* is smaller than the timestamps of the alert instances that are included in the alert cluster *J*, and if we have no historical information about timing relationships between the alert types included in *I* and in *J*, than we assume that the alert cluster *I* was likely to "cause" the alert cluster *J* in this specific attack scenario. We denote this concept with the notation $I \rightarrow J$.
- If the historical alert database already contains previous instances of alerts of the same type of those included in *I* and in *J*, and if the number of instances in which alerts of the same type of those included in *I* preceded those included in *J* is larger than the number of instances in which alerts of the same type of those included in *J* preceded those included in *I*, than we assume that the alert cluster *I* was likely to "cause" the alert cluster *J* in this specific attack scenario. We denote this concept with the notation $I \rightarrow J$.

Since this correlation algorithm is completely unsupervised, and it extracts useful information from the database that contains past intrusion alerts generated by the same NIDS, correlation results are influenced by the size of the alert history H.

To embed this notion in our model, we multiply the value $\mathrm{Corr(I, J)}$ by a weight factor w, that is close to zero for small values of $|\mathrm{H}|$ and approaches asymptotically 1 as $|\mathrm{H}|$ increases. For a given set of historical alerts H, w is defined as $\mathrm{w} = \mathrm{W}(|\mathrm{H}|)$, where $\mathrm{W(x)}$ represents a suitable weight function. In our implementation of the SOM correlation algorithm we use the same weight function that have already been presented in Section 3 for the pseudo-Bayesian correlation algorithm, and that is described in equations 10 and 11.

The correlation graph between alert clusters generated by this correlation algorithm is then merged with the correlation graphs generated by the other correlation algorithms included in the framework, as described in Section 2.

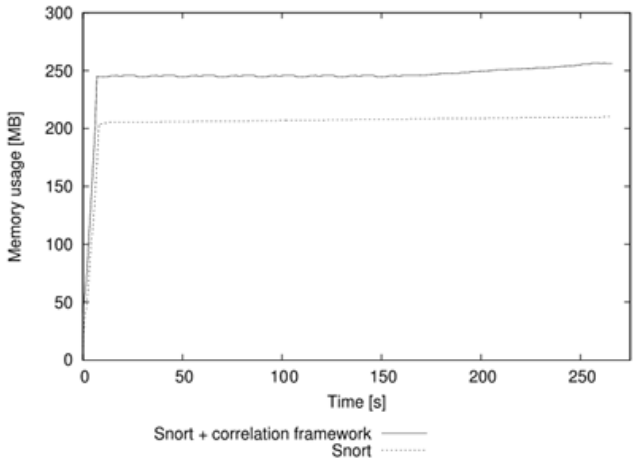



**Figure 3: Memory usage of snort with and without the proposed alert correlation and multistep attack detection**

## 5. Experimental Results

We carried out several experiments using a prototype implementation of the framework for alert correlation and multistep attack detection proposed in Section 2. The framework is mainly implemented in C as a module for the popular open source NIDS Snort (version 2.9 and later). It also includes a Web-based interface for the visualization of correlation graphs implemented in Perl, HTML and AJAX.

The goals of the experimental evaluation are twofold:

1. To demonstrate that the computational cost of the proposed solution is compatible with the temporal constraints of (soft) real-time traffic analysis.
2. To verify the capability of the proposed framework to recognize and correlate alert instances that belong to the same multistep attack scenario.

### 5.1. Performance Evaluation

The proposed alert correlation algorithms are based on unsupervised machine-learning techniques characterized by computationally expensive training processes. However, modern multi-core architectures can be used to perform training in concurrent threads with respect to alert analysis and correlation. Our prototype makes extensive use of concurrent threads to parallelize independent tasks. Training is always and continuously performed in the background, while alert analysis and correlation continue unaffected. In particular, since Snort is a monolithic process that only uses one core, all the other cores can be used to execute the other tasks required for alert correlation with no or minimal influence on Snort's performance.

Evaluation of the computational cost of the proposed framework was carried out by analyzing a PCAP traffic trace of 486 MB, taken from traffic traces recorded during the 2010 *Capture the Flag* competition (organized by the Computer Science Department of the University of California at Santa Barbara [9]). We first analyzed this traffic trace through a basic version of Snort and measured its execution time and its memory footprint. We then repeated the experiment through our modified version of Snort that includes the prototype implementation of the alert correlation framework.

Experimental results are show in

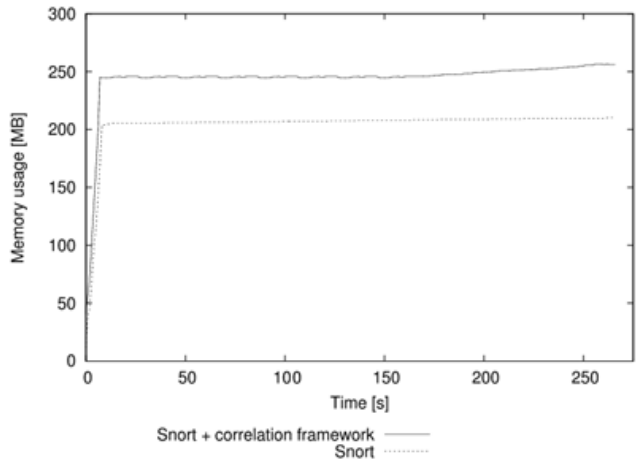



Figure 3. The solid line represents the memory usage of Snort augmented with the proposed framework, while the dashed line represents the memory usage of an unmodified

version of Snort. Experiments show that the alert correlation framework introduces a noticeable increase in memory consumption that still remains well within the capabilities of modern hardware. Moreover, the chart shows how both the modified and unmodified versions of Snort take exactly the same time (267 seconds) to analyze the traffic dump. Hence the proposed framework does not impact the processing time required by Snort, and it is compatible with live traffic analysis.

### 5.2. Alert Correlation and Multistep Attack Detection

Extensive evaluation of the proposed framework was carried out against the Capture the Flag (CTF) 2010 dataset. For these experiments we enabled both the pseudo-Bayesian and the SOM algorithms for alert correlation. Both algorithms include several heuristics that allow them to autonomously adapt to the characteristics of the input traffic traces, and that drastically reduce the number of user-defined configurations.

In particular:

- The first hierarchical clustering phase, shown in
- Figure 1, does not require static initialization, since the number of clusters is computed dynamically based on the average cluster heterogeneity.
- The results of the SOM are largely insensitive to the size of the SOM output layer. While larger output layers cause an increase in the average distance among the neurons, this increase does not impact the normalized correlation values.
- The k-means clustering algorithm used as second processing step by the SOM aggregation algorithm uses the Schwarz criterion to determine the optinal value of k.

For the pseudo-Bayesian correlation algorithm, we used $T_{max} = 1\ hour$ and $x_m = 1000\ alerts$.

The last processing step, called dynamic threshold in

Figure 1, is quite sensitive to the choice of parameter $\lambda$ in equation 1. As discussed in Section 2, feasible values for $\lambda$ are in the interval [0,2]. By setting $\lambda = 1.7$ our alert correlation framework identified four multistep attack scenarios, each characterized by a different correlation graph. Two examples of multistep attacks are shown in Figure 4 and Figure 5.

Figure 4 represents a complex attack scenario, in which several reconnaissance activities targeting the same subnet and executed within the same time window are correlated with an exploit attempt. Of particular relevance is the correlation between the two alert clusters at the bottom of Figure 4. These two alert clusters represent a portscan activity that has been successfully correlated with an alert raised after the detection of a shellcode. Portscans are performed to identify vulnerable machines and services, hence it is very likely that the attacker tried to exploit a vulnerable service through a shellcode injection only after having identified it through a portscan.

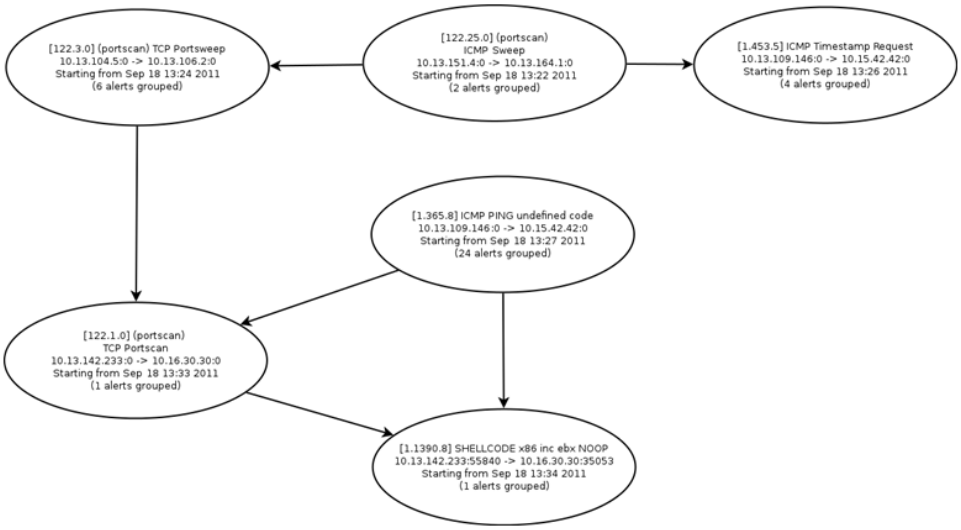



**Figure 4: Multistep attack scenario that correlates reconnaissance activities and exploitation through shellcode**

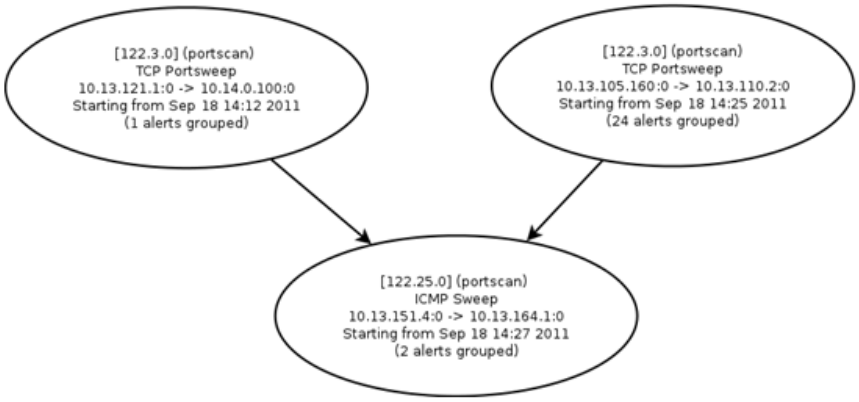



**Figure 5: Multistep attack scenario that correlates TCP and ICMP scanning activities**

Figure 5 represents a simpler attack scenario, in which our framework is able to to correlate reconnaissance activities that originates from the same subnet and that have been performed within a short time interval.

## 6. Related Work

The application of unsupervised machine-learning techniques such as neural networks and Bayesian networks to the field of intrusion detection is not novel. Several papers propose naïve Bayes classifiers, Bayesian networks, support vector machines [20], neural networks and self-organizing maps as the main detection engine for their intrusion detection systems.

In particular, naïve Bayes classifiers have been used in [10] and [11] for the implementation of anomaly-based NIDS. More complex anomaly-detection algorithms, based on complete Bayes networks, are proposed in [12] and [13]. Anomaly detection systems based on Self-Organizing Maps have also been described in [14], [15]. However, all existing approaches leverage machine-learning techniques to detect intrusions and anomalies, rather than trying to correlate them as we do in this paper.

Our work relates to other techniques for NIDS alert correlation. According to the comprehensive framework for NIDS alert correlation proposed in [16], our solution can be classified as a *multistep correlation* component. In this context, three interesting works are [17], [18], [19] and [3].

In [17], the authors propose an alert correlation algorithm based on naïve Bayesian networks. Their approach aims to predict the target of a multistep attack based on previous alert history. Their algorithm starts by identifying a set of possible intrusion objectives, and then it analyzes the historical data to build a naïve Bayes network for each intrusion objective. On the other hand, the work proposed in this paper does not aim to predict the future objectives of a possible multistep attack, but to identify and highlight groups of intrusion alerts that, based on their detection time and on the past alert history, are more likely to be correlated and to belong to the same multistep attack scenario.

In [18], the authors use Bayes networks to fuse the results obtained by several detection models, thus improving the overall detection rate of multi-model host IDS. While that work can be considered as an example of alert correlation, its purpose is to validate single IDS alerts, without trying to correlate different alerts.

The use of a SOM to compute the similarity between alerts generated by a NIDS was introduced in [19]. Our work differentiates from [19] for a twofold reason: our evaluation is not limited to the use of SOM to compute alert similarity, but the use of SOM is just one intermediate step of only one correlation algorithm included in our framework; moreover, with respect to [19] we propose an innovative initialization algorithm for SOM and an adaptive training strategy that makes SOM more robust with respect to perturbations in the training data (see Section 4.1).

In [3] security alerts generated by a NIDS are grouped through a hierarchical clustering algorithm. The classification hierarchy is defined by the user to aggregate 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 pre-processing step (Hierarchical clustering in

Figure 1). We then use the alert clusters generated by this hierarchical clustering algorithm as an input for all the correlation algorithms that are included in our framework. Hence we take advantage of the algorithm presented in [3], but the proposed framework and all the subsequent processing steps are quite novel.

## 7. Conclusion

This paper proposes a novel framework for the correlation of intrusion alerts generated by modern network intrusion detection systems, and for the identification of multistep attack scenarios. One of the main strength of the proposed framework is its ability to leverage multiple different correlation algorithms, thus achieving better results with respect to previous efforts in the same field.

The proposed framework is based on two completely unsupervised techniques borrowed from the machine-learning domain: Self-Organizing maps and pseudo-Bayesian correlation probability. These two correlation algorithms are modified for the inclusion as modules within our framework, that can easily be expanded through the addition of novel algorithms.

The performance of the proposed framework are evaluated experimentally through a prototype implementation based on Snort. Experimental results show that our solution is able to identify complex multistep attack scenario, and to represent them as directed graphs. This result is extremely useful to network analysts, that can focus immediately on the real target of complex multistep attack scenarios without losing focus in the analysis of a high number of individual alerts.

Moreover, our multithread implementation is able to leverage modern multi-core architectures to perform expensive learning tasks in parallel with traffic analysis and in a non-blocking fashion. Hence the proposed framework is compatible with real-time traffic analysis and has no noticeable impact on the time required by Snort to analyze network traffic.

## Acknowledgements

The authors acknowledge the support of MIUR-PRIN project DOTS-LCCI “Dependable Off-the-Shelf based middleware system for Large-scale Complex Critical Infrastructures”

## References

[1] F. Manganiello, M. Marchetti, and M. Colajanni, “Multistep attack detection and alert correlation in intrusion detection systems”, Proc. of the 5th International Conference on Information Security and Assurance, Brno, Czech Republic, Aug. 2011.

[2] M. Marchetti, M. Colajanni, F. Manganiello, “Identification of correlated network intrusion alerts”, Proc. of the 3rd International Workshop on Cyberspace Safety and Security, Milan, Italy, Sep. 2011.

[3] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis", ACM Transactions on Information and system security, Vol. 6, 443-471, 2003.

[4] T. Kohonen, P. Somervuo, "Self-organizing maps of symbol strings", Neurocomputing, Volume 21, Issues 1-3, Nov. 1998.

[5] M.C. Su, T.K. Liu, H.T. Chang, "Improving the self-organizing feature map algorithm using an efficient initialization scheme", Tamkang Journal of Science and Engineering, Volume 5, Issue 1, 2002.

[6] J.H. Yoo, B.H. Kang, J.W. Kim, "A clustering analysis and learning rate for self organizing feature map", Proc. of the 3rd International Conference on Fuzzy Logic, Neural Networks and Soft Computing, Fukuoka, Japan, Aug. 1994.

[7] F. Chapeau-Blondeau, "Numerical evaluation of the Lambert W function and application to generation of random gaussian noise with exponent 1/2", IEEE Transactions on Signal Processing, Volume 50, Issue 9, 2002.

[8] D. Pelleg, A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters", Proc. of the 17th International Conference on Machine Learning, Stanford, CA, USA, Jun. 2000.

[9] Capture the Flag traffic dump, available online at http://defcon.org/html/links/dc-ctf.html

[10] A. Valdes, K. Skinner, "Adaptive model-based monitoring for cyber attack detection", Proc. of the 3rd international workshop on Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France, 2000

[11] M. Panda, M.R. Patra, "Network intrusion detection using Naïve Bayes", International Journal on Computer Science and Network Security, Volume 7, Issue 12, 2007

[12] M. Mehdi, S. Zahir, A. Anou, M.Bensebti, "Bayesian networks in intrusion detection systems", Journal of Computer Science, Volume 3, Issue 5, 2007

[13] A. Cemerlic,, L. Yang, J. M. Kizza, "Network Intrusion Detection based on Bayesian Networks", Proc. of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08), Redwood City, CA, USA, 2008.

[14] V.A. Patole, V.K. Pachghare, P. Kulkarmi: "Self Organizing Maps to build intrusion detection systems", Journal of Computer Applications, Volume 1, Issue 7, Feb. 2010

[15] Z.G. Chen, G.H. Zhang, L.Q. Tian, Z.L. Geng, "Intrusion detection based on Self-Organizing map and artificial immunization algorithm", Engineering Materials, 2010

[16] F. Valeur, G. Vigna, C. Kruegel, R. A. Kemmerer, "A comprehensive approach to intrusion detection alert correlation," IEEE Transactions on Dependable and Secure Computing (TDSC), vol. 1, pp. 146–169, 2004.

[17] S. Benferhat, T. Kenaza, A. Mokhtari, "A naive bayes approach for detecting coordinated attacks," in Proc. of the 32nd IEEE International Annual Conference on Computer Software and Applications COMPSAC'08, 2008.

[18] C. Kruegel, D. Mutz, W. Robertson, F. Valeur, "Bayesian event classification for intrusion detection," in Proc. of the 19th Annual Computer Security Applications Conference (ACSAC '03), Las Vegas, NV, USA, 2003.

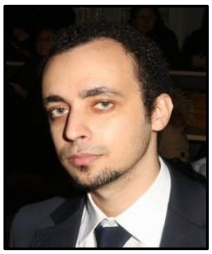
[19] K. Munesh, S. Shoaib, N. Humera, "Feature-based alert correlation in security systems using self organizing maps". Proc. of SPIE, the International Society for Optical Engineering, 2009

[20] S. Mukkamala, G. Janoski, A. Sung, "Intrusion detection using neural networks and support vector machines", Proc. of the 2002 International Joint Conference on Neural Networks, 2002

## Authors

**Mirco Marchetti** received his Ph.D. in Information and Communication T echnologies (ICT) in 2009. He holds a post-doc position at the Interdepartment Center for Research on Security and Safety (CRIS) of the University of Modena and Reggio Emilia. He is interested in intrusion detection and in all aspects of information security. Home page: http://weblab.ing.unimo.it/people/marchetti

**Michele Colajanni** is full professor in computer engineering at the University of Modena and Reggio Emilia since 2000. He received the Master degree in computer science from the University of Pisa, and the Ph.D. degree in computer engineering from the University of Roma in 1992. He manages the Interdepartment Research Center on Security and Safety (CRIS), and the Master in "Information Security: Technology and Law". His research interests include security of large scale systems, performance and prediction models, Web-based and cloud-based systems. Home page: http://weblab.ing.unimo.it/people/colajanni



**Fabio Manganiello** received his M.Sc. in Information Engineering from the University of Modena and Reggio Emilia in 2010. He currently works as software engineer at ION® Trading S.p.a. and collaborates with the university of Modena and Reggio Emilia. Former director of the computer security magazine "Hacker Journal". His interests range from intrusion detection to machine learning, from distributed systems to advanced techniques of software engineering