International Journal of Innovative Computing, Information and Control
Volume **9**, Number **1**, January **2013**

ICIC International ©2013 ISSN 1349-4198
pp. **231–255**

AN AIS-INSPIRED ARCHITECTURE FOR ALERT CORRELATION

Mehdi Bateni[1], Ahmad Baraani[1], Ali Ghorbani[2] and Abbas Rezaei[3]

[1]Department of Computer Engineering
University of Isfahan
HezarJerib Street, Isfahan, Iran
{ bateni; ahmadb }@eng.ui.ac.ir

[2]Faculty of Computer Science
University of New Brunswick
550 Windsor Street, Fredericton, New Brunswick, Canada
ghorbani@unb.ca

[3]Department of Immunology
Isfahan University of Medical Sciences
HezarJerib Street, Isfahan, Iran
rezaei@mui.ac.ir

Received October 2011; revised February 2012

Abstract. *There are many different approaches to alert correlation such as using correlation rules and prerequisite-consequences, using machine learning and statistical methods and using similarity measures. In this paper, iCorrelator, a new AIS-inspired architecture, is presented. It uses a three-layer architecture that is inspired by three types of responses in the human immune system: the innate immune system's response, the adaptive immune system's primary response, and the adaptive immune system's secondary response. In comparison with other correlators, iCorrelator does not need information about different attacks and their possible relations in order to discover an attack scenario. It uses a very limited number of general rules that are not related to any specific attack scenario. A process of incremental learning is used to encounter new attacks. Therefore, iCorrelator is easy to set up and work dynamically without reconfiguration. As a result of using memory cells and improved alert selection policy, the computational cost of iCorrelator is also acceptable even for online correlation. iCorrelator is evaluated by using the DARPA 2000 dataset and a netForensics honeynet data. The completeness, soundness, false correlation rate and execution time are reported. Results show that iCorrelator is able to extract the attack graphs with acceptable accuracy that is comparable to the best known solutions.*

Keywords: Intrusion detection system (IDS), Alert correlation, Artificial immune system (AIS)

1. **Introduction.** Intrusion Detection System (IDS) is a rapidly growing field. It is the process of identifying and (possibly) responding to malicious activities targeted at computing and network resources [1]. When an IDS detects a malicious activity, it generates an alert. Alerts are usually in low-level format. It means that each alert contains a little information about the malicious activity that is almost useless for the administrator. On the other hand, an IDS in a large network of computers with many different users generates high volumes of low-level alerts. These raw alerts overwhelm the system administrator in such a way that she/he cannot use them effectively. As a result, the administrator may ignore these alerts and miss their possible related intrusions. Alert correlation is used to overcome this problem. It is a process that analyzes the alerts produced by one or more

intrusion detection system(s) and provides a more succinct and high-level view of occurring or attempted intrusions [2]. The most important goal of the alert correlation process is to reduce the number of alerts the system administrator should investigate manually. The administrator prefers to process a high-level attack scenario instead of a high volume of raw alerts. The correlation process is usually done by removing false alerts, aggregating related alerts and prioritizing alerts.

Most correlation systems need extra information such as correlation rules and network metadata in order to correlate the alerts. The rules define the relationship between different alerts, whereas metadata is about the features of the protected network and its assets. As a result, the correlation systems need experts with extensive knowledge of network and security in order to provide this information and keep it up to date. The everyday new emerging attack scenarios make it hard and time-consuming to provide and keep this information up to date.

In this paper, iCorrelator, a new architecture for alert correlation which employs many concepts of the Artificial Immune System (AIS), is presented. The human immune system is an efficient and accurate system which has successfully protected the humans against different types of diseases for millions of years. It gathers different signs of danger and correlates them to detect diseases. Multilayer of protection is one of the key points of its success.

iCorrelator is inspired by the human immune system. It employs three layers of correlation in order to assign a correlation probability to each pair of alerts. These correlation probabilities are used to extract the attack scenarios from the input alerts. Moreover, a group of matrices are used to store the experiences of the system from each correlation.

In the first layer of correlation, twenty one fuzzy rules are used. Each rule contains a vector and a class number that is assigned to it. The vector contains the values of six features. The values are extracted from a pair of alerts and a few matrices. The class number is used to calculate the correlation probability of two alerts. The vector and its assigned correlation probability are called a cell. To assign a correlation probability to a cell, C_i, iCorrelator explores the rules and finds the most compatible rule, R_j, with C_i. If the correlation probability in R_j is more than a predefined threshold, then it is used to assign the correlation probability to C_i. Otherwise, next layers of correlation are examined.

In the next layer of correlation a pool of immune memory cells is used. The immune memory cells have been generated by the learning-based layer of correlation (next layer) in the previous correlation processes. For each new cell, C, iCorrelator explores the pool and finds the most similar cell, C_{mx}, with C (The similarity is measured by the weighted Euclidean distance). If the similarity between C and C_{mx} is more than a predefined threshold, then the correlation probability of C_{mx} is used as the correlation probability for C. Otherwise, C is examined by the next layer of correlation.

In the next layer of correlation an incremental learning correlation method is used. The AIRS algorithm with a few changes is used in this layer. AIRS is an AIS-based algorithm which produces a population of memory cells from the training data with the ability to classify the new data. It is used to assign a correlation probability to a novel cell. A novel cell is a cell without a compatible rule in the set of general rules and without a similar cell in the pool of immune memory cells. The fuzzy rules of the first layer are used as the training data for AIRS which assigns a correlation probability to the novel cell. When AIRS assigns a correlation probability, p, for a novel cell, C, it also stores the pair of C and p in the pool of immune memory cells. This pair (C, p) is used in the memory-based layer of correlation as described above.

When a new alert, a, arrives, a few alerts from the previously observed alerts are selected, and a group of new cells are generated by using a and each selected alert. The cell, $C_{\max}$, with the most correlation probability is determined among the generated cells by using three layers of correlation. a and the other alert in $C_{\max}$ are added to a hyper-alert if their correlation probability is more than a predefined threshold (correlation threshold). Otherwise, a new hyper-alert with only one alert (a) is created. The alert selection policy defines the scope and method of the search in the previous alerts. A new enhanced time window-based approach with random alert selection is introduced for iCorrelator, named Random Directed Time Window (RDTW). A time window with a fixed number of sliding time slots is defined. When a new alert arrives, iCorrelator randomly selects m alerts from each time slot, t_i, for correlation. m is different for each time slot, and it is a function of the total number of alerts in t_i, the slot number, i, and the maximum correlation probability calculated for alerts in t_i.

In comparison with other correlation systems, iCorrelator does not need information about different attacks and their possible relations in order to discover an attack scenario. Most correlation systems use a database of rules to define the causal relations among the events. These rules describe an attack scenario or define the prerequisites and consequences of each event. Definition of these rules or scenarios needs an extensive knowledge of computer security and different types of attacks. Furthermore, this knowledge should be updated in response to everyday new emerging attacks. Therefore, the initial setup and the subsequent maintenance of the correlation systems are difficult to do and need expertise. On the other hand, iCorrelator uses a very limited number of general rules that are not related to any specific attack scenario. A process of incremental learning is used to encounter new attacks. Therefore, iCorrelator is easy to set up and work dynamically without reconfiguration. As a result of using memory cells and improved alert selection policy, the computational cost of iCorrelator is also acceptable even for online correlation.

The main contribution of the present work is a multilayer correlation architecture that is inspired by the human immune system. The architecture is based on three layers of correlation: a limited number of fuzzy rules, an AIS-based incremental learning method and a group of immune memory cells. Each layer is the equivalent of one working layer of the human immune system: innate immune system, primary response in the adaptive immune system and secondary response in the adaptive immune system. The innate immune system evolves genetically. We are born with it. It has the ability to recognize a fixed number of common pathogens. The agility, effectiveness and accuracy are the most important features of the innate immune system. The adaptive immune system has the ability to encounter new pathogens. It is able to learn the structure of a new pathogen and generate an antibody for it (primary response). The most considerable features of the primary responses are their adaptability and flexible learning. Moreover, the adaptive immune system is able to memorize the pathogens and their antibodies for future use (secondary response). The most attractive features of the secondary responses are their quickness and performance. By using a three-layer architecture for correlation, iCorrelator enjoys all these features.

As mentioned before, AIRS is used in the learning-based layer of correlation to assign a correlation probability to each cell. A new attribute weighting method is presented and used in order to improve the accuracy of AIRS. There are many methods of attribute weighting such as Pearson Partial Correlation, Information Gain and Symmetric Uncertainty, each of which is appropriate for a different application. We introduce a new attribute weighting method which is very fast and accurate, named *average contribution*. It is used to assign a weight or degree of importance to each attribute of a dataset.

The generated weights are used to improve the AIRS algorithm. Significant accuracy improvement is observed in the obtained algorithm.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 provides a brief review of the human immune system. Section 4 presents iCorrelator in details. It illustrates its architecture and provides the details of its components. Section 5 reports the result of running the system with the DARPA2000 and netForensics honeynet data. Finally, Section 6 provides the conclusion and some suggestions for further research.

2. **Related Works.** Alert correlation has two main goals: reducing the number of alerts and increasing the relevance and abstraction level of the produced reports [2]. Alert correlation techniques are usually classified in three groups: Fusion-based, Filter-based and Causality-based.

The fusion-based correlation [3, 4, 5] is based on the similarity between two alerts. It defines a function for similarity and looks for alerts that are similar. If the similarity value is more than a predefined threshold, alerts are placed in one cluster. Filter-based approaches [6, 7, 8, 9, 10, 11] either identify the false positive and the irrelevant alert or assign a priority to each alert. For instance, an alert could be classified as irrelevant if it represents an attack against a non-existent service. Priority is usually assigned to alerts based on the importance of the assets on the target. Causality-based approaches use the logical relationships among alerts to correlate them [12, 13, 14, 15, 16, 17, 18, 19, 20]. They either use the expert knowledge to find related alerts or aim to infer it from statistical or machine learning analysis. Because the approach of iCorrelator to alert correlation is closer to the causality-based approach, we focus on the works that use this approach.

There are several causality-based approaches that use known scenarios to find relationships among alerts. They match the sequence of incoming events with some predefined scenarios. These scenarios should be defined by an attack language (e.g., STATL [21], LAMDBA [22], ADeLe [23]) or learned by using machine learning techniques [12, 13]. Specifying all scenarios in advance is time-consuming and error-prone work and needs extensive knowledge of the domain. Moreover, they cannot handle new attack scenarios. Wang et al. [13] proposed a multi-step attack pattern discovering method that aims at solving problems of new attack pattern discovery and also overcoming the difficulty in complex attack association rule definition and maintenance. They mine multi-step attack activity patterns with the attack sequential pattern mining method from history aggregated high-level alerts. Their method requires good integration of history database which should include various multi-step attack instances.

Another type of causality-based correlation systems uses the rule-based correlation approach [14, 15, 16, 17]. They rely on the fact that complex attacks are usually executed in several phases or steps, where the first step prepares for attacks executed in the later steps. Each step of the attack has its prerequisites and consequences. Thus, analyzing alerts based on the predefined rules containing prerequisites and consequences of the attack steps is sufficient to identify related alerts.

Both scenario-based and rule-based approaches rely on expert knowledge to find related alerts and cannot handle novel attacks. Statistical approaches [18, 19, 20, 24] analyze relationships among alerts based on their co-occurrence within a certain time period, and thus, are generally independent of the prior domain knowledge.

Qin [18] presented a Bayesian correlation engine for discovering the statistical relation among alerts. He analyzes statistical patterns among aggregated alerts, with the assumption that alerts are causally related if a strong statistical correlation exists among them. The degree of relevance of alerts is evaluated by calculating the conditional probability

between each pair of hyper-alerts. The approach builds an attack scenario by evaluating the causal relationship between each pair of hyper-alerts. Because of the large number of possible combinations between hyper-alerts, the running of the system in online mode is infeasible.

Ren et al. [19] presented an approach for adaptive online alert correlation. The approach incorporates two components: the offline module that is responsible for retrieving relevant attack information from the previously observed alerts based on the Bayesian causality mechanism; and the online component that is based on the extracted information. It correlates raw alerts and constructs attack scenarios online.

There are other works that use machine learning algorithms to estimate the correlation probabilities among alerts and use them in correlation. Zhu et al. [24] used Multilayer Perceptron and Support Vector Machines to estimate the alert correlation probability, and Sadoddin et al. [20] used the frequent structure mining technique. All statistical and machine learning-based approaches do not require expert knowledge and are capable of representing unknown attacks. The most important drawback of these methods is their computational cost. As a result, these methods are not usually appropriate for online correlation.

3. **Human Immune System.**

The Human immune system is responsible for protecting the body against invasion by diseases and other pathogens including viruses, bacteria and parasites. Its ability to detect and eliminate most pathogens is essential for survival; without an immune system we die within weeks [26]. The architecture of the immune system is multi-layered, with defenses on several levels.

3.1. **Innate immune system.**

The innate immune system provides immediate defense against infection. It evolves genetically. We are born with it. It does not change or adapt during our life. The innate immune system identifies and removes the foreign substances present in organs, tissues, the blood and lymph, by specialized white blood cells. It initiates and directs the response of the adaptive immune system. It comprises the cells and mechanisms that defend the body from infection by pathogens in a non-specific manner. This means that the cells of the innate immune system recognize and respond to pathogens in a generic way, but unlike the adaptive immune system, it does not confer long-lasting or protective immunity.

3.2. **Adaptive immune system.**

The adaptive (acquired) immune system provides the vertebrate immune system with the ability to recognize and remember a specific pathogen, and to mount stronger attacks each time the pathogen is encountered. It comprises two types of immune cells (lymphocytes): the B-Cells and T-Cells. These cells are able to adapt to new pathogens and learn their structure. Each individual cell learns the structure of a certain pathogen and adapts with it. The adaptation occurs when chemical bonds are established between receptors on the surface of an immune cell and epitopes which are located on the surface of a pathogen. The strength of the bond between a receptor and an epitope is termed the affinity. If the affinity between a cell and a pathogen is strong enough, then the cell produces copies of itself (clones that are produced through cell division). The number of clones for the cell is dependent to the affinity strength. The higher the affinity of a cell for the pathogens present, the more likely it is that the cell will clone. Within the cloning process, mutation is occurred, and if the affinity of the mutated clone would be higher than the original cell, then it will be cloned more. During this process a specific cell for the encountered pathogen is evolved by the immune system (primary response). This specific cell is transformed to the memory cell and remains in the immune system for future use (secondary response). A consequence of learning and

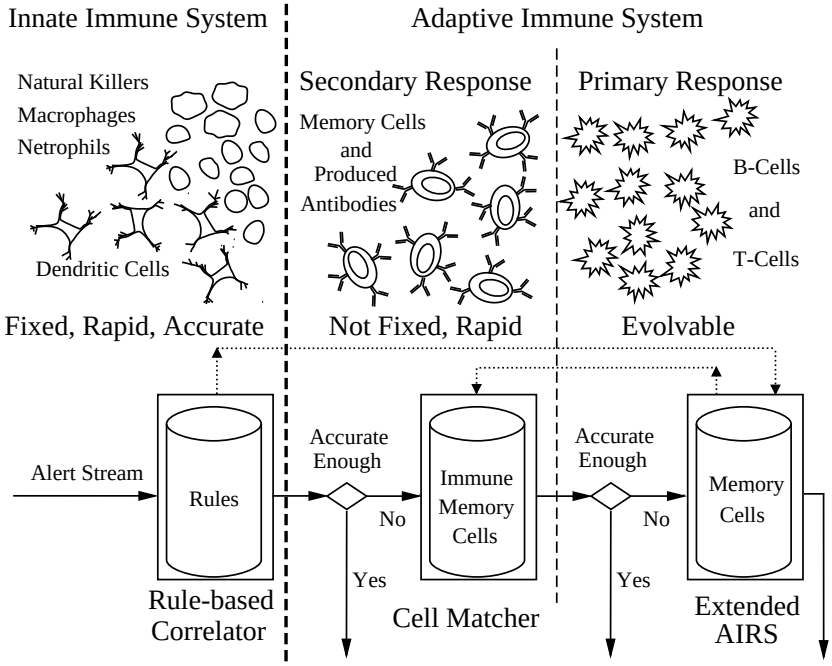

FIGURE 1. Comparing iCorrelator and the human immune system

memory is that two types of immune response can be distinguished: a primary response to previously unencountered pathogens, and a secondary response to pathogens that have been encountered before [26].

3.3. **Artificial immune system.** The human immune system is an adaptive, robust, decentralized and error tolerant system. These properties are desirable for the development of novel computer systems. Artificial Immune Systems (AIS) are algorithms and systems that use the human immune system as inspiration [25]. It is used in anomaly detection, pattern recognition, fault detection, data mining and computer security. Most applications of AIS in computer security are related to its ability to discriminate between the self and non-self. There are so many researches in this area. Hofmeyr [26] proposed an immunological model of distributed detection and used it in intrusion detection application. Kim [27] investigated some AIS algorithms and used them to design an intrusion detection system. Schaust et al. [28] and Fu et al. [29] used AIS in misbehavior detection and anomaly detection. Lin et al. [30] proposed a functional-link-based neural fuzzy network with immune particle swarm optimization for solving prediction and control problems.

Our research in AIS area is focused on alert correlation as a new application for AIS in computer security. The present work is an extension of our previous works [31, 32] on alert correlation which employed AIS and fuzzy logic for alert correlation. In present work a new multilayer architecture, called iCorrelator is introduced and evaluated.

4. **iCorrelator.** iCorrelator is an immune-inspired architecture for alert correlation. Its goal is to extract attack scenario from raw alerts. In order to correlate the input alerts, it creates a cell from each pair of alerts and assigns a correlation probability to the cell. Three layers of correlation are used to assign a correlation probability to a cell: rule-based correlation layer, learning-based correlation layer and memory-based correlation layer.

The correlation probability assignment process for each cell is a serial process (Figure 1). Firstly, a limited number of general rules which are defined in set up time are used. These rules are some typical cases and are used by iCorrelator in order to enjoy the

simplest correlation relationships. The antecedent part of a rule contains six features and their corresponding values, and its consequent part contains a class number. These six features are calculated by considering a pair of alerts and their similarity (see Section 4.2). iCorrelator searches the rules to find the most compatible rule with the current pair of alerts and uses the class number in the consequent part of the rule to calculate the correlation probability of the pair. There are a limited number of fixed and general rules in this layer. Thus, the response in this layer is rapid and accurate (if it finds a compatible rule). This mechanism is comparable to the fixed, rapid and accurate response of the innate immune system.

Secondly, if iCorrelator does not find a compatible rule, then the next layer of correlation (memory-based) is used. In this layer, a pool of previously generated immune memory cells is explored for a cell similar to the current cell. The cells which are in the pool have been generated by learning-based layer during the previous correlation processes. In fact, each immune memory cell contains one previous correlation experience. The content of the pool of immune memory cells is updated dynamically. Each new correlation generates a new cell and it will be added to the pool. Using these immune memory cells by iCorrelator is comparable to the secondary response in the adaptive immune system. The correlation process in this layer is dynamic and rapid.

Finally, if iCorrelator does not find a matching cell in the pool of immune memory cell, then AIRS is used to calculate the correlation probability. AIRS is an AIS-based algorithm. It is a supervised learning algorithm and uses the same aforementioned general rules as input and generates some memory cells for them. These memory cells are used to classify new cells. The process of generating memory cells in AIRS is an evolutionary process that is inspired by the primary response in the adaptive immune system.

The calculated probability and the generated cell in the learning-based layer are stored in the pool of immune memory cells for future usages. It is important to differentiate between the immune memory cells which are placed in the pool of immune memory cells and the memory cells which are used in the AIRS algorithm. The memory cells in AIRS

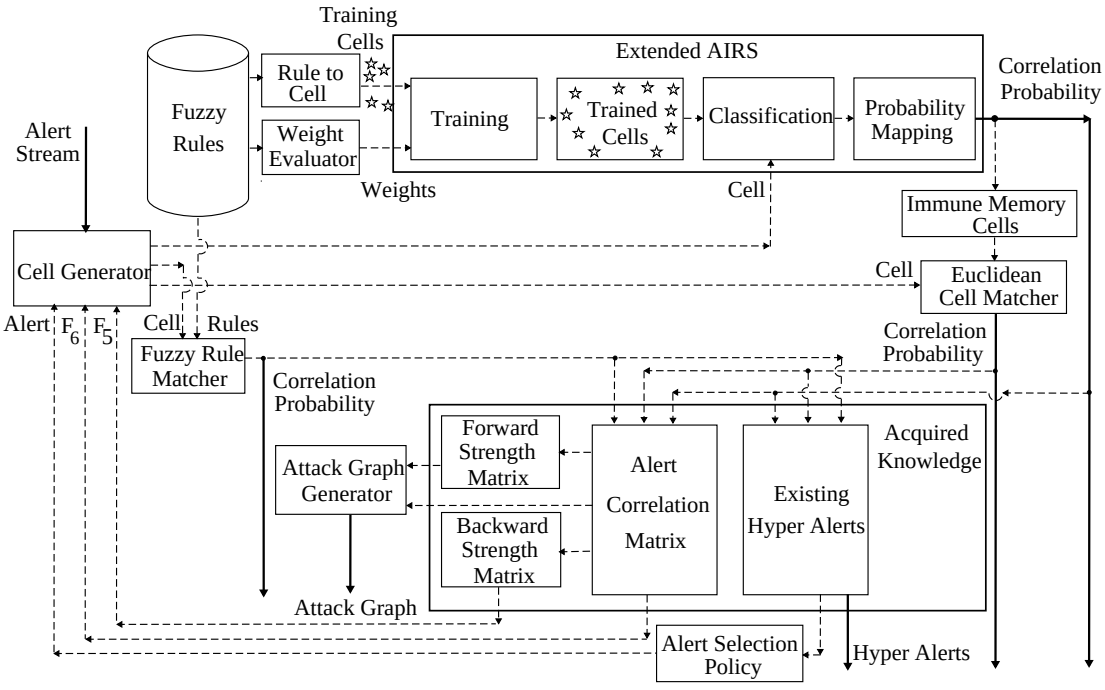

FIGURE 2. Architecture and components of iCorrelator

are generated in an evolutionary process by considering the rules which are in the rule-based layer. On the other hand, immune memory cells are the memory of the system from its correlation experience in learning-based layer.

Figure 2 shows the architecture and components of iCorrelator. Three different correlation probability assignment modules are observable. The first one is the output of Fuzzy rule matcher, the second one is the output of Extended AIRS and the third one is the output of Euclidean cell matcher. There are also a cell generator unit, an attack graph generator, an alert selection unit and a collection of acquired knowledge in the form of matrices and hyper-alerts. Different parts of this architecture are described in details in the next seven subsections.

4.1. **Knowledge acquiring.** In this section, three matrices which are used in the correlation process are introduced. They are Alert Correlation Matrix (ACM), forward correlation strength matrix (Π^f) and backward correlation strength matrix (Π^b). Each element of ACM is the correlation weight between a pair of alert types. The correlation weight for two alert types is the total correlation probabilities between them during the correlation process. It is calculated as [24]:

$$W_{c(a_i,a_j)} = \sum_{k=1}^{n} P_{i,j}(k) \tag{1}$$

where a_i and a_j are two alerts from type i and type j, and $P_{i,j}(k)$ is the correlation probability for a_i and a_j in their k^{th} correlation. $P_{i,j}(k)$ is produced by the correlation engine. ACM is not symmetric. It encodes the temporal relationship between two alerts (a_i is occurred before a_j). The ACM elements are used later for generating two strength matrices (Π^b, Π^f). By each new correlation between a pair of alert types, ACM is updated dynamically. As a result, the correlation probability of the corresponding alert types will be updated accordingly. The elements of two strength matrices are calculated as [24].

$$\Pi^f_{c(a_i,a_j)} = \frac{W_{c(a_i,a_j)}}{\sum_{k=1}^{n} W_{c(a_i,a_k)}} \tag{2}$$

$$\Pi^b_{c(a_i,a_j)} = \frac{W_{c(a_i,a_j)}}{\sum_{k=1}^{n} W_{c(a_k,a_j)}} \tag{3}$$

$\Pi^f(a_i, a_j)$ is used to predict the correlation probability of one alert (a_i) with another alert that happens after it (a_j). It is used in the attack graph generation. On the other hand, $\Pi^b(a_i, a_j)$ is used to find the correlation probability of one alert (a_j) with another alert that has happened before it (a_i). $\Pi^b(a_i, a_j)$ is used as the fifth feature in the process of feature vector generation (cell generation). Both matrices initially are filled with zero. After each correlation the corresponding elements in ACM, Π^b and Π^f are updated. These matrices play the role of some sort of memory or acquired knowledge for the correlation system [31, 32].

4.2. **Cell generation.** iCorrelator is an AIS-based correlator. It uses concepts such as cell, antigen and memory cell. It works on input antigens and produces the memory cells. The memory cells are used to classify the new input cells. In iCorrelator, each cell (antigen) is a vector of six features and one class number. The cell generator creates a cell by using a pair of alerts. Suppose that a_1 and a_2 are two alerts, and a_1 was produced before the a_2. Six features of two alerts are selected to produce a feature vector (cell) [24].

- F_1: Source IP similarity for a_1 and a_2 ([0-1])
- F_2: Destination IP similarity for a_1 and a_2 ([0-1])
- F_3: Equality of destination port for a_1 and a_2 (0 or 1)

- F_4: Equality of destination IP of a_1 with the source IP of a_2 (0 or 1)
- F_5: Backward strength correlation for alerts of type a_1 and a_2 ([0-1])
- F_6: Correlation frequency for alerts of type a_1 and a_2 ([0-1])

Suppose that the timestamp, source address, destination address and alert type for two alerts a_1 and a_2 are 4:13:20, 172.16.114.50:1227, 172.16.113.50:25, Email_Ehlo, and 5:06:16, 172.16.113.50:1048, 172.16.112.50:21, FTP_User respectively. Their corresponding cell is (F_1=.6872, F_2=.7187, F_3=0, F_4=1, F_5=.2424, F_6=1) [31].

4.3. **Fuzzy rules and fuzzy rule matcher.** A limited number of fuzzy rules are defined in order to assign a correlation probability to each pair of alerts. The antecedent part of a rule contains six features ($F_1 - F_6$) and their corresponding values ($V_1 - V_6$). Two features (F_3 and F_4) are Boolean. Five others are expressed by linguistic terms such as *high*, *low* and *medium*. These linguistic terms are defined by appropriate fuzzy sets. The consequent part of a rule is a class number. It is an integer number between 1 and λ. The class number of 1 is equal to the probability value of 0, and the class number of λ is equal to the probability value of 1. Other class numbers (between 1 and λ) are mapped to the probability values by using a mapping function. Assigning a class number to the consequent part of a rule requires certain degree of knowledge about the similarity measurement, but it is not directly related to any particular alert types. For example, a rule expresses that if the similarity of source and target IP addresses in two alerts are high, and the target ports for both alerts are the same, and the target IP of the first alert is not equal with the source IP of the second alert, and the frequency of previous correlation for these two alert types are high, and the backward correlation strength of these two alert types are high, then the class number is λ (correlation probability is 1). A few more sample rules are shown in Table 1.

TABLE 1. Sample predefined rules with $\lambda = 20$

	F_1	F_2	F_3	F_4	F_5	F_6	$Class$
Rule 1	Med	Med	1	0	High	High	16
Rule 2	High	High	1	0	Low	Low	19
Rule 3	High	High	1	0	Low	Low	18
Rule 4	Med	Med	0	0	Med	Low	4
Rule 5	Med	Med	0	0	Med	Low	3

The rule format is as follows:

If $(F_1 = V_1)$ *and . . .* $(F_6 = V_6)$ *Then* $(Class = C)$

A fuzzy rule matcher is used to assign a probability value to an input feature vector x. It uses the concept of compatibility. The compatibility of a feature vector $x = (v_1, v_2, v_3, v_4, v_5, v_6)$ with a rule R_j is the average of its six features membership values with respect to the rule R_j. It is calculated as follows [32]:

$$Compatibility(x, R_j) = \frac{1}{n} \sum_{i=1}^{n} \mu(v_i, V_i) \tag{4}$$

where n is the number of features, v_i is the value of i^{th} feature in x, V_i is the value of i^{th} feature in the antecedent part of the rule R_j, and μ is the membership function for the fuzzy set V_i. The compatibility of x with all rules is calculated in order to find the most compatible rule. If the compatibility value for the most compatible rule is more than a predefined rule selection threshold, rs, then the class number for x is determined by the fuzzy rule matcher. The class numbers in the consequent parts of the three most

compatible rules with x, their compatibility values and their distances from each other are considered in order to assign a class number, C, to input vector, x (see Algorithm 1). After assigning the class number C to input vector x, a mapping function is required to convert C to a probability value. Equation (5) is used for this purpose.

$$P = \frac{\mathrm{C} - 1}{\lambda} + \frac{1}{2\lambda} \tag{5}$$

Algorithm 1 outlines the class number identification and probability mapping function [32].

4.4. **Extended AIRS.** An extended version of the AIRS algorithm is used in the learning-based layer of correlation. It uses the AIRS algorithm with a few modifications. Weighted Euclidean is used for distance calculation, and least average distance is used for class selection. A mapping function is also provided for mapping the class number to probability.

4.4.1. *The AIRS algorithm.* AIRS is a supervised-learning algorithm. It was introduced in 2001 for the first time by Watkins [33]. Its revised version was introduced later [34]. It is more efficient than the original version and has the same level of accuracy. In this

Algorithm 1 Probability calculation for rule x in fuzzy classifier

R_1, R_2, $R_3 \leftarrow$ The three most compatible rules with x
$\lambda \leftarrow$ The number of classes
if $(R_1.cmpt - R_2.cmpt > .15)$ or $(R_1.cmpt - R_3.cmpt > .25)$ **then**
  return $(R_1.class - 1)/\lambda + 1/(2{*}\lambda)$
end if
Sort R_1, R_2, R_3 to R_a, R_b, R_c based on $R_i.class$
if $(R_b.class - R_a.class \geq 3)$ and $(R_c.class - R_b.class \geq 3)$ **then**
  return $(R_1.class - 1)/\lambda + 1/(2{*}\lambda)$
end if
if $R_b.class - R_a.class \geq 3$ **then**
  $d \leftarrow \frac{\min(R_c.cmpt, R_b.cmpt)}{(R_c.cmpt + R_b.cmpt)} * (R_c.class - R_b.class)$
  if $R_b.cmpt > R_c.cmpt$ **then**
    $C \leftarrow R_b.class + d$
  else
    $C \leftarrow R_c.class - d$
  end if
  return $(C - 1)/\lambda + 1/(2{*}\lambda)$
end if
if $R_c.class - R_b.class \geq 3$ **then**
  $d \leftarrow \frac{\min(R_b.cmpt, R_a.cmpt)}{(R_b.cmpt + R_a.cmpt)} * (R_b.class - R_a.class)$
  if $R_a.cmpt > R_b.cmpt$ **then**
    $C \leftarrow R_a.class + d$
  else
    $C \leftarrow R_b.class - d$
  end if
  return $(C - 1)/\lambda + 1/(2{*}\lambda)$
end if
if $(R_b.class - R_a.class = 2)$ and $(R_c.class - R_b.class = 2)$ **then**
  return $(R_1.class - 1)/\lambda + 1/(2{*}\lambda)$
end if
return $(R_b.class - 1)/\lambda + 1/(2{*}\lambda)$

paper, we refer to this new version as AIRS. The main goal of the algorithm is to produce a population of memory cells from the training data with the ability to classify the new data. The AIRS design refers to many immune system metaphors including resource competition, clonal selection, affinity maturation and memory cell retention. It uses the resource limited artificial immune system concept. In this algorithm, the feature vectors that are presented for training and test are called antigens while the system units are called B-cells. A group of similar B-cells are called Artificial Recognition Ball (ARB). The ARBs compete with each other for a fixed number of resources, and the ARBs with higher affinities to the training antigen are improved. Each antigen in training data is presented to algorithm once, and the algorithm creates a memory cell for it. The memory cells that are formed after the presentation of all training antigens are used to classify test antigens. The class of a test antigen, x, is determined by majority voting among the k most stimulated memory cells with x, where k is a user defined parameter.

The aforementioned fuzzy rules are converted to appropriate form for using by AIRS. They are called training cells or antigens in the AIRS algorithm. Table 2 shows a population of cells that are generated based on the rules of Table 1. These cells are used in the training phase of the AIRS algorithm. The algorithm processes the training cells and generates a population of memory cells. The memory cells are used in the classification phase of the algorithm. The goal of the classification phase is to assign a class number (probability value) to each new input cell. Table 3 shows a sample population of generated memory cells.

As mentioned before, some modifications are made in AIRS in order to be used in alert correlation. These modifications are made to improve the accuracy of the algorithm for alert correlation and to convert the output of the algorithm to a real value. The modifications are described in next three subsections.

4.4.2. *Weight assignment.* One of the key elements of the AIRS algorithm is its distance calculation method. The distance between two cells is used to calculate their stimulation level. It is calculated as follows:

$$Stimulation(c_1, c_2) = 1 - Distance(c_1, c_2)$$

where $Distance(c_1, c_2)$ is the Euclidean distance between two cells, c_1 and c_2.

TABLE 2. The training cells corresponding to rules of Table 1

	F_1	F_2	F_3	F_4	F_5	F_6	$Class$
Cell 1	0.5	0.5	1	0	1.0	1.0	16
Cell 2	1.0	1.0	1	0	0.0	0.0	19
Cell 3	1.0	1.0	1	0	0.0	0.0	18
Cell 4	0.5	0.5	0	0	0.5	0.0	4
Cell 5	0.5	0.5	0	0	0.5	0.0	3

TABLE 3. Sample generated memory cells by AIRS

	F_1	F_2	F_3	F_4	F_5	F_6	$Class$
Cell 1	0.55	0.99	1	0	0.38	0.62	16
Cell 2	0.98	1.0	1	0	0.10	0.61	19
Cell 3	1.0	1.0	1	0	0.50	0.07	18
Cell 4	0.20	0.46	0	0	0.24	0.20	4
Cell 5	0.65	0.50	0	0	0.10	0.11	3

Since, the number of classes is high ($\lambda = 20$), and the number of training data is limited, a more accurate method for computing the distance values is required. By observing the training data, it is obvious that some features are more important than others. A slightly change in these features causes a considerable change in the class number (probability value). It is better to assign a weight to each feature in order to reflect its importance in distance calculation [31].

There are many methods of attribute weighting such as Pearson Partial Correlation, Information Gain and Symmetric Uncertainty, each of which is appropriate for a different application. These methods are used to assign a weight or degree of importance to each attribute in a dataset of vectors. We introduce a new attribute weighting method which is very fast and accurate, named *Average Contribution*.

Consider a feature vector v that contains m numerical predictor features, F_1-F_m, and a numerical predicted value P. Assume that all m features are normalized between 0 and 1. The *Contribution* of the feature F_j in providing the value of P for vector v is calculated as below:

$$C(F_j, P) = \frac{F_j \times P}{\sum_{k=1}^{m} F_k} \tag{6}$$

Average contribution of a feature in a dataset of vectors is defined based on its *contribution*. It shows the total *contribution* of F_j in providing different values of P. It is used as the weight or importance degree of the F_j in a dataset of vectors.

Consider dataset S in Figure 3 in which there are n feature vectors (v_1-v_n). Each vector contains m numerical predictor values (F_1-F_m) and a numerical predicted value (P). *Average contribution* of the feature F_j in dataset S is calculated as follows:

$$W_j = AC(F_j) = \frac{\sum_{k=1}^{n} C(f_{kj}, p_k)}{\sum_{k=1}^{n} f_{kj}} \tag{7}$$

where f_{kj} is the value of the j^{th} feature in the k^{th} vector, p_k is the numerical value that is assigned to the k^{th} vector (predicted value) and $C(f_{kj}, p_k)$ is *Contribution* of the j^{th} feature in the k^{th} vector.

As mentioned before, the rules that are used in the first layer of correlation are used as the training data for the AIRS algorithm (Table 2). The *Average Contribution* is used to assign a weight to each attribute in this data. As a result, the following coefficients are produced.

$W_1 = 0.154$, $W_2 = 0.155$, $W_3 = 0.148$, $W_4 = 0.249$, $W_5 = 0.152$, $W_6 = 0.141$.

	F_1	F_2	...	F_j	...	F_m	P
v_1	$f_{1,1}$	$f_{1,2}$	...	$f_{1,j}$	...	$f_{1,m}$	p_1
v_2	$f_{2,1}$	$f_{2,2}$	...	$f_{2,j}$	...	$f_{2,m}$	p_2
⋮	⋮	⋮		⋮		⋮	⋮
v_i	$f_{i,1}$	$f_{i,2}$	...	$f_{i,j}$	...	$f_{i,m}$	p_i
⋮	⋮	⋮		⋮		⋮	⋮
v_n	$f_{n,1}$	$f_{n,2}$	...	$f_{n,j}$	...	$f_{n,m}$	p_n

FIGURE 3. Feature vectors and weight calculation

They are used in weighted Euclidean distance calculation for two cells, c_1 and c_2, as follows:

$$Distance(c_1, c_2) = \sqrt[2]{\sum_{j=1}^{m} W_j \times (c_1.F_j - c_2.F_j)^2} \tag{8}$$

4.4.3. *Class selection policy.* The KNN algorithm is used in the last step of AIRS. It is used to assign a class number to data item x based on the class numbers of its K nearest memory cells. AIRS uses the majority vote in the KNN algorithm. It means that the class label of data item x is the class label with the most number of memory cells among the K nearest memory cells. Our experimental results show that the replacement of the majority vote with the least average distances improves the accuracy of the AIRS algorithm in correlation system. Therefore, the class label for the data item x is the class label with the least average distances from x (among the K nearest memory cells). It is possible that class, C_a, with two memory cells identifies the class label of x instead of class, C_b, with three memory cells. Because the average distance from x for memory cells with class C_a is less than the average distance from x for memory cells with class C_b.

4.4.4. *Probability mapping.* Eventually, a class number, C_i, is assigned to an input data item (a cell) by AIRS. In order to use C_i in alert correlation, AIRS should map it to a probability value. Equation (5) is used again. To improve the accuracy of the mapping the predecessor and successor class of C_i (C_{i-1}, C_{i+1}) are also considered in mapping process. Suppose that the class label generated for an antigen Ag is C_i, and the average distance of Ag with C_i is d_i, and the distances of Ag with C_{i-1} and C_{i+1} are d_{i-1} and d_{i+1} respectively. Equation (5) is modified as follows:

$$P = \frac{C_i - 1}{\lambda} + \frac{1}{2\lambda} + \Delta \tag{9}$$

$$\Delta = \begin{cases} \dfrac{-d_i}{\lambda(d_i + d_{i-1})} & \text{if } (d_{i-1} < d_{i+1}) \text{ or } (\nexists C_{i+1}) \\ 0 & \text{if } (d_i = 0) \text{ or } (d_{i-1} = d_{i+1}) \text{ or } (\nexists C_{i-1} \text{ and } \nexists C_{i+1}) \\ \dfrac{d_i}{\lambda(d_i + d_{i+1})} & \text{if } (d_{i+1} < d_{i-1}) \text{ or } (\nexists C_{i-1}) \end{cases}$$

4.5. **Immune memory cells and Euclidean cell matcher.** In natural immune system the cell that recognizes a pathogen is transformed to the memory cell and is stored in the pool of immune memory cells which provide the secondary response of the immune system. The secondary response is more rapid than the primary response.

Inspiring by the secondary response of the immune system, iCorrelator stores the cell and its correlation probability for future usages. Each time a correlation probability, p, is assigned to cell, x, by AIRS, the pair of x and p is stored in the pool of immune memory cells. As a result, if iCorrelator encounters a cell similar to x, then the value of p will be assigned to it immediately. In this way, the performance of the correlation probability assignment is increased considerably.

Weighted Euclidean distance is used to calculate the similarity between a new cell and an immune memory cell. A cell matching threshold (cm) that is between 0 and 1, is considered to define the minimum required similarity. A large number (more than 0.95) is assigned to cm, in order to match the cells more accurately.

The immune memory cells are different from the memory cells that are generated in the training phase of AIRS. The memory cells generated by AIRS are a limited number of static memory cells, and they are generated from the predefined rules. However, the

Algorithm 2 Probability calculation for cell x in AIRS

$n \leftarrow$ The number of memory cells in MC
$\lambda \leftarrow$ The number of classes
for $j = 1$ **to** n **do**
  $d_j \leftarrow$ Weighted_Euclidean (x, MC_j)
end for
$KNN \leftarrow K$ Memory cells with least distances to x
$i \leftarrow$ The index of memory cell with least distance to x
$C_i \leftarrow MC_i.class$ // class number of MC_i
if $d_i = 0$ **then**
  $\Delta \leftarrow 0$
end if
if $(\exists j, MC_j \in KNN)$ and $(MC_j.class = MC_i.class - 1)$ **then**
  $d_{i-1} \leftarrow$ Weighted_Euclidean (x, MC_j)
end if
if $(\exists k, MC_k \in KNN)$ and $(MC_k.class = MC_i.class + 1)$ **then**
  $d_{i+1} \leftarrow$ Weighted_Euclidean (x, MC_k)
end if
if $((\nexists C_{i-1})$ and $(\nexists C_{i+1}))$ or $(d_{i-1} = d_{i+1})$ **then**
  $\Delta \leftarrow 0$
end if
if $(\nexists C_{i-1})$ or $(d_{i-1} > d_{i+1})$ **then**
  $\Delta \leftarrow \frac{d_i}{(d_i+d_{i+1})*\lambda}$
end if
if $(\nexists C_{i+1})$ or $(d_{i-1} < d_{i+1})$ **then**
  $\Delta \leftarrow \frac{-d_i}{(d_i+d_{i-1})*\lambda}$
end if
return $(C_i\text{-}1)/\lambda + 1/(2*\lambda) + \Delta$

immune memory cells are dynamically generated cells that are generated and stored during the correlation process.

Immune memory cells are organized in a 4-way search tree in order to improve the performance of the search among them. Each time AIRS produces a new cell and assigns a correlation probability to it, AIRS inserts the cell and the probability in this tree. The structure of the tree is based on the value of the six features of the cells. During a search, a path in the tree from the root to a leaf node is examined. The path is determined by the values of the features in the input cell. Each leaf node contains a group of previously generated similar cells and their correlation probability values. Figure 4 shows the search paths for two cells c_1:(0.21, 0.75, 0, 1, 0.55, 0.88) and c_2:(1, 0.15, 0, 1, 0.13, 0.5).

If iCorrelator finds a cell in the designated leaf node that its similarity with the input cell, x, is more than the cell matching threshold, cm, then the correlation probability in that cell is considered as the correlation probability of x. Otherwise, iCorrelator uses the AIRS algorithm to calculate the correlation probability of x. After calculating the correlation probability by AIRS, iCorrelator inserts x and the calculated probability in the designated leaf node.

4.6. **Alert selection policy and correlation.** After obtaining the proper accuracy in the probability calculation process, it is possible to start the correlation process. Each new input alert, a, is probably related to few previously observed alerts that are inserted into a structure called hyper-alert. A hyper-alert contains the previously observed alerts with a degree of correlation that could be placed in a candidate attack scenario. An

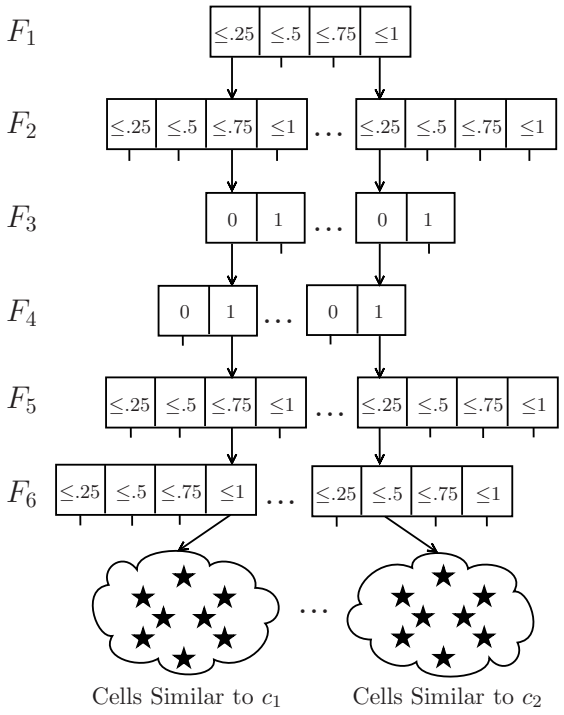

FIGURE 4. Finding two cells c_1 and c_2 in immune memory cells

alert selection policy which defines the scope and method of the search in the previously observed alerts, is required to search through the existing hyper-alerts.

New alert a first goes through the cell generator which requires two alerts to generate a cell. One of them is a. Another one is provided by the alert selection policy module by searching through the existing hyper-alerts (see Figure 2). A simple selection policy is to select all previous alerts for examining their correlation with a. It is a good policy for small dataset, but it is not scalable for large dataset. Moreover, it is not necessary to check a with all previous alerts. It is possible to examine only the most recent alerts during a time window.

A new enhanced time window-based approach with random alert selection, named Random Directed Time Window (RDTW) is introduced for iCorrelator. A time window with a fixed number of sliding time slots is defined. Only the previous alerts in the time window are selected for examining their correlation, and all old alerts, out of the time window are ignored.

Suppose that the number of time slots in a time window is n, and the time slots are numbered from 1 to n. Also, suppose that there are s_i alerts in slot number i. When a new alert arrives, iCorrelator randomly selects m alerts from each time slot, t_i, for correlation. m is different for each time slot and is less than or equal to s_i. The value of m is determined by considering three values: i, the slot number that is between 1 and n; s_i, the total number of alerts in t_i and mx, the maximum correlation between new alert, a, and all selected alerts from t_i. Two first values (s_i and i) are used to calculate the initial value of m as below [35]:

$$m = \left\lfloor \frac{s_i \times i}{n} \right\rfloor \tag{10}$$

For example, in Figure 5, the numbers of selected alerts for slot 1 to 5 are 2, 6, 7, 10 and 11 respectively. In order to select alerts more effectively, iCorrelator uses RDTW to direct the selection toward the time slots with more relevant alerts. Suppose that mx is

the maximum correlation value between the new alert, a, and all selected alerts in t_i at this moment. Also suppose that *min_accept* is the minimum acceptable probability value for continuing the alert selection from a slot. For alert a, the alert selection from slot t_i is started by calculating the initial value of m (using Equation (10)). After selecting and correlating $m/2$ alerts of t_i, the calculated value of mx is checked. If mx is less than *min_accept* threshold, then it seems that a is not related with t_i. As a result, m is decremented by one, and the selection and correlation process is continued by the new value of m. On the other hand, if after correlating m alerts the mx value is more than *1-min_accept*, then it is reasonable to conclude that a is strongly related with t_i. Thus, m is incremented by one and the system continues the process of alert selection and correlation from t_i. The selection terminates either by encountering an alert with correlation probability less than *1-min_accept* or by selecting all alerts in t_i. As a result of this strategy, iCorrelator directs the selection toward the more relevant time slots. There are three adjustable parameters that influence the performance of RDTW policy: n, the number of slots in a time window; W_s, the width of each time slot and *min_accept*, the minimum acceptable correlation probability for a slot.

By arriving each new alert, a, iCorrelator selects some previously observed alerts as described above and calculates the correlation probability between a and each selected alert. Suppose alert, x that belongs to hyper-alert $H_{\max}$, has the most correlation probability, $C_{\max}$, with a among the alerts of all time slots. If $C_{\max}$ is more than a predefined threshold (*Correlation Threshold*), then a is added to $H_{\max}$, and the correlation of a with all alerts in $H_{\max}$ is calculated. Otherwise, a new hyper-alert is created, and a is added to it.

Each time an alert is selected by the alert selection policy and is correlated by one of the two layers of correlation (fuzzy and AIRS-based), the ACM, Π^b and Π^f matrices are also updated. As a result, the system updates its acquired knowledge dynamically and adapts itself incrementally with the new correlation results. Algorithm 3 outlines the random directed time window-based alert selection and correlation.

4.7. **Attack graph generation.** Hyper-alert is useful for presenting the relationships among the alerts. However, by considering the number of generated alerts, it is obvious that the size of the hyper-alerts is increased very quickly, and it becomes very difficult to extract the required information. Each hyper-alert contains the step by step progress of an attack. An attack graph is a directed graph that shows the overall scenario of an attack, and it contains one node for each alert type. An attack graph presents an overall and concise view of an attack scenario. As mentioned before, Π^f and ACM are generated during the correlation process, and are used in the attack graph generation.

The attack graph generation starts with an alert that represents a particular type of attack. Then it performs a horizontal search in the ACM to find alerts that are most likely to happen after this alert. These alerts become new starting points to search for alerts that are more likely to happen next. The process is repeated until no other alerts are found to follow any existing alerts in the graph [24].

5. **Evaluation and Results.** The input of iCorrelator is the information such as source IP address, destination IP address and destination port number. Hence, it is working on the network layer data. Moreover, iCorrelator is not able to process the alerts generated by different IDSs directly. The format of alerts for each IDS is possibly different from others. Therefore, alerts should be converted to the internal format of iCorrelator before they are processed. The alerts produced by Realsecure and Snort on DARPA2000 and netForensics honeynet data are employed to evaluate iCorrelator. DARPA2000 is a well known dataset

Algorithm 3 Alert selection and correlation for input alert a

```
1: C_max ← −1
2: n ← The number of time slots
3: for i = 1 to n do
4:    m ← ⌊s_i * i/n⌋ // s_i is the number of alerts in slot i
5:    k ← 0
6:    mx ← −1
7:    while (k < m) and (k < s_i) do
8:       b ← a random alert from slot i
9:       c ← GenerateCell(a, b)
10:      y ← Correlation(c)
11:      if y > mx then
12:         mx ← y
13:      end if
14:      if y > C_max then
15:         C_max ← y
16:         x ← b
17:      end if
18:      k ← k + 1
19:      if (k = m) and (mx > 1-min_accept) and (k < s_i) then
20:         m ← m + 1
21:         mx ← 0
22:      end if
23:      if (k > m/2) and (mx < min_accept) then
24:         m ← m − 1
25:      end if
26:      if y ≥ Correlation Threshold then
27:         Update ACM, Π^b and Π^f
28:      end if
29:   end while
30: end for
31: if C_max < Correlation Threshold then
32:    Create a new hyper-alert and add a to it
33: else
34:    H_max ← hyper-alert contains x
35:    for j = 1 to length(H_max) do
36:       b ← alert j^th of H_max
37:       c ← GenerateCell(a, b)
38:       y ← Correlation(c)
39:       if C_max − y ≤ Correlation Sensitivity then
40:          Update ACM, Π^b and Π^f
41:       end if
42:    end for
43:    Add a to the hyper-alert H_max
44: end if
```

for evaluating the alert correlation systems. There are many works that have used this dataset. Hence, it is possible to compare the generated results with the other works. netForensics honeynet data is used to evaluate the accuracy and performance of the iCorrelator. It contains about 70000 alerts, and it is more appropriate for performance evaluation than DARPA2000.

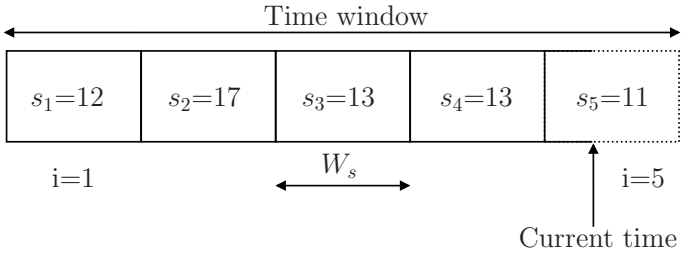

FIGURE 5. A time window containing 5 time slots ($n = 5$)

Realsecure produces 922 alerts from 22 different types for the Inside1 traffic of the DARPA2000 data. It contains the LLDoS1.0 attack scenario. LLDoS1.0 has the following stages:

1. IPsweep of the network from a remote site
2. Probe of live IPs to look for the sadmind daemon
3. Break-ins via the sadmind vulnerability
4. Installation of the trojan mstream DDoS
5. Launching the DDoS

Realsecure also produces 494 alerts from 20 different types for the Inside2 traffic of DARPA2000. It contains the LLDoS2.0 attack scenario. LLDoS2.0 has the following stages:

1. Probe of a public DNS server on the network, via the HINFO query
2. Breakin-to the DNS server via the sadmind vulnerability
3. FTP upload of mstream DDoS software and the attack script
4. Initiate the attack on the other hosts of the network
5. Launching the DDoS

DARPA2000 is used to evaluate the ability and accuracy of iCorrelator to extract the multi-step attack scenarios. As a result of the limited number of alerts in both Inside1 and Inside2, another dataset with more alerts is used to evaluate the performance of iCorrelator. The netForensics honeynet dataset contains 35 days of traffic logs collected from February 25, 2005 to March 31, 2005. During this period, attackers issued several multi-step attacks to compromise the honeynet. Here, the word compromised is defined as a successful attack, followed by some follow-up activities.

From the honeynet owner's point of view, the most compelling evidence of compromise was the outbound IRC communication, which implies that the intrusion succeeded, the attacker has some degree of control over the machine and that he managed to install his own software (an IRC client or Bot). The owner of the honeynet also pointed out that their victim server was first compromised on February 26 and then continued in March. The traffic of the first two days of netForensics honeynet data is employed to test the ability of iCorrelator to extract the attack scenarios. netForensics is also used to evaluate the performance of the system.

Twenty one general rules are used in the rule set, and their corresponding twenty one training antigens are used for AIRS training phase. Before starting the correlation process, the AIRS algorithm is executed, and the generated memory cells are stored for future use. Therefore, the initial knowledge of the system consists of the fuzzy rules and the generated memory cells.

TABLE 4. The system parameter setting during the experiments

AIRS parameters		**Correlation parameters**	
Number of lymphocytes	200	Correlation Threshold	0.5
Stimulation Threshold	0.95	Correlation Sensitivity	0.1
Mutation Rate	0.1	Rule Selection Threshold (rs)	0.9
Hyper Mutation Rate	3	Cell Matching Threshold (cm)	0.97
Clonal Rate	10	**Time window parameters**	
Number of Initial Memory Cells	1	Number of Time Slots (n)	20
Affinity Threshold Scalar	0.2	Width of Each Slot in Seconds (W_s)	300
Number of Nearest Neighbors	3	Minimum Acceptable Probability (min_accept)	0.75

Three measures are used to evaluate the accuracy of the system: completeness, soundness and false correlation rate. Each one is defined as below [14]:

$$Completeness = \frac{(correctly\ correlated\ alerts)}{(related\ alerts)}$$

$$Soundness = \frac{(correctly\ correlated\ alerts)}{(total\ correlated\ alerts)}$$

$$False\ Correlation\ rate = \frac{(incorrectly\ correlated\ alerts)}{(related\ alerts)}$$

where, the correlated alerts are all alerts in the extracted scenario and the related alerts are all alerts in the desired complete scenario.

To evaluate the accuracy of the system, it is examined with two mentioned scenarios (LLDoS1 and LLDoS2). Table 4 shows the system parameter setting during the experiments. The most important parameters that influence the accuracy and performance of the system are the *number of alerts*, the *number of cells (lymphocytes)*, the *correlation threshold*, the *rule selection threshold,* rs, the *cell matching threshold,* cm, the *number of time slots,* n, and the *width of each time slot,* W_s. The best results are obtained by the values that are shown in Table 4. The values of AIRS related parameters are selected based on the well known works on this algorithm [33, 34]. For other parameters, admissible values that are selected by trial and error are used. The system is executed 10 times for each dataset, and the results are reported based on the average values.

The *select all* and RDTW policy with three different configurations are used to evaluate the performance of the system. The numbers of the time slots, n, in all three configurations are 10, and the width of time slots, W_s, are 150, 300 and 600 Seconds. For each policy and configuration, the numbers of alerts are changed from 1000 to 5000 and the execution times are considered.

5.1. **Accuracy evaluation.** Both DARPA2000 and netForensics honeynet data are employed to examine the accuracy of iCorrelator. Two different alert selection policies are investigated: the *select all* policy and RDTW with $n = 20$ and $W_s = 300$. The results are compared with the results reported by Ning et al. [14] and Al-Mamory et al. [36]. These two works use the same dataset (DARPA2000) and the same measures (soundness and completeness) for their accuracy evaluation.

The results for LLDoS1.0 show that the alerts in this scenario are from six different types: *Sadmind_Ping*, *Sadmind_Amslverify_Overflow*, *Admind*, *Rsh*, *Mstream_Zombie* and *Stream_DoS*. The first five alert types are appeared in all extracted scenarios with both policies. The last step of the attack is a *Stream_DoS* alert. It is the only alert that is not correlated with other alerts. It is placed in a hyper-alert with only one alert.

Table 5 shows the average of completeness, soundness and false correlation rate for iCorrelator with *select all* and RDTW policies in processing the inside1 data. The results reported by Ning [14] and Al-Mamory [36] are also provided in Table 5 for comparison. iCorrelator with *select all* policy, offers the highest completeness. It shows the ability of three-layer architecture in correctly correlating the most number of alerts. The mean completeness is 0.941. It is close to the completeness reported by Ning et al. (0.932) [14] and is considerably better than the completeness reported by Al-Mamory et al. (0.833). It is important to consider that in [14] prerequisites and consequences are used as the basic mechanism for correlation. Extensive knowledge of network security and different attack types are required in order to define the prerequisites and consequences of different steps of attacks. Also it needs to maintain this knowledge up to date. It is hard and time consuming and need extensive technical knowledge. This process is done by iCorrelator without the predefined rules about different attack types and their relations. The soundness and false correlation rate with *select all* policy are 0.950 and 0.050 respectively. They are 0.932 and 0.068 for Ning et al. and 1.0 and 0.0 for Al-Mamory et al. The completeness for RDTW is less than all other methods (0.816). By considering the extracted attack scenarios, it is obvious that the lower completeness for iCorrelator with RDTW policy is not influential. Because, most missing alerts are Rsh alerts. There are 17 alerts of this type in LLDoS1.0. The decline of the completeness in the RDTW is the consequence of the missing Rsh. In spite of the missing of few Rsh alerts, iCorrelator is able to extract the scenario correctly. Figure 6(a) shows the extracted scenarios with RDTW and *select all* policy for LLDoS1.0. The extracted graphs for both policies are the same with a little difference in the weights of the edges. Thus, the less completeness for RDTW is not influential in scenario extraction. Figure 6(b) shows the graph reported in [24] for comparison. The extracted graphs (regardless of the weights) are the same with both selection policies. As we will see later in Section 5.2 the performance of the system with RDTW is much better than the system with *select all* policy.

TABLE 5. Accuracy comparison for LLDoS1.0

	Select All	*RDTW*	*Ning* [14]	*Al-Mamory* [36]
Completeness	0.941	0.816	0.932	0.833
Soundness	0.950	0.943	0.932	1.000
False Correlation	0.050	0.052	0.068	0.000

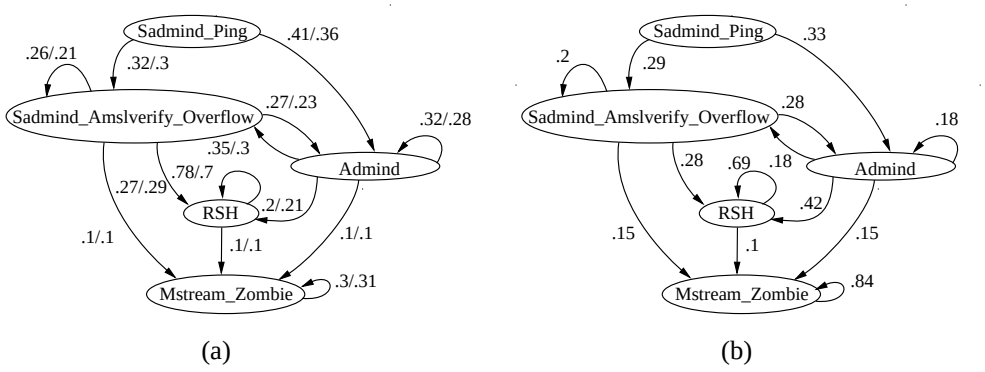

FIGURE 6. The attack graph generated for LLDoS1.0 by iCorrelator (a) and by Zhu et al. (b) [24]. The results for iCorrelator are for two policies: RDTW ($W_s = 300$, $n = 20$) and *Select All* (separated by slash).

It is worth pointing out that the different weights for graphs generated by different methods are unavoidable. Each method uses different ways for probability calculation and also different policies for alert selection. The different sequences of random numbers and random selections generate different results in each execution even for one specific method.

Both mentioned alert selection policies are also used to examine the Inside2 traffic. Again both policies are able to extract the attack scenario almost completely (except the last step). Alerts that appear in all extracted scenario are *Admind*, *Sadmind_Amslverify_Overflow*, *FTP_Put* and *Mstream_Zombie* for both policies. The last step of the attack is not extracted in all experiments. Its related alert (*Stream_DoS*) is placed in a hyper-alert with only one alert.

Table 6 shows the average of completeness, soundness and false correlation rate for iCorrelator with *select all* and RDTW policies in processing the inside2 data. Again the results reported by Ning [14] and Al-Mamory [36] are provided for comparison. Although Almamory et al. report the highest completeness (0.875), their reported soundness (0.583) is considerably less than iCorrelator and their reported false correlation rate (0.625) is considerably higher. The completeness for RDTW is less than its corresponding value for *select all* policy, but its soundness and false correlation rate (0.982 and 0.014) are the best among all. The results for iCorrelator are close to the results reported by Ning et al. In spite of the less completeness for iCorrelator, it is able to extract the LLDoS2.0 scenario correctly. Figure 7(a) shows the extracted scenarios with RDTW and *select all*. The extracted graphs for both policies are the same with a little difference in the weights on the edges. Hence, the less completeness for RDTW is not influential in scenario extraction. Figure 7(b) shows the graph reported in [24] for comparison. The extracted graphs (regardless of the weights of the edges) are the same with both selection policies.

TABLE 6. Accuracy comparison for LLDoS2.0

	Select All	*RDTW*	*Ning* [14]	*Al-Mamory* [36]
Completeness	0.607	0.600	0.667	0.875
Soundness	0.937	0.982	0.923	0.583
False Correlation	0.050	0.014	0.056	0.625

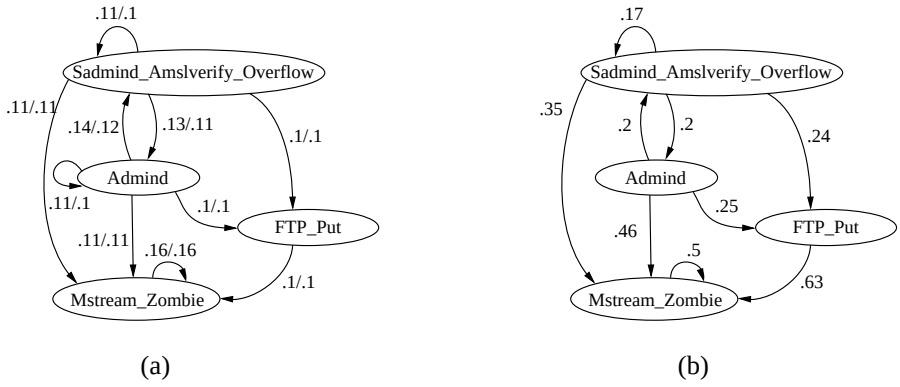

FIGURE 7. The attack graph generated for LLDoS2.0 by iCorrelator (a) and by Zhu et al. (b) [24]. The results for iCorrelator are for two policies: RDTW ($W_s = 300$, $n = 20$) and *Select All* (separated by slash).

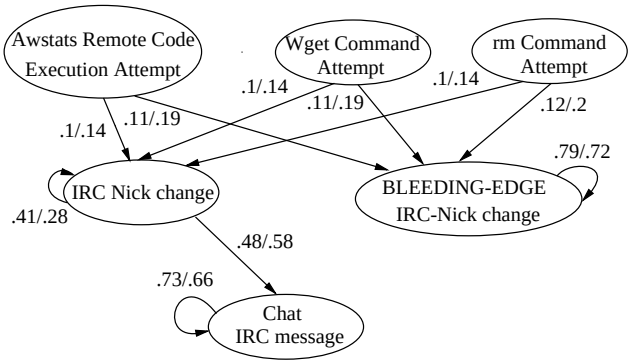

FIGURE 8. The attack graph generated for netForensics honeynet with RDTW policy ($W_s = 300$, $n = 20$) and *Select All* policy (separated by slash)

iCorrelator is also examined with netForensics honeynet data. Snort generates 3419 alerts belonging to 43 different alert types for the first two days of this data. Results show that all 43 types of alerts in the input data are correlated with each other with different strengths. As mentioned before, the most compelling evidence of compromise is the outbound IRC communication, which implies that the intrusion succeeded. Our extracted scenario is started by three alert types: *WEB-ATTACKS rm command attempt*, *BLEEDING-EDGE EXPLOIT Awstats Remote Code Execution Attempt* and *WEB-ATTACKS wget command attempt*. The attacker uses these remote command attempts to download and install malicious software on the target machines. Then the attacker issues IRC attacks from those compromised targets to the final victim. Snort is produced alerts such as *CHAT IRC nick change*, *BLEEDING-EDGE IRC-Nick change on non-std port* and *CHAT IRC message* for the rest of the attack, and our system correlates these alerts. Figure 8 compares the extracted attack scenarios with RDTW and *select all* policies.

5.2. **Performance evaluation.** There are two performance considerations for iCorrelator. Firstly, the correlation method in second layer of correlation is learning-based. Generally, statistical and learning-based methods are time consuming and are not usually applicable for online correlation. Secondly, using the nested loops in order to examine the previous alerts for correlation are time consuming in large datasets.

The first problem is solved by using three layers of correlation. According to our experimental results for iCorrelator, more than 98 percent of all correlations go through the rule-based layer or memory-based layer of correlation and less than 2 percent go through the learning-based layer. Moreover, iCorrelator uses an enhanced alert selection policy to address the second problem. RDTW is used to reduce the execution time and improve the performance of iCorrelator without (or with minimum) accuracy degradation.

The netForensics honeynet is used to evaluate the performance of iCorrelator with RDTW and *select all* policy. Three different settings for the number of time slots (n) and the width of the time slots (W_s) are used. Also the number of alerts is changed from 1000 to 5000, and the execution time is reported. All experiments are done on a system with an Intel Core 2 Duo, 2.0 GHZ processor and 1 MB of main memory. Table 7 shows the comparing results of the execution time. It is observable in Table 7 that the execution time with *select all* policy is $O(n_a^2)$, where n_a is the number of alerts. For example for 1000 alerts the execution time is 16 seconds, and with 2000 alerts the execution time is 64 seconds.

TABLE 7. Comparing the execution time (in seconds) with different time window

Number of Alerts	*Select All*	W_s = *600 s*, n = *10*	W_s = *300 s*, n = *10*	W_s = *150 s*, n = *10*
1000	16	10	8	7
2000	64	25	21	16
3000	146	48	44	30
4000	258	91	79	51
5000	402	158	133	90

As it is expected, the execution time with RDTW policy is decreased considerably. Table 7 shows that even for the small number of alerts the difference between two policies is considerable. The execution time for 1000, 2000 and 3000 input alerts with $W_s = 600$ Seconds and $n = 10$ (time window of 6000 seconds) are 10, 25 and 48 seconds respectively, and with $W_s = 150$ Seconds and $n = 10$ (time window of 1500 seconds) are 7, 16 and 30. It shows that the time complexity is approximately linear, and it is applicable for online correlation. The time window size is an important factor for the performance and accuracy of the system. It is possible to adjust the performance and accuracy based on the conditions in use. For example, a higher rate of alerts implies a shorter time window, a more important asset in the protected network implies a longer time window, and a longer time window needs more processing resources. The time window is controlled by the number of time slots (n), the width of each time slot (W_s) and the *min_accept* value. In the current version of iCorrelator, the adjustment of the parameters is done manually by the administrator.

6. **Conclusion and Future Work.** In this paper, iCorrelator, an immune-inspired architecture for alert correlation is presented. It is based on the three layers of correlation that is inspired by the three different responses in the human immune system. In the first layer, an agile rule-based correlation system is used. It contains a very limited number of general rules which are used to address some typical cases. An immune inspired learning algorithm along with a group of matrices provides an incremental learning method in second layer of correlation. It is used to manage the previously unencountered cases. The correlation results in the second layer of correlation are stored in a pool of immune memory cells and are used in the third layer of correlation. Considerable performance improvement is obtained by using the experiences which are stored in the pool of immune memory cells. iCorrelator is simple to set up. It does not need complicated initial knowledge. It is able to learn the correlation probability between different attack types, and is able to remember this knowledge for future usages.

A new attribute weighting algorithm, named *average contribution* is also presented. It is used in the AIRS algorithm to improve its accuracy. Moreover, a new alert selection policy is used to improve the performance of the system. It is based on a time window and a directed random alert selection method.

iCorrelator is examined by two DARPA2000 traffic data and a netForensics honeynet data. The completeness, soundness and false correlation rate are examined for evaluating its accuracy. The results are compared with other similar works. In spite of its simple set up, iCorrelator is able to correlate the alerts accurately and extract the attack scenarios correctly. The performance of iCorrelator is also examined by different number of alerts. While the complexity of the correlation process with *select all* policy is $O(n_a^2)$ (n_a is the number of alerts), it is almost linear with the RDTW policy. Although RDTW policy decreases the time complexity of the system, it does not decrease its accuracy.

More works are required in order to examine the system for online correlation. It is also useful to work on automatic parameter selection for time window management.

REFERENCES

[1] A. Ghorbani, W. Lu and M. Tavallaee, *Network Intrusion Detection and Prevention*, 1th Edition, Springer, New York, 2010.

[2] F. Valeur, G. Vigna, C. Kruegel and R. A. Kemmerer, A comprehensive approach to intrusion detection alert correlation, *IEEE Transactions on Dependable and Secure Computing*, vol.1, pp.146-169, 2004.

[3] A. Valdes and K. Skinner, Probabilistic alert correlation, *Recent Advances in Intrusion Detection*, vol.2212, pp.54-68, 2001.

[4] F. Maggia, M. Matteucci and S. Zanero, Reducing false positives in anomaly detectors through fuzzy alert aggregation, *Inform. Fusion*, vol.10, no.4, pp.300-311, 2009.

[5] K. Julisch, Clustering intrusion detection alarms to support root cause analysis, *ACM Trans. Inf. Syst. Secur.*, vol.6, pp.443-471, 2003.

[6] T. Pietraszek, Using adaptive alert classification to reduce false positives in intrusion detection, *Lecture Notes in Computer Science*, vol.3224, pp.102-124, 2004.

[7] S. Manganaris, M. Christensen, D. Zerkle and K. Hermiz, A data mining analysis of rtid alarms, *Comput. Networks*, vol.34, no.4, pp.571-577, 2000.

[8] J. Viinikka, H. Debar, L. M. A. Lehikoinen and M. Tarvainen, Processing intrusion detection alert aggregates with time series modeling, *Inform. Fusion. Special Issue on Information Fusion in Computer Security*, vol.10, no.4, pp.312-324, 2009.

[9] B. Morin, L. M. H. Debar and M. Ducass, M2D2: A formal data model for ids alert correlation, *Proc. of the 5th International Conference on Recent Advances in Intrusion Detection*, pp.115-137, 2002.

[10] B. Morin, L. M. H. Debar and M. Ducass, A logic-based model to support alert correlation in intrusion detection, *Inform. Fusion. Special Issue on Information Fusion in Computer Security*, vol.10, no.4, pp.285-299, 2009.

[11] P. A. Porras, M. W. Fong and A. Valdes, A mission-impact-based approach to infosec alarm correlation, *Proc. of the 5th International Conference on Recent Advances in Intrusion Detection*, vol.2516, pp.95-114, 2002.

[12] O. Dain and R. Cunningham, Fusing a heterogeneous alert stream into scenarios, *Proc. of the 2001 ACM Workshop on Data Mining for Security Applications*, pp.1-13, 2001.

[13] L. Wang, A. Ghorbani and Y. Li, Automatic multi-step attack pattern discovering, *Int. J. Netw. Secur.*, vol.10, no.2, pp.142-152, 2010.

[14] P. Ning, Y. Cui and D. S. Reeves, Techniques and tools for analyzing intrusion alerts, *ACM Transactions on Information and System Security*, vol.7, no.2, pp.274-318, 2004.

[15] S. Cheung, U. Lindqvist and M. Fong, Modeling multistep cyber attacks for scenario recognition, *Proc. of DARPA Information Survivability Conference and Exposition*, vol.1, pp.284-292, 2003.

[16] F. Cuppens and A. Miege, Alert correlation in a cooperative intrusion detection framework, *Security and Privacy, IEEE Symposium on Security and Privacy*, pp.202-215, 2002.

[17] S. J. Templeton and K. Levitt, A requires/provides model for computer attacks, *Proc. of the 2000 Workshop on New Security Paradigms*, pp.31-38, 2000.

[18] X. Qin, *A Probabilistic-Based Framework for INFOSEC Alert Correlation*, Ph.D. Thesis, Georgia Institute of Technology, 2005.

[19] H. Ren, N. Stakhanova and A. Ghorbani, An online adaptive approach to alert correlation, *Lecture Notes in Computer Science*, vol.6201, pp.153-172, 2010.

[20] R. Sadoddin and A. A. Ghorbani, An incremental frequent structure mining framework for real-time alert correlation, *Comput. Secur.*, vol.28, no.3-4, pp.153-173, 2009.

[21] S. T. Eckmann, G. Vigna and R. A. Kemmerer, Statl: An attack language for state-based intrusion detection, *J. Comput. Secur.*, vol.10, no.1-2, pp.71-104, 2002.

[22] F. Cuppens and R. Ortalo, Lambda: A language to model a database for detection of attacks, *Lecture Notes in Computer Science*, vol.1907, pp.197-216, 2000.

[23] E. Totel, B. Vivinis and L. Mé, A language driven intrusion detection system for event and alert correlation, *IFIP International Federation for Information Processing*, vol.147, pp.208-224, 2004.

[24] B. Zhu and A. Ghorbani, Alert correlation for extracting attack strategies, *Int. J. Netw. Secur.*, vol.3, no.3, pp.244-258, 2006.

[25] L. de Castro and J. Timmis, *Artificial Immune Systems: A New Computational Approach*, Springer-Verlag, London, UK, 2002.
[26] S. A. Hofmeyr, *An Immunological Model of Distributed Detection and Its Application to Computer Security*, Ph.D. Thesis, University of New Mexico, 1999.
[27] J. Kim and P. Bentley, Toward an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection, *Proc. of the 2002 Congress on Evolutionary Computation*, Honolulu, HI, USA, pp.1015-1020, 2002.
[28] S. Schaust and H. Szczerbicka, Artificial immune system in the context of misbehavior detection, *Cybernetics and Systems*, vol.39, no.2, pp.136-154, 2008.
[29] H. Fu, X. Yuan, K. Zhang, X. Zhang and Q. Xie, Investigating novel immune-inspired multi-agent systems for anomaly detection, *Asia-Pacific Conference on Services Computing*, pp.466-472, 2007.
[30] C. Lin, Y. Liu and C. Lee, An efficient neural fuzzy network based on immune particle swarm optimization for prediction and control, *International Journal of Innovative Computing, Information and Control*, vol.4, no.7, pp.1711-1722, 2008.
[31] M. Bateni, A. Baraani and A. Ghorbani, Alert correlation using artificial immune recognition system, *Int. J. Bio-Inspired Computation*, vol.4, no.3, pp.181-195, 2012.
[32] M. Bateni, A. Baraani and A. Ghorbani, Using artificial immune system and fuzzy logic for alert correlation, *Int. J. Netw. Secur.*, vol.15, no.1, pp.160-174, 2013.
[33] A. Watkins, *AIRS: A Resource Limited Artificial Immune Classifier*, Master Thesis, Mississippi State University, 2001.
[34] A. Watkins, J. Timmis and L. Boggess, Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm, *Genetic Programming and Evolvable Machines*, vol.5, no.3, pp.291-217, 2004.
[35] H. Ahmadinejad and S. Jalili, Alert correlation using correlation probability estimation and time windows, *Proc. of the 2009 International Conference on Computer Technology and Development*, vol.2, pp.170-175, 2009.
[36] S. O. Al-Mamory and H. L. Zhang, Building scenario graph using clustering, *Proc. of the 2007 International Conference on Convergence Information Technology*, pp.799-804, 2007.