Using Artificial Immune System and Fuzzy Logic for Alert Correlation

Mehdi Bateni[1], Ahmad Baraani[1], and Ali Akbar Ghorbani[2]
(Corresponding author: Mehdi Bateni)

Department of Computer Engineering, University of Isfahan[1]
Hezar Jerib St., Isfahan, 81746-73441, Iran
Faculty of Computer Science, University of New Brunswick[2]
550 Windsor Street, Fredericton, New Brunswick, Canada
(Email: Bateni@eng.ui.ac.ir)

(Received Feb. 25, 2011; accepted July 13, 2011)

Abstract

One of the most important challenges facing the intrusion detection systems (IDSs) is the huge number of generated alerts. A system administrator will be overwhelmed by these alerts in such a way that she/he cannot manage and use the alerts. The best-known solution is to correlate low-level alerts into a higher level attack and then produce a high-level alert for them. In this paper a new automated alert correlation approach is presented. It employs Fuzzy Logic and Artificial Immune System (AIS) to discover and learn the degree of correlation between two alerts and uses this knowledge to extract the attack scenarios. The proposed system doesn't need vast domain knowledge or rule definition efforts. To correlate each new alert with previous alerts, the system first tries to find the correlation probability based on its fuzzy rules. Then, if there is no matching rule with the required matching threshold, it uses the AIRS algorithm. The system is evaluated using DARPA 2000 dataset and a netForensics honeynet data. The *completeness*, *soundness* and *false alert rate* are calculated. The average completeness for LLDoS1.0 and LLDoS2.0, are 0.957 and 0.745 respectively. The system generates the attack graphs with an acceptable accuracy and, the computational complexity of the probability assignment algorithm is linear.

Keywords: Alert correlation, artificial immune system, fuzzy logic, intrusion detection system

1 Introduction

Intrusion Detection System (IDS) is a rapidly growing field that deals with detecting and responding to malicious network traffic and computer misuse. Intrusion detection is the process of identifying and (possibly) responding to malicious activities targeted at computing and network resources [7].

Based on their functionality IDSs are divided into two categories, misuse detection and anomaly detection systems. Misuse detection systems use a database of known attack signatures, then compare any new activity by this database and decide about its safety status. On the other hand, anomaly detection systems use a profile of normal behavior for each user or system, and compare each new activity with the normal profile. Any notable changes or anomalies could be considered as a possible attack [1].

The number of false alerts for the misuse detection systems is less, but they cannot identify new attacks. On the other hand, anomaly detection systems can detect some new attacks, but the rate of false alarms for them is higher. Both types of IDSs have a more serious common problem: the huge and unmanageable number of produced alerts. In most cases the large number of low-level alerts confused the system administrator. Each alert has a little information, and if there are a large number of these alerts that contain little information then the system administrator may ignore alerts because she/he cannot handle a large number of alerts.

The best known solution for this problem is to correlate alerts with each other and create higher level scenarios. Alert correlation is the process of analyzing alerts that are produced by one or more IDSs to provide a more succinct and concise high-level view of the occurring or attempted intrusions [28]. The most important goal of alert correlation is to reduce the number of alerts that the administrator should investigate manually to find the signs of attacks. The administrator prefers to have a high-level scenario of an attack instead of a large numbers of low-level alerts.

An alert correlator usually carries out its job by removing false alerts, aggregating related alerts and prioritizing alerts. Most of the correlators use a complex knowledge base of rules that define the relationship between alerts and store metadata about the protected network. Thus it has to use some expert people to enter the proper knowledge in the knowledge base. It is hard work and needs a deep knowledge of network and security. Also considering the changes in network configuration and everyday new-appearing attacks, it has to maintain this knowledge up-to-date. It is also a hard, time-consuming and error-prone work.

In this paper we propose an alert correlator, which uses a combination of predefined fuzzy rules and dynamic learning-based solution. To facilitate the rule definition process a limited number of general rules are used. The number of the rules in our implementation is limited to 31 and the rules are not related to any specific network configuration or any specific attack type or predefined scenario. Besides the predefined rules there is a learning subsystem, which uses Artificial Immune Recognition System (AIRS) algorithm. AIRS is trained using the predefined fuzzy rules in order to discover and remember the correlation relationship between each two alert types. For a new alert, the system finds its correlation with the previous alerts. Firstly, it tries to find a rule in its predefined rule set with matching value higher than a tuneable threshold (rule selection threshold). If it cannot find a proper rule, then it uses the AIRS. AIRS uses the same fuzzy rules as input to generate a collection of memory cells. The correlation system uses these memory cells to find correlation probability.

After correlating two alerts, the system stores its experiences about these two alert types and their correlation in three matrices, Alert Correlation Matrix (ACM), forward strength correlation matrix (Π^f) and backward strength correlation matrix (Π^b). The new values of these three matrices affect the calculation of probability of correlation for future alerts. It is also possible to use the ACM and Π^f to extract the attack scenario and to create the attack graph.

The proposed system needs no deep knowledge about attacks and the protected network. Also, it is a self-organizing system with the ability to adapt to the changes in order to detect new attacks and scenarios. As mentioned before, the system uses AIRS as its learning algorithm. The algorithm is very fast and has a low computational cost, so the overall computational cost of the system depends on the correlation algorithm and not the learning and correlation probability estimation algorithm.

For each new alert (a_i), the system searches all hyper-alerts (groups of related alerts) for an alert (a_j) with the highest correlation probability with a_i. If correlation probability between a_i and a_j is greater than a given threshold, then a_i is inserted in the hyper-alert which contains a_j. Otherwise, a new hyper-alert is created and a_i is inserted in it.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents our system. It illustrates the architecture of the system and provides the details of its components. Section 4 reports the result of running the system with the DARPA2000 and netForensics honeynet data. Finally, Section 5 provides the conclusion and some suggestions for future work.

2 Related Works

As mentioned before, alert correlation has two main goals: reducing the number of alerts and increasing the relevance and abstraction level of the produced reports [28]. Commonly used techniques for alert correlation can be categorized as follows:

- Fusion-based
- Filter-based
- Causality-based

Fusion-based correlation [10, 13, 27] 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 some threshold, alerts are placed in one cluster. Filter-based approaches [14, 15, 16, 19, 20, 29] 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. Priorities are usually assigned to alerts depending on how important attacked assets are. Causality-based approaches use the logical relationships between alerts to correlate them [2, 3, 5, 18, 21, 22, 23, 24, 30]. They either use the knowledge of experts to find related alerts or aim to infer it from the statistical or machine learning analysis. Because our work is more related to the causality-based approach, we focus on the work that uses 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., LAMDBA [4], STATL [6], ADeLe [26]) or learned using machine learning techniques [5, 30]. Specifying all scenarios in advance is time-consuming and error-prone work and needs a deep knowledge of the domain. Moreover, it has problem with the new attack pattern. Wang et al. [30] proposed a multi-step attack pattern discovering method that aims at solving the problems of new attack pattern discovery and 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 use the rule-based correlation approach [2, 3, 18, 24]. 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 [21, 22, 23, 33] 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 [21] presented a Bayesian correlation engine for discovering the statistical relationship between alerts. They analyze 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 among 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. [22] 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 algorithm to estimate the correlation probability among alerts and use it in correlation time. Zhu et al. [33] used Multilayer Perceptron and Support Vector Machines to estimate the alert correlation probability, and Sadoddin et al. [23] 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. However, the most important drawback is their high computational cost, which makes them impractical for online computation.

3 The Proposed System

The main goal of the alert correlation process is to reduce the number of alerts that the system administrator encounters and has to handle manually. It is a complex and multifaceted problem, and there are many different ways to face it. In this paper we introduce a combinatory fuzzy and AIS-based solution.

3.1 An Overview

The goal of our system is to assign a correlation probability to each pair of input alerts and to use this probability for next correlation. In order to accomplish this, firstly, It creates a feature vector from each pair of input alerts, then the system searches among its rules for a rule that matches with this vector with the value higher than the rs threshold (rule selection threshold). If it finds such rule, then simply uses the probability in that rule as the correlation probability of two alerts. Otherwise, it uses AIRS algorithm to find the correlation probability. The AIRS algorithm is a supervised learning algorithm that is able to classify the unseen data based on its previous training data. Here the training data is a set of fuzzy rules and AIRS discovers and remembers the relationships between the values of the features in the rules. The correlation probability that is produced by fuzzy rules or AIRS is stored in three matrices as the experience of the system about these two alert types. If the correlation probability is higher than a predefined threshold (correlation threshold) then the incoming alert will be added to the collection of alerts, which contains the alert previously encountered. The name for this collection of related alerts is a hyper-alert. If the correlation probability for two alerts is less than the correlation threshold, then a new hyper-alert is created and the new alert is inserted into it. Figure 1 illustrates the overall structure of the correlation system. The system contains 6 main components.

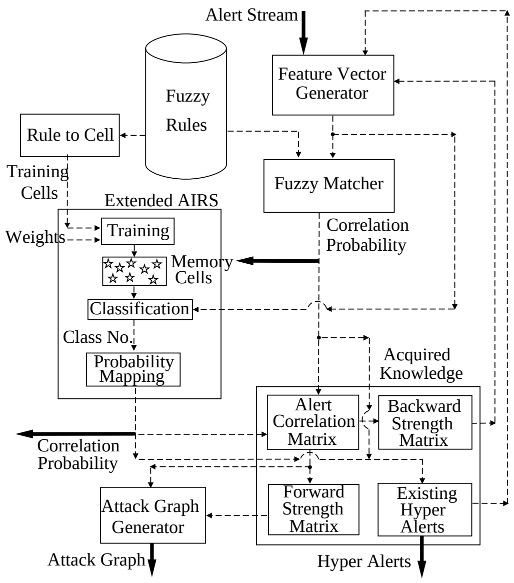

Figure 1: The architecture and components of the system

- Collection of fuzzy rules;
- Feature vector (cell) generator;
- Fuzzy rule matcher;
- AIS-based classifier;
- Hyper-alert generator and acquired knowledge;
- Attack graph generator.

As mentioned before, the system uses two classifiers in correlation probability calculation: fuzzy rule matcher and AIS-based classifier. One of the most important parameter of our system is the rule selection threshold (rs). By changing the value of rs, we can change our system from a completely static system that uses only fuzzy rules to a completely learning-based system that uses AIRS algorithm. If the value of 0 is assigned to rs, then the system uses only fuzzy rule matcher and if the value of 1 is assigned to it, then the system uses only the AIS-based classifier. Although the static system may work rapidly and accurately in predefined cases it cannot work properly in undefined situation. It is better to have a dynamic system that learns other cases from the presented one. Then the learning-based algorithm can work properly in this undefined situation. By changing the value of rs from 0 to 1 we can change the balance from a pure fuzzy correlator to a fuzzy AIS-based correlator and a pure AIS-based correlator. Choosing the value of rs is an important task and may affect both accuracy and performance of the system.

3.2 Feature Selection

Before starting its work, the correlator needs to extract some useful information from alert stream. Suppose that a_2 is the last generated alert by an IDS, and a_1 is an alert that is chosen from one existing hyper-alerts to investigate its correlation with a_2. To decide about the correlation of a_2 with a_1 the value of some feature in two alerts should be selected to generate a feature vector (we call the feature vector also cell or antigen in some cases). Seven features are chosen for this proposes. Five of them are calculated directly from a_1 and a_2, and two of them are calculated from the history of the experience of the system about alerts similar to a_1 and a_2. This experience is stored in three matrices ACM, Π^b and Π^f. Selected features are as below.

- F_1: Similarity between source IP addresses of a_1 and a_2 (between 0 and 1);
- F_2: Similarity between destination IP addresses of a_1 and a_2 (between 0 and 1);
- F_3: Equality of destination port of a_1 and a_2 (0 or 1);
- F_4: Equality of destination IP address of a_1 with the source IP address of a_2 (0 or 1);
- F_5: Backward strength correlation between alerts of type a_1 and a_2 (between 0 and 1);
- F_6: Correlation frequency between alerts with the same type as a_1 and a_2 (between 0 and 1);
- F_7: Freshness of a_1 in the arrival time of a_2 (between 0 and 1).

F_1-F_6 are adopted from [33]. F_7 is added to the feature vector in order to add the time as an important parameter to the correlation system. Features F_1 and F_2 are the similarity of IP addresses for source and destination addresses. To calculate the similarity between two addresses, it should count the number of common higher order bits of two addresses and divide it by 32. The values of these features are between 0 and 1. For example, two IP addresses 192.168.10.60 and 192.168.42.25 have 18 similar higher order bits, and the similarity of 0.56.

Next two features (F_3 and F_4) are about the port number of the source and destination addresses. Because the destination port number of two alerts are either equal or not, the value of F_3 is either 1 or 0. The value of F_4 is also either 0 or 1. F_4 is important, because in multi-step attacks the success in one step is the precondition of starting the next step, and usually the attacker tries to compromise one host and use it to compromise the next one. Thus, equivalence of the target IP address of a_1 with the source IP address of a_2 may indicate a multi-step attack.

F_5 is backward correlation strength. It is the probability of correlation between two alerts a_1 and a_2 when a_1 has been seen before a_2. This value is extracted from a matrix with the same name. The matrix initially is set to zero, and during the process of correlation it is updated with proper values according to the process that will be described later in Section 3.3.

F_6 is correlation frequency. If two types of alert frequently are correlated, then it is acceptable to say that there is a meaningful correlation between them. Subsequently if we have two choices of correlation with equal values in all other features, then it is acceptable to choose the alert with the higher value of F_6 for correlation. Initially F_6 is 0. During the correlation process with each correlation between a_1 and a_2, the F_6's value for these two types of alerts is increased.

F_5 and F_6 together can be used to improve the process of correlation, especially when the other feature values are not strong enough. Assume that two alerts of type t_1 and t_2 have occurred ten times before, and seven of these occurrences have led to correlation because of strong value in the other features other than F_5 and F_6. If two alerts of these types occur for the 11^{th} time with the weak values of other features, then because of the strength value of F_5 and F_6, it is possible to correlate them without the high value in other features. On the other hand, after several observations of two alerts, the system learns that they are in correlation with each other even if the other features are not strong enough [33].

F_7 is freshness. It is about alerts' arrival times. When a new alert arrives, it is fresh and its freshness value is 1, which indicates a possible new attack. Over time, the level of freshness declines, and after a definable time, the freshness reaches zero. This feature is added in order to add the time as an important feature to the correlation process. It increases the correlation probability of an alert with the most recently arrived alerts. Even after the freshness reaches zero, an alert can be correlated with other alerts if its other 6 features are high. A parameter, t, is defined to adjust this feature. The freshness value of a_1 with respect to the arrival time of a_2 can be calculated by using Equation (1) [11]. We consider t=3600 then F_7 reaches zero after one hour.

$$F_7{=}1-\sqrt{\frac{(a_2.time-a_1.time)}{t}} \tag{1}$$

Therefore, when two alerts a_1 and a_2 arrive the system delivers them to the feature vector generation module. It extracts required features from alerts and from stored

$$\begin{array}{c|cccc|} & a_1 & a_2 & a_3 & a_4 \\ a_1 & W_{c(a_1,a_1)} & W_{c(a_1,a_2)} & W_{c(a_1,a_3)} & W_{c(a_1,a_4)} \\ a_2 & W_{c(a_2,a_1)} & W_{c(a_2,a_2)} & W_{c(a_2,a_3)} & W_{c(a_2,a_4)} \\ a_3 & W_{c(a_3,a_1)} & W_{c(a_3,a_2)} & W_{c(a_3,a_3)} & W_{c(a_3,a_4)} \\ a_4 & W_{c(a_4,a_1)} & W_{c(a_4,a_2)} & W_{c(a_4,a_3)} & W_{c(a_4,a_4)} \end{array}$$

Figure 2: An Alert Correlation Matrix

matrices and creates a feature vector. Suppose for alert a_1 the timestamp, source address, destination address and alert type are 4:13:20, 172.16.114.50 : 1227, 172.16.113.50 : 25 and Email_Ehlo, and for a_2 are 5:06:16, 172.16.113.50 : 1048, 172.16.112.50 : 21 and FTP_User. Their corresponding feature vector is (F_1=.6872, F_2=.7187, F_3=0, F_4=1, F_5=.2424, F_6=1, F_7=.06).

3.3 Knowledge Acquiring Matrices

In this section, three matrices that are used in the correlation process are introduced: the Alert Correlation Matrix (ACM), the forward correlation strength matrix (Π^f) and the backward correlation strength matrix (Π^b). ACM contains the correlation weights between every two alerts. For example, if possible alerts in the system are a_1 to a_4, then ACM is as shown in Figure 2.

The ACM elements are the correlation weights of two corresponding alerts and are the sum of correlation probabilities for two alerts during the correlation process until now. It is calculated by using Equation (2) [33].

$$W_{c(a_i,a_j)} = \sum_{k=1}^{n} P_{i,j}\,(k), \tag{2}$$

where, $P_{i,j}(k)$ is the correlation probability for a_i and a_j in the kth correlation of them when a_i was occurred before a_j. Because of this time ordering, the ACM is not symmetric. It encodes the temporal relationship between two alerts. $P_{i,j}(k)$ is produced by the correlation engine, and for its calculation F_5 is used from the Π^b matrix. On the other hand, the calculated values in the ACM are used later for generating two strength matrices (Π^b, Π^f). Subsequently, ACM is updated dynamically by correlating each pair of new alerts, and the updated values cause the changes in the next correlation of these two alert types. Two strength matrices' elements are calculated by using Equation (3) and Equation (4) [33].

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

$$\Pi^f_{c(a_i,a_j)} = \frac{W_{c(a_i,a_j)}}{\sum_{k=1}^{n} W_{c(a_i,a_k)}} \tag{4}$$

Unlike correlation weights in the ACM, these two matrices' values are between 0 and 1. $\Pi^f(a_i, a_j)$ is calculated by dividing the correlation weight of a_i and a_j to the sum of correlation weights of a_i with all alerts that happened after a_i with it. It can be used to predict the correlation probability of one alert to another alert that happens after it. It will be used for generating the attack graph later.

On the other hand, $\Pi^b(a_i, a_j)$ is calculated by dividing correlation weight of a_i and a_j by the sum of correlation weights of all alerts that happened before a_j with it. It can be used to find the correlation probability of one alert with another alert that happened before it. As mentioned before, $\Pi^b(a_i, a_j)$ is used as one feature (F_5) in the process of feature vector generation. Both matrices initially are filled with zero, and the correlation process is done considering the other five features. After each correlation ACM, Π^b and Π^f matrices are changed. After several correlations, the contents of matrices are meaningful. These matrices play the role of some sort of memory or acquired knowledge for the correlation system.

3.4 Fuzzy Rules

We define a limited number of fuzzy rules in order to be able to assign a correlation probability to each feature vector. These rules declare the relation between seven features (F_1-F_7) and the class number (correlation probability). For example, one rule says 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 port of the first alert is not the same as the source port of the second alert and the frequency of previous correlation for alerts of these types are high and the backward correlation strength of these two types are high and the freshness of first alerts in the arrival time of the second one is high, then the class of the feature vector is 20 (means that probability of correlation is 1). The format of each rule is as following.

If $(F_1 = V_1)$ *and . . .* $(F_7 = V_7)$ *Then* $(Class = C)$

The antecedent of each rule contains seven features ($F_1 - F_7$) and their corresponding values ($V_1 - V_7$). Two features (F_3 and F_4) have crisp values (0 or 1). The value of five other features are expressed by linguistic terms such as *high*, *low* and *medium*. These linguistic terms are defined by proper fuzzy sets. The consequent of a rule is a class number that is assigned to it. The class number is an integer value between 1 and 20. The class number of 1 is equal to the probability value of 0 and the class number of 20 is equal to the probability value of 1. Each other class number simply can be mapped to its probability value with the step length of 0.05. Table 1 shows some sample rules of our defined rules. To classify an input vector such as x with the value of $(v_1, v_2, v_3, v_4, v_5, v_6, v_7)$ all rules are investigated and three rules with the most compatibility with the x are determined.

To calculate the compatibility of feature vector x with rule R_j we use the average membership value of the seven features with respect to rule R_j. It is calculated by using

Equation (5).

$$Compatibility(x, R_j) = \frac{1}{n}\sum_{i=1}^{n}\mu(v_i, V_i), \tag{5}$$

where n is the number of features, v_i is the value of ith feature in x, V_i is the value of ith feature in the antecedent part of rule R_j and μ is the membership function for the fuzzy set V_i. We calculate the compatibility of x with each rule to find the most compatible rule. If the compatibility value for the most compatible rule is more than a given rule selection threshold (rs), then the class number for x is determined by our fuzzy rule matcher. Otherwise, the system tries to calculate the correlation probability by using AIS-based classifier.

As mentioned before, the three most compatible rules are identified by the system. In the case of using the fuzzy rule matcher the probability value is calculated based on the class number in the consequent part of these three rules. First, we determine the class number, C_i, of x and then map it to a probability value. To calculate the class number, C_i, we consider not only the class number of three most compatible rules but also their compatibility value and their distances from each other. Algorithm 1 outlines the probability mapping function.

Algorithm 1 Probability calculation for rule x in fuzzy classifier

```
Begin
$R_1, R_2, R_3 \leftarrow$ 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*$\lambda$)
End
```

After determining the class number, C_i, we need a mapping function in order to convert a class number to a probability value. Equation (6) is used to do this.

$$P = \frac{\mathrm{C_i} - 1}{\lambda} + \frac{1}{2 * \lambda} \tag{6}$$

Table 1: Sample predefined rules

F_1	F_2	F_3	F_4	F_5	F_6	F_7	$Class$
Med	Med	1	0	High	High	Low	16
High	High	1	0	Low	Low	High	19
High	High	1	0	Low	Low	Low	18
Med	Med	0	0	Med	Low	High	4
Med	Med	0	0	Med	Low	Low	3

3.5 AIRS Algorithm

AIRS is a supervised-learning algorithm. It was introduced in 2001 for the first time by Watkins [31]. A revised version of it was introduced later [32]. It is more efficient than the original version, but with the same level of accuracy. We refer to this new version as AIRS in this paper. 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, memory cell retention. It also uses the resource limited artificial immune system concept. In this algorithm, the feature vectors presented for training and test are named as antigens while the system units are called as B cells. Similar B cells are represented with Artificial Recognition Balls (ARBs) which compete with each other for a fixed number of resources. The ARBs with higher affinities to the training antigen improve. Each antigen in training data is presented to algorithm once and the algorithm creates a memory cell for it. The memory cells formed after the presentation of all training antigens are used to classify test antigens.

The AIRS has four stages: Normalization and initialization; ARB generation; Competition for resources and nomination of candidate memory cell; and memory cell introduction [31, 32]. The mechanism to develop a candidate memory cell is as follows:

1) A training antigen is presented to all the memory

cells belonging to the same class as the antigen. The memory cell most stimulated by the antigen is cloned. The memory cell and all the recently generated clones are stored into the ARB pool. The number of clones generated depends on the affinity between the memory cell and antigen, and affinity in turn is determined by Euclidean distance between the feature vectors of a memory cell and a training antigen. The smaller the Euclidean distance, the higher the affinity, the more is the number of clones allowed.

2) Next, the training antigen is presented to all the ARBs in the ARB pool. All the ARBs are appropriately rewarded based on affinity between the ARB and the antigen. The rewards are in the form of number of resources. After all the ARBs have been rewarded, the sum of all the resources in the system typically exceeds the maximum number allowed for the system. The excess number of resources held by ARBs are removed in order starting from the ARB of lowest affinity and moving higher until the number of resources held does not exceed the number of resources allowed for the system. Those ARBs, which are not left with any resources, are removed from the ARB pool. The remaining ARBs are tested for their affinities towards the training antigen. If the average normalized stimulation level for all instances does not meet a user defined stimulation threshold, then the ARBs are mutated and their clones are placed back in the ARB pool. The mutation range for highly stimulated ARBs is more limited than the mutation range of less stimulated ARBs. (class mutation is not valid). Step 2 is repeated until the affinity meet the stimulation threshold.

3) The most stimulated ARB is chosen as a candidate memory cell. If its affinity for the training antigen is greater than that of the original memory cell selected for cloning at step 1, then the candidate memory cell is placed in the memory cell pool. If in addition to this the difference in affinity of these two memory cells is smaller than a user defined threshold, the original memory cell is removed from the pool.

These steps are repeated for each training antigen. After completion of training the test data are presented only to the memory cell pool, which is responsible for actual classification. The class of a test antigen is determined by majority voting among the k most stimulated memory cells, where k is a user defined parameter.

AIRS has been applied to a wide variety of publicly available classification benchmarks. AIRS proved to be a very good classifier, thus far it has been among the ten most accurate classifiers known in every case to which it has been applied [12]. In order to use the AIRS for alert correlation purpose we propose some improvements to it. The main goals of these improvements are to improve the accuracy of the algorithm for our usage and to enable AIRS to produce real value (probability) instead of integer value (class number) for its input antigens. In order to use the fuzzy rules as training cell for AIRS, we make a slight change in the rules that is shown in the Table 1. We replace the terms such as *high* and *low* with some appropriate real value such as 1 and 0 and make a vector of real value for each rule. This vector of real value is called antigen and is used in training process of AIRS. Table 2 shows some sample training antigens for our algorithm and Table 3 is some sample memory cells that are produced by the AIRS algorithm. These memory cells will be used later for probability calculation. We describe the improvements in the AIRS for alert correlation in more details in next three subsections.

Table 2: The training cells corresponding to rules of Table 1

F_1	F_2	F_3	F_4	F_5	F_6	F_7	*Class*
0.5	0.5	1	0	1.0	1.0	0.0	16
1.0	1.0	1	0	0.0	0.0	1.0	19
1.0	1.0	1	0	0.0	0.0	0.0	18
0.5	0.5	0	0	0.5	0.0	1.0	4
0.5	0.5	0	0	0.5	0.0	0.0	3

Table 3: Sample generated memory cells

F_1	F_2	F_3	F_4	F_5	F_6	F_7	*Class*
0.55	0.99	1	0	0.38	0.62	0.40	16
0.98	1.0	1	0	0.10	0.61	0.15	19
1.0	1.0	1	0	0.50	0.07	0.85	18
0.20	0.46	0	0	0.24	0.20	0.57	4
0.65	0.50	0	0	0.10	0.11	0.43	3

3.5.1 Weight Calculation

In distance calculation in the AIRS algorithm the weights of all features are equal, therefore, the stimulation of one cell (antigen) by the other is calculated by the following Equation

$$Stimulation(a_1, a_2) = 1 - Distance(a_1, a_2).$$

Where $Distance(a_1, a_2)$ is the Euclidean distance of two cells a_1 and a_2.

Since the number of classes is high and the number of training data are limited, a more accurate method for computing the distance values is needed. By examining the training data, we found out that the features do not have an equal effect in the calculation of the probability values.

To determine the effect of each feature, we use the notion of Symmetrical Uncertainty [8]. This score is a variation of the Information Gain measure. It compensates for InfoGains bias toward attributes with more values and normalizes its value to the range [0,1]. Symmetrical Uncertainty is defined by Equation (7).

$$SU(X,Y) = 2 \times \left[\frac{InformationGain(X|Y)}{H(X) + H(Y)} \right] \qquad (7)$$

where $H(X)$ and $H(Y)$ are the entropies of the random variables X and Y, respectively. Before using this score, all continuous attributes should be discretized into intervals. We discretized F_1, F_2, F_5, F_6 and F_7 into 10 intervals. Using Symmetric Uncertainity between the $Class$ and each feature (F_1-F_7) in the training dataset produced the following coefficients, which are used for weight estimation.

W_1=0.272, W_2=0.292, W_3=0.253, W_4=0.107, W_5=0.224, W_6=0.388, W_7=0.160. The weighted Euclidean distance is calculated by using Equation (8).

$$Distance\,(a_1, a_2) = \sqrt[2]{\sum_{\mathrm{i=1}}^{\mathrm{n}} W_i * (a_1.F_i - a_2.F_i)^2} \qquad (8)$$

Note that the weight estimation is done only once in the parameter discovery phase of the system.

3.5.2 Class Selection Policy

The other improvement that is applied to the last step of the AIRS algorithm is to change the policy of class selection. The standard version of AIRS uses the majority vote in the KNN algorithm. Our experimental results show that, replacement of majority vote selection with the least average distances selection improves the output of AIRS in the correlation engine. This means that for identifying the proper class label of a vector of data x, it is better to choose the class label with least average distances from the x, instead of the class label with the most members.

3.5.3 Probability Mapping

To map the class number to the probability value we use again Equation (6). We also consider the predecessor and successor class numbers of each class to calculate its accurate probability value. Suppose that the class label that is generated for an antigen Ag is C_i, and the average distance of Ag to C_i is d_i. If C_{i-1} is the predecessor and C_{i+1} is the successor class of C_i, then the distances of Ag with C_{i-1} and C_{i+1}, are d_{i-1} and d_{i+1}. Note that, it is possible that d_{i-1} or d_{i+1} do not exist because C_{i-1} and C_{i+1} are not necessarily one of the K nearest neighbors of Ag. By using d_{i-1} and d_{i+1}, Equation (6) is changed as Equation (9).

$$P = \frac{C_i - 1}{\lambda} + \frac{1}{2 * \lambda} + \Delta \qquad (9)$$

Where,

$$\Delta = \begin{cases} -\frac{1 + {d_i}^2 - {d_{i-1}}^2}{2 * \lambda} & \text{if } (d_{i-1} < d_{i+1}) \text{ or } (\nexists C_{i-1}) \\ 0 & \text{if } (d_i = 0) \text{ or } (d_{i-1} = d_{i+1}) \\ & \text{or } (\nexists C_{i-1} \text{ and } \nexists C_{i+1}) \\ \frac{1 + {d_i}^2 - {d_{i+1}}^2}{2 * \lambda} & \text{if } (d_{i+1} < d_{i-1}) \text{ or } (\nexists C_{i+1}) \end{cases}$$

In Equation (9) the initial probability is shifted toward one of the two classes, C_{i-1} or C_{i+1} respectively. The value of the shift is Δ, and the direction of the shift is dependent on the value of d_{i-1} and d_{i+1} (toward the one with the least value). By using Equation (9), the value of probability changes continuously between 0 and 1, and it would be accurate enough. Algorithm 2 outlines the probability assignment algorithm for an input cell.

Algorithm 2 Probability calculation for cell x in AIRS

```
Begin
$n \leftarrow$ 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
    $\Delta \leftarrow \frac{1 + d_i^2 - d_{i+1}^2}{2 * \lambda}$
end if
if $(\nexists C_{i+1})$ or $(d_{i-1} < d_{i+1})$ then
    $\Delta \leftarrow -\frac{1 + d_i^2 - d_{i-1}^2}{2 * \lambda}$
end if
return $(C_i$-1$)/\lambda$+1/(2*$\lambda$)+$\Delta$
End
```

3.6 Alert Correlation Process

After acquiring the proper accuracy in the probability calculation process, it is possible to give a stream of alerts to our correlator to process it. Input alerts first go through the feature vector generator one by one, then the correlation probability for each vecotr is calculated by using fuzzy matcher. If the matching value is less than rs (rule selection threshold) then the AIRS algorithm and memory cells are used for probability calculation. Each alert probably is correlated with few previous alerts and is added to a structure called hyper-alert. Each hyper-alert contains alerts with some degree of correlation that could be placed in a possible attack scenario. When a new alert such as a_i arrives, its correlation with all previous alerts in existing hyper-alerts is calculated, and the hyper-alert such as H_j that contains the maximum probability of correlation (C_{max}) with a_i is identified. If the probability is higher than a minimum predefined threshold, then a_i is added to H_j. Otherwise, a new hyper-alert with only one alert (a_i) is created. If H_j exists, then all alerts in H_j are checked, and each alert with the correlation probability near to the (C_{max}) is correlated with a_i. The value of nearness can be defined in the system parameters as the correlation sensitivity. In this process each time an alert is correlated with another one, the ACM, Π^b and Π^f matrices are also updated. As a result, the system changes its acquired knowledge dynamically and adapts itself incrementally with the new correlation results. The result of this process is the hyper-alerts. Each hyper-alert contains a number of alerts and is considered as a possible attack scenario.

3.7 Attack Graph Generation

Hyper-alert is a valuable means for presenting alerts' relationships. However, by considering the number of generated alerts in a real system, it can be concluded that the size of the hyper-alerts increase very quickly, and it becomes very difficult to extract the required information from it. Each hyper-alert contains the step by step activities of an attacker. But, an attack graph is a directed graph that shows the overall scenario of an attack, and it contains one node for each alert type. By using the attack graph, it is possible to have an overall and concise view of the attack scenario. As mentioned before, Π^f matrix is generated during the correlation process, and it is used in attack graph generation.

The algorithm 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 [33].

4 Evaluation and Results

The alerts produced by Realsecure [25] on the DARPA2000 [9] dataset are employed to evaluate the accuracy of the system in the extracting two attack scenarios LLDoS1 and LLDoS2. The alerts produced by Snort on the netForensics honeynet data [17] are also employed to evaluate the performance of the system. We use 31 general rules in our rule set and use their corresponding 31 training antigens for AIRS training part. Before system starting its work, the AIRS algorithm is executed and the result memory cells are stored in the system. Therefore the initial knowledge of the system consists of the fuzzy rules and memory cells. System uses this initial knowledge to correlate the input alert stream. To evaluate the accuracy of our system, we use three measures, completeness, soundness and false error rate.

Completeness is defined as the ratio of the correctly correlated alerts to the related alert for a scenario. Soundness is defined as the ratio of the correctly correlated alerts to the total correlated alerts for a scenario and false alert rate is defined as the ratio of the incorrectly correlated alerts to the related alerts for a scenario.

There are so many parameters in the system. We used many different values for each parameter in order to test the system. After finding the best value of parameters in the system we execute our system for two mentioned scenarios (LLDoS1 and LLDoS2). Each scenario is examined 10 times for each setting and the reported results are based on the average values. We change the rule selection threshold (rs) from 0 to 1 to investigate the effect of each classifier (fuzzy and AIS-based) in the accuracy of the system. The value of 0 for rs means that the system is working only with fuzzy rules and is not relies on the AIS-based correlator, and the value of 1 for rs means that the system completely uses AIS-based correlator. By changing the value of the rs system can work from a pure fuzzy correlator to a combinatory fuzzy and AIS-based correlator and finally to a pure AIS-based correlator. Our goal here is to use as less as possible initial knowledge and gain the best accuracy. As results we use only 31 initial rules in our rule collection. Although the pure fuzzy correlator (rs=0) may work with higher number of rules it is not possible to define all situations of two alerts for correlation and to declare their correlation probability. With our limited number of rules the results of the pure fuzzy correlator is weak and we do not report them here. As the result we increase the value of rs and investigate the results. By increasing the value of rs the accuracy of the system is increased until it reach near 0.9. For rs=0.9 the system uses fuzzy rules if the matching value of a rule is more than or equal to 0.9. It is reasonable to use a rule with matching value more than 0.9, beacuse it is accurate enough and there is no need to learn anything to be able to classify this data. By increasing the value of rs from this point (0.9) we neglect the existence of matched rules and this cause to increase the execution time and to decrease the accuracy of the results. The best results are obtained by rs=0.9 to 1 in different datasets. We report the results of rs=0.9 as fuzzy AIS-based and rs=1 as pure AIS-based correlator. We also change the value of the lymphocyte number for both scenarios and for both pure AIS-based and fuzzy AIS-based correlator from 100 to 2000 and report the results.

To evaluate the performance of the system two types of experiments are done. We change the number of lymphocyte from 100 to 2000 for both rs=1 and rs=0.9 and compare the execution time of the correlator. We also change the number of alerts in netForensics honeynet data from 1000 to 5000 and investigate the execution time for three values of rs: 0.1, 0.9 and 1.0.

4.1 Experiments with LLDoS1.0

In this experiment the produced alerts by Realsecure for inside1 traffic are used. Realsecure produces 922 alerts from 22 types for this data. The LLDoS1.0 is a five stages attack. It consists of the following stages:

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

We examine this data with rs=1 and rs=0.9 (pure AIS correlator and fuzzy AIS correlator). Both correlators are examined ten times for each different number of lymphocytes (from 100 to 2000). Both correlator are able to extract the attack scenario almost completely. The alerts that appear in almost all extracted scenario are *Sadmind_Ping*, *Admind*, *Sadmind_Amslverify_Overflow*, *Rsh* and *Mstream_Zombie*. The last step of the attack is not extracted in every experiment. Its related alert(*Stream_DoS*) is placed in an hyper-alert with only one alert. There are also alert types such as *SSH_Detected*, *TelnetEnvAll*, *TelnetXdisplay* and *TelnetTerminaltype* that apear in some runs. We consider all these alert types as false correlation. Figure 3 shows one sample extracted scenario by fuzzy AIS-based correlator (rs=0.9).

The differences of the results for two values of rs is in the number of required lymphocytes to get the best results. Although the fuzzy AIS-based correlator (rs=0.9) is able to extract the scenario with less number of lymphocytes (even 100) the pure AIS-based correlator (rs=1) do the same with more number of lymphocytes (about 2000). Table 4 shows the comparison of the completeness, soundness and false error rate for two correlators with the same parameters. It shows that although the soundness and false error rate for two correlators are very close the compleness of the fuzzy-AIS-based is better than pure AIS-based. Choosing the value of rs=0.9 means that the system first tries to find a fuzzy rule with the matching value of 0.9 or more with the input alerts and if it cannot find such rule it uses AIS-based correlator. The results show that there are some evidences of attack in inside1 dataset that is extractable with our general fuzzy rules. But AIS-based correlator needs to train with more number of lymphocytes to be able to extract these evidences. Figure 4 shows the completeness of both correlators with different number of lymphocytes for LLDoS1.0. We consider the number of related alerts for scenario of LLDoS1.0 58.

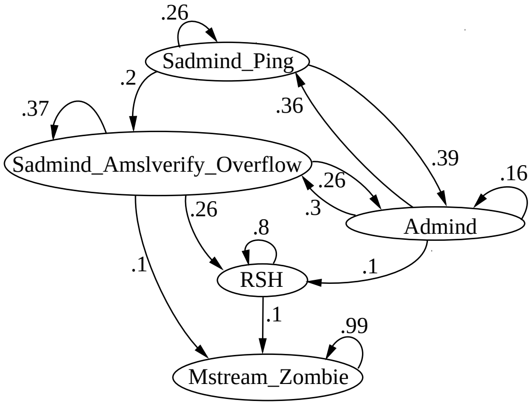

Figure 3: The attack graph generated for LLDoS1.0 (rs=0.9)

4.2 Experiments with LLDoS2.0

In this experiment the produced alerts by Realsecure for inside2 dataset are used. Realsecure produces 494 alerts from 20 different types for this data. The LLDoS2.0 is also a five stages attack. It consists of the following stages:

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

We examine this data with rs=1 and rs=0.9 (pure AIS correlator and fuzzy AIS correlator). Both correlators are examined ten times for each different number of lymphocytes (from 100 to 2000). Both correlators are able to extract the attack scenario almost completely. The alerts appear in almost all extracted scenario are Admind, Sadmind_Amslverify_Overflow, FTP_Put and Mstream_Zombie. The last step of the attack is not extracted in every experiment. Its related alert (Stream_DoS) is placed in an hyper-alert with only one alert. There are also alert types such as *FTP_User*, *FTP_Pass*, *FTP_Syst*, *TelnetEnvAll*, *TelnetXdisplay* and *TelnetTerminaltype* that apear in a few runs. We consider all these alert types as false correlation. Figure 5 shows one sample extracted scenario by fuzzy AIS-based correlator (rs=0.9).

Table 4: Accuracy comparison of pure AIS-based and Fuzzy AIS-based method for LLDoS1.0

	rs=1 (Pure AIS)			**rs=0.9 (Fuzzy-AIS)**		
	Completeness	**Soundness**	**False Alert**	**Completeness**	**Soundness**	**False Alert**
Mean	.720	.977	.022	.957	.948	.053
Std. Dev.	.097	.017	.017	.013	.007	.008

Table 5: Accuracy comparison of pure AIS-based and Fuzzy AIS-based method for LLDoS2.0

	rs=1 (Pure AIS)			**rs=0.9 (Fuzzy-AIS)**		
	Completeness	**Soundness**	**False Alert**	**Completeness**	**Soundness**	**False Alert**
Mean	.750	.792	.296	.745	.82	.245
Std. Dev.	.056	.082	.131	.053	.082	.119

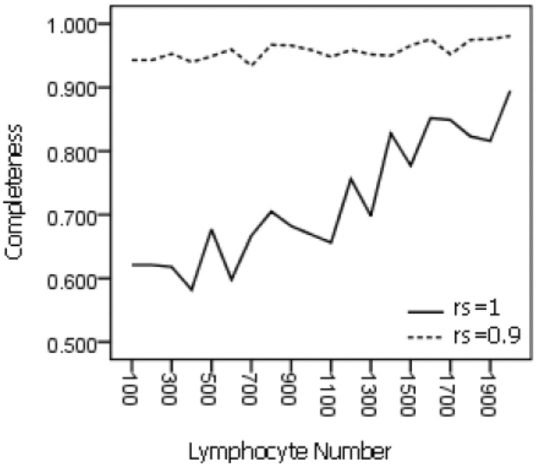

Figure 4: Comparing the completeness of the correlator for rs=1 and rs=0.9 with different number of lymphocyte for Inside1 traffic

Here two correlators are working almost in the same way and they work with different number of lymphocytes almost the same. The advantages of fuzzy AIS-based correlator (rs=0.9) are its better execution time and its better average soundness and average false alert rate than the pure AIS-based correlator (rs=1.0). Table 5 shows the comparison of the completeness, soundness and false error rate for two correlators with the same parameters. The average false alert rate for both correlators is relatively high the reason is that we do not consider the telnet alerts as related alerts in this scenario. By considering the telnet alerts as related alerts the mean false alert rate decreases to 0.1 with standard deviation of 0.05 for both correlator. The results show that the accuracy of our system is comparable with some more complex correlators without the need to complex rules definition task. Figure 6 shows the soundness and false alert rate for both correlators with different number of lymphocytes for LLDoS2.0.

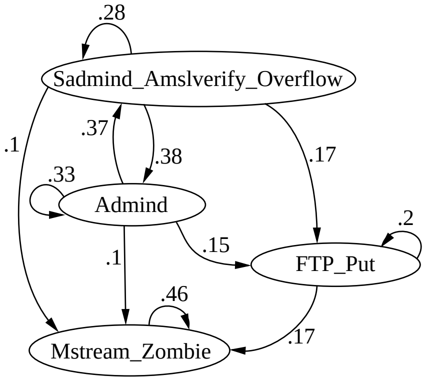

Figure 5: The attack graph generated for LLDoS2.0 (rs=0.9)

4.3 Experiments with netForensics Honeynet Dataset

The netForensics honeynet dataset contains 35 days of traffic logs collected from February 25, 2005 to March 31, 2005 [17]. 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 two first days of netForensics honeynet data is employed to test the ability of our system for extracting the attack scenarios.

Snort generates 3419 alerts belonging to 43 different alert types for these two days. Results show that all 43 types of alerts in the input data are correlated with

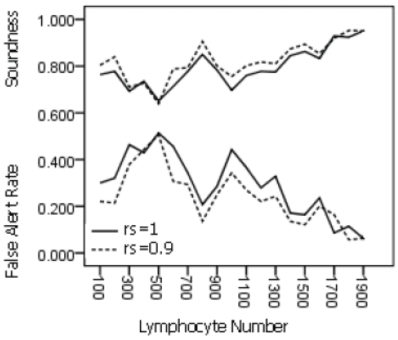

Figure 6: Comparing soundness and false alert rate for rs=1 and rs=0.9 with different number of lymphocyte for inside2 traffic

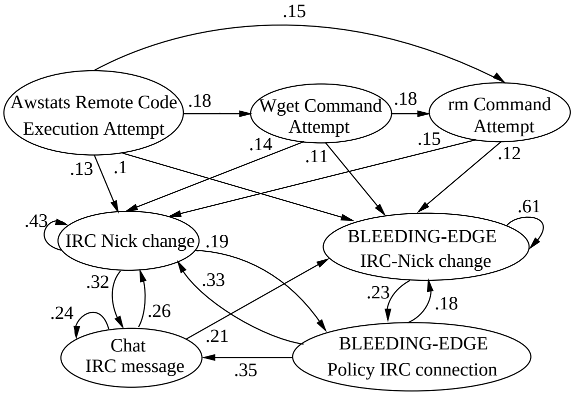

Figure 7: The attack graph generated for netForensics honeynet (rs=0.9)

each other with different strengths. The ACM, Π^b and Π^f matrices are created and the correlation information are stored in them and are used for hyper-alert generation and attack graph generation. 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 types of alerts: *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*, *BLEEDING-EDGE POLICY IRC connection* and *CHAT IRC message* for the rest of the attack, and our system correlates these alerts with alerts of the first step of the attack scenario. Figure 7 shows the extracted attack scenario for rule selection value of 0.9.

4.4 Performance Analysis

To evaluate the performance of our correlator, we considere its execution time with different number of lymphocytes and with different values of rs. Figure 8 shows the execution time with different number of lymphocytes for LLDoS2.0. As it is expected the execution time has a direct relation with the value of rs. For rs=0.1 the time is the least and for rs=1 the time is the most. The reason is that less value for rs means the more selection of fuzzy rule matcher and bypassing the AIS-based correlator. With increasing the value of rs the possibility of finding a rule is decreased and the system uses AIRS for more times and as result the execution time is increased. Moreover, as the number of lymphocytes increases the execution time is increased very slowly. For example in Figure 8 when the number of lymphocytes increases from 100 to 2000 the execution time increases from 5 to 10 seconds for rs=1. It means that 20 times increment in lymphocyte number creates an increasing of two times for execution time. Then the effect of the number of lymphocytes is low.

Figure 9 shows the effect of the number of alerts in the execution time for netForensics honeynet data. The number of alerts is increased from 1000 to 5000 and the execution time of the system is measured for different values of rs. Results show that, the execution time of the algorithm is $O(n^2)$ (n is the number of alerts). For example for 1000 alerts and rs=0.9 the execution time of the system is 19 seconds and with 2000 alerts the execution time is 76 seconds that is four times increasing. It seems that the execution time of the correlation algorithm with higher number of alerts is not acceptable. In this version of the system we try to show the ability of our correlation engine to correlate alerts accurately but, for using our correlator in online mode it is better to work more on the correlation process to improve its performance. One possible improvement is to define a time window for correlation. Time window can considerably decrease the execution time of the system. It is not necessary to correlate each alert with all previous alerts. It is sufficient to correlate it with a limited number of alerts during a time window. In a real environment, it is possible to adjust the execution time of the algorithm by an acceptable length of time window.

5 Conclusion and Future Work

We use AIS and fuzzy logic as two soft computing techniques for alert correlation. Our proposed system needs only a few general rules about the relation between seven selected features. These rules are defined by some linguistic terms such as *low*, *high* and *medium* to simplify the rule definition and make it more general. Also, we need these rules as input to AIRS algorithm. AIRS is a supervised learning algorithm and by using our initial rules, it extracts more information for correlation of previous unseen patterns and stores them in the form of memory

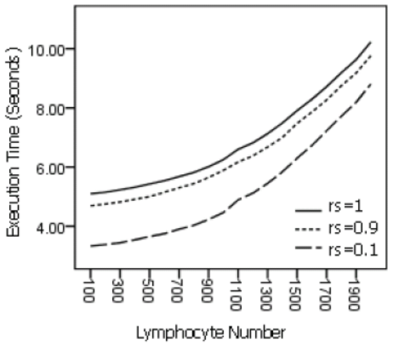

Figure 8: Execution time of the correlator for rs=0.1, rs=0.9 and rs=1 with different number of lymphocyte for inside2 traffic

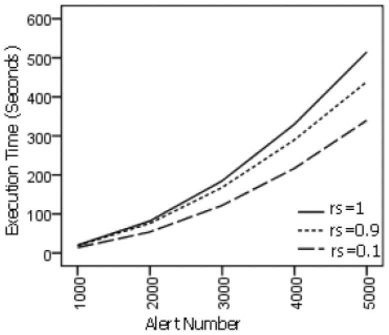

Figure 9: Execution time of the correlator for rs=0.1, rs=0.9 and rs=1 with different number of alerts for netForensics honeynet data

cells. During the correlation process we use a simple fuzzy rule matcher to find the proper rule for each feature vector. It needs that the matching value become more than a threshold (rs). If it is not, then it uses the memory cells produced by the AIRS algorithm to find the correlation probability. In this way, we use both the simplicity and speed of fuzzy rules and the learning ability of AIRS to correlate input alerts. Our system is examined by two traffic data of DARPA2000 and netForensics honeynet data and its ability to extract the attack scenario is proven. Our system is simple to run, it needs no complicated initial data. It can learn and remember the correlation between different attack types. The rs parameter is an important parameter and makes our system more flexible. It is used to balance between the static nature of predefined fuzzy rules and dynamic nature of AIS-based learning system. We get the best results for both datasets with the value of 0.9 for rs. But it is possible to define more number of rules and decrease the rs value. The execution time of the system is increased gradually with the number of lymphocytes. It is more dependent to the number of alerts. The results show that the execution time of the system is in the order of n^2 with the number of alerts but in fact, in a real environment it is not necessary to correlate each alert with all previous alerts. It is sufficient to correlate each alert with a limited number of alerts during a time window. Then it is possible to adjust the execution time of the algorithm by an acceptable time window.

Acknowledgments

This work is supported by the Iranian Research Institute for Information and Communication Technology. The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

[1] M. R. Ahmadi, "An intrusion prediction technique based on co-evolutionary immune system for network security (coco-idp)," *International Journal of Network Security*, vol. 9, no. 3, pp. 290–300, 2009.

[2] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multistep cyber attacks for scenario recognition," vol. 1, pp. 284 – 292, Apr. 2003.

[3] F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," *Security and Privacy, IEEE Symposium on Security and Privacy*, p. 202, 2002.

[4] F. Cuppens and R. Ortalo. "Lambda: A language to model a database for detection of attacks,". in *Recent Advances in Intrusion Detection*, vol. 1907, pp. 197–216. 2000.

[5] O. Dain and R. Cunningham, "Fusing a heterogeneous alert stream into scenarios," in *Proceeding of the 2001 ACM Workshop on Data Mining for Security Applications*, pp. 1–13, September 2001.

[6] S. T. Eckmann, G. Vigna, and R. A. Kemmerer, "Statl: An attack language for state-based intrusion detection," *Journal of Computer Security*, vol. 10, no. 1-2, p. 71, 2002.

[7] A. Ghorbani, W. Lu, and M. Tavallaee, *Network Intrusion Detection and Prevention.* Springer New York, 1^{th} edition, 2010.

[8] M. A. Hall. *Correlation-based Feature Selection for Machine Learning.* PhD thesis, University of Waikato, Hamilton, NewZealand, Apr. 1999.

[9] Laboratory ML. Darpa2000 intrusion detection scenario specific data sets. "Http://www.ll.mit.edu,", Apr. 2012.

[10] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Transactions on Information System Security*, vol. 6, pp. 443–471, Nov. 2003.

[11] S. Lee, B. Chung, H. Kim, Y. Lee, C. Park, and H. Yoon, "Real-time analysis of intrusion detection alerts via correlation," *Comput Secur*, vol. 25, no. 3, pp. 169 – 183, 2006.

[12] G. M. Lois and L. Boggess, "Artificial immune systems for classification : Some issues," in *University of Kent at Canterbury*, pp. 149–153, 2002.

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

[14] 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. Recent Advances in Intrusion Detection Systems.

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

[16] B. Morin, L. Mé, H. Debar, and M. Ducassé, "M2d2: a formal data model for ids alert correlation," in *Proceedings of the 5th international conference on Recent advances in intrusion detection*, pp. 115–137, Berlin, Heidelberg, 2002. Springer-Verlag.

[17] netForensics honeynet team. honeynet traffic logs. "Http://old.honeynet.org/scans/scan34,".

[18] P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 245–254, New York, NY, USA, 2002. ACM.

[19] T. Pietraszek. "Using adaptive alert classification to reduce false positives in intrusion detection,". in *Recent Advances in Intrusion Detection*, vol. LNCS 3224, pp. 102–124. Springer-Verlag, 2004.

[20] P. A. Porras, M. W. Fong, and A. Valdes, "A mission-impact-based approach to infosec alarm correlation," in *Proceedings of the 5th international conference on Recent advances in intrusion detection*, pp. 95–114, Berlin, Heidelberg, 2002. Springer-Verlag.

[21] X. Qin. *A Probabilistic-Based Framework for INFOSEC Alert Correlation*. PhD thesis, Georgia Institute of Technology, August 2005.

[22] H. Ren, N. Stakhanova, and A. Ghorbani. "An online adaptive approach to alert correlation,". in *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. LNCS 6201, pp. 153–172. Springer-Verlag, 2010.

[23] R. Sadoddin and A. A. Ghorbani, "An incremental frequent structure mining framework for real-time alert correlation," *Computer Security*, vol. 28, no. 3-4, pp. 153 – 173, 2009.

[24] S. J. Templeton and K. Levitt, "A requires/provides model for computer attacks," in *Proceedings of the 2000 workshop on New security paradigms*, pp. 31–38, New York, NY, USA, 2000.

[25] Lab NCSUCD. Tiaa: A toolkit for intrusion alert analysis. "Http://discovery.csc.ncsu.edu/software/ correlator/ ver0.4/index.html,".

[26] E. Totel, B. Vivinis, and L. Mé. "A language driven intrusion detection system for event and alert correlation,". in *Security and Protection in Information Processing Systems*, vol. 147 of *IFIP International Federation for Information Processing*, pp. 208–224. Springer Boston, 2004.

[27] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Recent Advances in Intrusion Detection*, pp. 54–68, October 2001.

[28] 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.

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

[30] 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.

[31] A. Watkins. "Airs: A resource limited artificial immune classifier,". Master's thesis, Mississippi State University, 2001.

[32] 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, pp. 291–317, 2004.

[33] B. Zhu and A. Ghorbani, "Alert correlation for extracting attack strategies," *International Journal of Network Security*, vol. 3, no. 3, pp. 244–258, 2006.

Mehdi Bateni received his B. Sc. in Computer Engineering in 1997 from University of Isfahan, Isfahan, Iran and his M. Sc. in Computer Engineering from Ferdowsi University of Mashhad, Mashhad, Iran in 2000. He is currently a Ph.D. student in Department of Computer Engineering at the University of Isfahan.

Ahmad Baraani is an associate professor of computer engineering at the Faculty of Engineering of the University of Isfahan (UI). He got his BS in Statistics and Computing in 1977. He got his MS and PhD degrees in Computer Science from George Washington University in 1979 and University of Wollongong in 1996, respectively. He was Head of the Research Department of the Communication systems and Information Security (CSIS) and Head of the ACM International Collegiate Programming Contest (ACM/ICPC) of University of Isfahan from 2000 until 2008. He has published more than 70 papers and He coauthored three books in Persian and received an award of "the Best e-Commerce Iranian Journal Paper" (2005). Currently, he is teaching PhD and MS courses of Advance Topics in Database, Data Protection, Advance Databases, and Machining Learning. His research interests lie in Databases, Data security, Information Systems, e-Society, e-Commerce, Security in e-Commerce, and Security in e-Society.

Ali Akbar Ghorbani currently serves as Dean of the Faculty of Computer Science. His current research focus is Web Intelligence, Network and Information Security, Complex Adaptive Systems, and Critical Infrastructure Protection. He authored more than 240 reports and research papers in journals and conference proceedings and has edited 8 volumes. He served as General Chair and Program Chair/co-Chair for 8 International Conferences and 10 International Workshops. He is the co-inventor of 3 patents in the area of Web Intelligence and Network Security. He has supervised more than 120 research associates, postdoctoral fellows, and undergraduate and graduate students. Dr. Ghorbani is the founding Director of Information Security Centre of Excellence at UNB. He is also the coordinator of the Privacy, Security and Trust (PST) network and PST annual conferences. Dr. Ghorbani is the co-Editor-In-Chief of Computational Intelligence, an international journal, and associate editor of the International Journal of Information Technology and Web Engineering and the ISC journal of Information Security.