I. J. Computer Network and Information Security, 2013, 11, 9-16
Published Online September 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2013.11.02

Time Window Management for Alert Correlation using Context Information and Classification

Mehdi Bateni
SheikhbahaeeUniversity,Isfahan, Iran
bateni@shbu.ac.ir

Ahmad Baraani
University of Isfahan,Isfahan,Iran
ahmadb@eng.ui.ac.ir

***Abstract*—**Alert correlation is a process that analyzes the alerts produced by one or more intrusion detection systems and provides a more succinct and high-level view of occurring or attempted intrusions. Several alert correlation systems use pairwise alert correlation in which each new alert is checked with a number of previously received alerts to find its possible correlations with them. An alert selection policy defines the way in which this checking is done. There are different alert selection policies such as *select all*, window-based *random* selection and *random directed* selection. The most important drawback of all these policies is their high computational costs. In this paper a new selection policy which is named *Enhanced Random Directed Time Window (ERDTW)* is introduced. It uses a limited time window with a number of sliding time slots, and selects alerts from this time window for checking with current alert. *ERDTW* classifies time slots to *Relevant* and *Irrelevant* slots based on the information gathered during previous correlations. More alerts are selected randomly from relevant slots, and less or no alerts are selected from irrelevant slots. *ERDTW* is evaluated by using DARPA2000 and netforensicshoneynet data. The results are compared with other selection policies. For LLDoS1.0 and LLDoS2.0 execution times are decreased 60 and 50 percent respectively in comparing with *select all* policy. While the completeness, soundness and false correlation rate for *ERDTW* are comparable with other more time consuming policies. For larger datasets like netforensicshoneynet, performance improvement is more considerable while the accuracy is the same.

***Index Terms*—**Alert Correlation, Alert selection policy, Time window management, Classification and regression tree (CART)

I. Introduction

Intrusion detection is the process of identifying and (possibly) responding to malicious activities targeted at computing and network resources[1]. The system that runs this process is named Intrusion Detection System (IDS). When an IDS detects a malicious activity, it generates an alert. The alert is usually in low-level format. It means that the alert contains a little information about the malicious activity and it is almost useless for system administrator. An IDS in a large network of computers with many different users generates high volumes of raw alerts. These alerts overwhelm the system administrator in such a way that she/he cannot use them effectively. As a result, the administrator may ignore alerts and miss their possible related intrusions. Alert correlation is used to overcome this problem.

Alert correlation is a process that analyzes the alerts produced by one or more intrusion detection systems and provides a more succinct and high-level view of occurring or attempted intrusions. It has two main goals: reducing the number of alerts and increasing the relevance and abstraction level of the produced reports[2].

Most alert correlation systems examine each new alert with a number of previous alerts in order to find its correlation with them. We refer to these correlation methods as pairwise correlations. The correlation process is started by arrival of a new alert, A_{last}. The system examines A_{last} with some previous alerts to find its possible causal relation with them. The main question is that with which previous alerts A_{last} should be examined? Alert selection policy describes the way in which this question is answered. It defines the scope and method of the search in previous alerts. The performance of pairwise correlation process is considerably affected by its alert selection policy.

Generally, the correlation process contains an outer loop that repeats for each new alert. The correlation process for each alert contains a couple of activities that are done in each repetition. The alert selection policy determines one important part of these activities. It determines the previous alerts which are checked for correlation with A_{last}. Alert selection policy does not determine the correlation between A_{last} and previous alerts. It only defines the way in which previous alerts are selected and other components determine the degree of correlation.

The first and simplest possible strategy for alert selection is to select all previous alerts. There are many researches[3] that use this strategy. It leads to another

inner loop that repeats for all previous alerts. By increasing the number of alerts and by arriving each new alert, this process takes longer and longer time. It is obvious that this strategy is not appropriate for large dataset and for online correlation. Another group of selection policies use a limited time window to restrict the scope of the search in previous alerts. By using a time window, the number of repetitions for inner loop is controlled by the width of the time window. Therefore the computational cost of the system is adjustable. There are many researches that use time window for alert correlation. Time window management is an important issue in all these researches. In a number of researches, all alerts in the recent time window are used for correlation[4-6]. In some other works only a limited number of alerts in the recent time window are used in correlation process. The previous alerts that are involved in the correlation process of A_{last}, could be selected randomly[7].It is possible to use more smart methods[8]. Generally, these methods gather some useful information during the correlation process and use this information for next alert selections.

In this paper a new alert selection policy, named *Enhanced Random Directed Time Window(ERDTW)* is introduced that is based on the time window. It uses a time window which is divided to a number of time slots. Several alerts are randomly selected from each time slot in order to check their correlation with A_{last}. The number of alerts that are selected from time slot, T_i, is specified by the number of alerts that are occurred during T_i, the time distance of T_i with current slot and the relevancy of T_i which is calculated by considering the context information that is gathered from T_i during the previous correlations.

The main contribution of this paper is the concept of relevancy for time slot T_i. During time slot T_i, some context information is used. Information such as the number of alerts which are occurred in T_i, the most observed source IP address in T_i and the number of its observation, the most observed destination IP address in T_i and the number of its observation, the number of observation of the most observed source IP address in T_i as the most observed destination IP address for slots prior to T_i, the number of dangerous destination ports that are observed in alerts which are occurred in T_i and the mean and standard deviation of the number of alerts in the time window containing T_i. This information is used to classify a time slot T_i to *Relevant* or *Irrelevant* slot. More alerts are selected from relevant (dangerous) slots and less or no alerts are selected from irrelevant (safe) slots.

A training dataset which is generated by gathering the above mentioned information is used. A label of *Relevant* or *Irrelevant* is assigned manually to each data. This dataset is used as the training data to make a decision tree. Classification and Regression Tree (CART) algorithm [9] is used to make the decision tree, and the generated tree is used in our new alert selection policy to classify slots.

DARPA2000[10] and netforensicshoneynet data[11] are used to investigate the performance and accuracy of *ERDTW* for alert correlation and, results are compared with results generated by other alert selection policies. The completeness, soundness and false correlation rate are used for accuracy evaluation, and the execution time is used for performance evaluation. Accuracy measures are very close for all different polices, but the running time is considerably varying for them. For example, the running times for LLDoS1.0 with different selection policies are considerably different. They are 12.53, 7.94, 7.81 and 4.95 Seconds for *select all*, *random*, *random directed (RDTW)* and *enhanced random directed (ERDTW)* respectively. For larger datasets like netforensicshoneynet data, performance improvement is more considerable. The results show that *ERDTW* is as accurate as other selection policies and, it is considerably more effective.

The rest of the paper is organized as follows. Section 2 illustrates different alert selection policies and their features. Section 3 describes the proposed alert selection policy in details. Section 4 reports the result of running the system with the DARPA2000 and netForensicshoneynet data. Finally, Section 5 provides the conclusion.

II. Alert Selection Policy

Different correlation systems use different methods to correlate alerts. For example, similarity measures, pre-defined rules, pre-specified scenarios and statistical measures are used by different correlation systems. In pairwise correlation, each new alert, A_{last}, is examined with several previous alerts and the decision for correlation is made after that. The correlation process contains a loop that repeats for each new alert, A_{last}. Other loop(s) is required to find and select the previous alerts for correlation with A_{last}. Regardless of the methods which are used to calculate the correlation between a pair of alerts, alert selection policy identifies which previous alerts should be selected for correlation calculation. It defines the scope and method of the search in previous alerts. Alert selection policy considerably affects the performance and accuracy of the pairwise correlation. There are many different selection policies. We discuss some well-known policies in this section.

A. Select All

The simplest selection policy for pairwise alert correlation is *select all* policy. It investigates each new alert A_{last} with all alerts that are occurred before it[3]. In this way A_{last} is checked with all previous alerts including the right relevant alerts. Therefore, by selecting all previous alerts the selection policy does not reduce the chance of A_{last} for correct correlation. *Select all* policy seems attractive, but it is not an applicable selection policy for large datasets and for online correlation. By increasing the number of alerts the cost of correlation is increased dramatically. It is obvious that this policy is not applicable to online correlation where, the number of alerts is not limited. The policy is not also appropriate for large datasets, because two nested loops make the correlation process very time consuming. Moreover,

checking a new alert A_{last} with an alert which is occurred for example, 5hours ago is not reasonable. Therefore, window-based alert selection policies are introduced.

B. Random Selection

In this strategy a limited time window which contains several sliding time slots is used [7]. Only the alerts that are occurred during this time window are considered for correlation and the older alerts which are out of time window are ignored[7]. Two influential parameters determine the performance and accuracy of this selection policy. The number of time slots, *n* and the width of each time slot, W_s.

Suppose that 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, m_i alerts are randomly selected for correlation from time slot, T_i. m_i is different for each time slot and is less than or equal to s_i. m_i is determined by considering two values: *i*, the slot number that is between 1 and *n* and s_i, the total number of alerts in T_i. m_i is calculated as below[7]:

$$m_i = \left\lfloor \frac{s_i \times i}{n} \right\rfloor \quad (1)$$

For example, in Fig. 1, the numbers of selected alerts for slot 1 to 5 are 2, 6, 7, 10, and 11 respectively. By using the time window, this selection policy restricts the search scope. It also considers the time distance of the alerts and assigns more importance to more recent time slots. All alerts of the last time slot are selected by this policy, but some alerts are randomly selected from other time slots. As a result, the percentages of alerts that are selected for each time slots 1 to 5 are 17, 35, 54, 77 and 100 percent respectively. It is obvious that by limiting the selection duration and decreasing the number of alerts that are selected from each time slot, considerable performance improvement is obtainable[7]. The main drawback of *randomselection* is that the blindly selection may lead to accuracy degradation. It is probably better to select alerts more wisely to increase the accuracy of the policy.

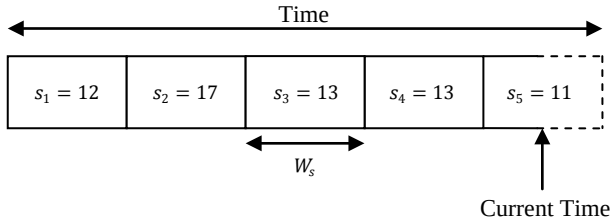

Figure 1. A time window with 5 sliding time slots

C. Random dircted time window (RDTW)

Although *randomselection* provides better performance, its accuracy is under question. Blindly selecting previous alerts may lead to missing some relevant alerts and leads to degrading the accuracy of the correlation process. *Random Directed Time Window(RDTW)*[8] employs a new parameter besides to *n* and W_s for each time slot T_i. This parameter, *mx*, is the maximum correlation between new alert, A_{last}, and all selected alerts from T_i. *RDTW*uses the Equation 1 to calculate the initial value of m_i. After selecting and correlating$\frac{m_i}{2}$ alerts of T_i, the calculated value of *mx* is checked. If *mx* is less than a minimum acceptable correlation threshold, min_{accept}, then it seems that A_{last} is not related with T_i. As a result, m_i is decremented by one, and the selection and correlation process is continued by the new value of m_i. On the other hand, if after correlating m_i alerts *mx* is more than $1\text{-}min_{accept}$, then it is reasonable to conclude that A_{last} is strongly related with T_i. Thus, $m_i$is incremented by one and the system continues the process of alert selection and correlation for this slot (T_i). The selection terminates either by encountering an alert with correlation probability less than $1\text{-}min_{accept}$ or by selecting all alerts from T_i[8]. As a result of this strategy, the selection process is directed 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. The main goal of *RDTW*is to direct the selection process toward the more relevant time slots and to select more alerts from them. Even it is possible to select all alerts of T_i, if it is determined as a relevant slot for A_{last}. For irrelevant slots, the selection process is stopped early, even after selecting$\frac{m_i}{2}$ alerts. Algorithm 1outlines *RDTW*selection policy [8].

Algorithm 1.*RDTW* selection policy

Input: New alert A_{last}
Output: A group of alerts for correlation with A_{last}
1: $n \leftarrow$ The number of time slots
2: **for** $i = 1$ **to** *n***do**
3:$m_i \leftarrow \left\lfloor \frac{s_i \times i}{n} \right\rfloor$//$s_i$is the number of alerts in T_i
4: $k \leftarrow 0$
5: $mx \leftarrow -1$
6: **while** ($k < m_i$) and ($k < si$)
7: $b \leftarrow$ a random alert from slot *i*
8: $y \leftarrow$ Correlation value between $A_{last}$and $A_{selected}$
9: **if** ($y > mx$)
10: $mx \leftarrow y$
11: **end if**
12: $k \leftarrow k + 1$
13: **if** ($k > \frac{m_i}{2}$) and ($mx < min_{accept}$)
14: $m_i \leftarrow m_i - 1$
15: **end if**
16: **if** ($k = m_i$) and ($mx > 1\text{-}min_{accept}$) and ($k < s_i$)
17: $m_i \leftarrow m_i + 1$
18: $mx \leftarrow 0$
19: **end if**
20: **end while**
21: **end for**

III. ENHANCED RANDOM DIRECTED TIME WINDOW (ERDTW)

*RDTW*selects alerts more wisely than *random* alert selection policy almost with the same performance. It selects more than m_i alerts from relevant slots and less than m_i alerts from irrelevant ones (*Random* policy selects m_i alerts from all slots). *ERDTW*uses some context information to classify time slot, T_i, to *Relevant* (dangerous) or *Irrelevant* (safe) slot. It uses Equation 2 to calculate the value of m_i for T_i.

$$m_i = \left\lfloor \frac{s_i \times i}{n} \right\rfloor \times \omega \quad (2)$$

ωis a reduction factor. It is an adjustable parameter that is 1 for *relevant* slots and is a real value between 0 and 1 for *irrelevant* slots. By using ω, it is possible to adjust the computational cost of the *ERDTW*policy more precisely. A small value (less than or equal to 0.5) is assigned to ω for *irrelevant* slots. As a result, the number of selected alerts is reduced considerably. Both performance and accuracy of the system is remarkably related to the process of context information gathering. If the context information is selected properly with low computational cost, then the performance of the system is considerably improved. The accuracy of the system is improved if the context information would be useful and related information for identifying the *Relevant* and *Irrelevant* time slots.

After calculating m_i, the same process as *RDTW*is used to increase and decrease the value of m_i. Two next subsections describe the context information and the classification process.

A. Context information

Context information for each time slot, T_i, is gathered during the processing of the alerts of T_i. This information is stored and used later to classify T_i as *Relevant* or *Irrelevant* time slot. The information gathering process should be a very light weight process in such a way that its cost does not exceed the computational cost that is saved by using ω. Suppose that S_IP_{max} and D_IP_{max} are the most observed source and destination IP addresses in T_i. The following information is used as context information of T_i.

S_IP_{per}: the percentage of alerts that $S_IP_{max}$is observed in them

If there are many alerts with the same source IP address in T_i, then T_i is more likely to be a dangerous slot and should be classified as *Relevant* (dangerous) slot. We assume that there is a direct relation between the S_IP_{per} and the degree of relevancy. S_IP_{per} is calculated by counting the number of observation of S_IP_{max} and dividing it to s_i. S_IP_{per} is a real value between 0 and 100.

D_IPper:the percentage of alerts that D_IPmax is observed in them

If there are many alerts with the same destination IP address in T_i, then T_i is more likely to be a dangerous slot and should be classified as *Relevant* slot. We assume that there is a direct relation between the D_IP_{per} and the degree of relevancy. D_IP_{per} is calculated by counting the number of observation of D_IP_{max} and dividing it to s_i. D_IP_{per} is a real value between 0 and 100.

Seqnum:the number of observation of S_IPmax of Ti in the set of D_IPmax of Tj (for all j less than i)

If $D_IP_{max}$of one previous time slot such as T_j is equal to S_IP_{max} of T_i, then it is possible that T_j contains the attack steps that prepare for attacks in T_i. We assume that the number of time slots like $T_j$has direct relation with the degree of relevancy of T_i. $Seq_{num}$is an integer value between 0 and (*n-1*).

D_Portper: the percentage of alerts with destination ports belonged to the dangerous port numbers

By considering different known attacks and the experience of administrator about them, it is possible to define a list of dangerous ports. Therefore, D_Port_{per} is calculated by counting alerts with the destination ports that are belonged to this list and dividing it to s_i.

InRange: si is in range or is not

It is a Boolean value which is used to identify that the value of $s_i$is close or it is far from the mean value of alert numbers in the time window and is calculated as follows.

$$InRange = \begin{cases} True & if\ Lower \le s_i \le Uper \\ False & otherwise \end{cases} \quad (3)$$

Where *Uper* and *Lower* are calculated as follow:

$$Uper = \begin{cases} Mean + StDev & if\ Uper < Max \\ Max & otherwise \end{cases}$$

$$Lower = \begin{cases} Mean - StDev & if\ Lower > Min \\ Min & otherwise \end{cases}$$

Where, *Max*, *Min*, *Mean* and *StDev* are the maximum, minimum, mean and standard deviation of s_j respectively (for j=1to n). We assume that if the value of *InRange* is false, then it is more likely to classify the T_i as *Relevant* or dangerous slot.

Window management for *ERDTW*is based on the sliding window. After a specified time (W_s), all time slots slide forward and one time slot is put out from one side and a new time slot is entered from the other side. Accordingly some information about time slots is modified. S_IP_{per},D_IP_{per} and D_Port_{per} are not modified by slot sliding, but $Seq_{num}$and *InRange* are probably changed by sliding slots. As a result, only when a new slot is started the context information about previous slots is updated. The process of information gathering for current slot is done during the correlation process. Hence, the computational cost of this process is crucial.

B. Classification

Considering attacks which exist in our three experimental scenarios, a dataset of aforementioned context information is generated and *Relevant* (dangerous) or *Irrelevant* (safe) labels are assigned to its record manually. Each record contains context information about a sample slot and its *Relevant* or *Irrelevant* label. This dataset is used to train the classification algorithm. The Classification and Regression Tree (*CART*) algorithm[9] with ten-fold cross-validation is used to generate a proper decision tree based on the training data. The total number of training data is 160 samples. They are generated by considering different possible values of five above context information, and their labels are assigned manually. 120 out of 160 samples are from Irrelevant and 40 samples are from *Relevant* class. The *CART* algorithm classifies 136 samples correctly and 24 samples incorrectly. Thus, its accuracy is 85 percents. The confusion matrix is as follows:

$$\begin{array}{lcc} \text{Classified as}\rightarrow & \textit{Relevant} & \textit{Irrelevant} \\ \text{Actual Class}\downarrow & & \\ \textit{Relevant} & \begin{bmatrix}25 & 15\\ 9 & 111\end{bmatrix} & \end{array}$$

The goal of this classification is not to decide about the correlation of two alerts. It only tries to recognize the more dangerous time slots and, even if its decision would be incorrect the processing of the slot will be continued only with less number of selections and will be continued by our directed strategy. Hence, 85 percent of accuracy is acceptable for this application.Fig. 2 shows a sample generated tree by the *CART* algorithm.

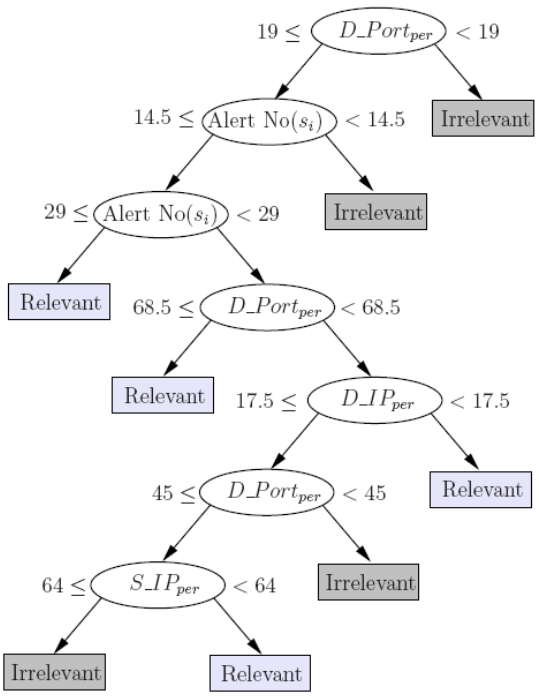

Figure 2. A sample generated tree by the CART algorithm for classifying slots to *Relevant* and *Irrelevant*

This tree is used in *ERDTW*to determine whether a slot is dangerous (*Relevant*) or safe (*Irrelevant*). By determining the relevancy of one slot, the amount of processing on its alerts is determined. *ERDTW*uses ω to enforce the impact of relevancy on each slot. For relevant slots, ω is considered 1. As a result, more alerts are selected from them, and for irrelevant slots, ω is considered a value between 0 and 1. Therefore, less processing efforts are imposed for them.

Algorithm 2 outlines the *ERDTW*selection policy. The main difference between this algorithm and algorithm 1 is in line 6. Where, the algorithm identifies the safety of time slot T_i. It is accomplished by considering the context information that is previously generated for each time slot. If T_i is not safe then it needs more processing efforts. It is achieved by assigning 1.0 to ω (line 8). Otherwise, ω gets the value of 0.5 (line 10), and it leads to less processing for slot T_i. The rest of the algorithm is similar to Algorithm 1. It uses random directed selection with time window to direct its selection toward the more relevant time slots.

Algorithm 2: *ERDTW* selection policy

Input: New alert A_{last}
Output: A group of alerts for correlation with A_{last}
1: $n \leftarrow$The number of time slots
2: **if** (new slot is started with alert A_{last})
3: Update (*ContextInfo*)
4: **end if**
5: **for** i= 1 **to** n**do**
6: $r \leftarrow$Classify(T_i, *ContextInfo*) // $T_i$is i^{th}time slot
7: **if** (r = *Relevant*)
8: $\omega \leftarrow 1$
9: **else**
10: $\omega \leftarrow 0.5$ // or other proper value
11: **end if**
12: $m_i \leftarrow \left\lfloor \frac{s_i \times i}{n} \right\rfloor \times \omega$// $s_i$is the number of alerts in T_i
13: $k \leftarrow 0$
14: $mx \leftarrow -1$
15: **while** ($k < m_i$) and ($k < s_i$)
16: $b \leftarrow$a random alert from T_i
17: $y \leftarrow$Correlation value between $A_{last}$and $A_{selected}$
18: **if** $y > mx$ **then**
19: $mx \leftarrow y$
20: **end if**
21: $k \leftarrow k + 1$
22: **if** ($k > \frac{m_i}{2}$) and ($mx < min_{accept}$)
23: $m_i \leftarrow m_i - 1$
24: **end if**
25: **if** ($k = m_i$) and ($mx > 1\text{-}min_{accept}$) and ($k < s_i$)
26: $m_i \leftarrow m_i + 1$
27: $mx \leftarrow 0$
28: **end if**
29: **end while**
30: **end for**

C. Performance estimation

The numbers of pairs which are selected for correlation by different selection policies are used to estimate their performance. It is a good measure, because the actions that are done for each pair of alerts are the same for different selection policies and are determined by the method of correlation calculation. Suppose that *m* is the total number of alerts, *n* is the number of time slots, *s* is the mean number of alerts in different time slots, *k* is the mean number of relevant slots in a time window, ω is the reduction factor and T_s is the total number of selected alerts by one policy.

For *select all* policy, each new alert is correlated with all previous alerts. Thus, for this policy $T_s$is $\frac{m\times(m-1)}{2}$. On the other hand, for *Random* selection policy, each new alert is correlated with alerts which are belonging to *n-1* previous time slots. Not all alerts from previous time slots are selected. In average *s* alerts are selected from each previous time slot. Therefore, $T_s$for *random*policy is $\frac{m\times s\times(n-1)}{2}$

It is obvious that if the correlation duration is more than the width of time window, then *(m-1)* is greater than *s×(n-1)*. For large dataset the difference between *(m-1)* and *s×(n-1)* is huge. As a result, running times for all window-based polices are considerably less than *select all* policy.

We assume that T_s is also $\frac{m\times s\times(n-1)}{2}$ for *RDTW*policy. It is not far from truth. Results show that the running time for *RDTW*is quite close to *Random* selection. For *RDTW*some slots are processed more than their counterpart in *random*selection and some slots are processed less. In average the value of T_s is very close to its counterpart in *random*selection.

For *ERDTW*two new parameters are introduced: the reduction factor ω and the mean numbers of relevant slots in a time window *k*. The total reduction in the number of selected alerts comparing with previous window based policies is $\frac{(k+(n-k)\times\omega)}{n}$. Thus, for this policy T_s is $\frac{(m\times s\times(n-1))}{2}\times\frac{(k+(n-k)\times\omega)}{n}$. It is obvious that the value of T_s is not more than its counterparts in *random*and *RDTW*policies. For example if half of slots are relevant slots $(k=\frac{n}{2})$and the reduction factor ω is 0.5, then T_s would be 0.75 of its counterparts in two other window-based policies. As a result, the computational cost of *ERDTW*is less than all other selection policies. In next section these theoretic estimations are evaluated by using several datasets.

IV. Evaluation and Results

Alert selection policy is one component of alert correlation process, particularly for pairwise correlation. In order to evaluate *ERDTW*and compare it with other selection policies, a pairwise alert correlation system is required. Regardless of the way in which the correlation between two alerts is calculated, *ERDTW*is able to work as selection policy. We evaluate *ERDTW*and other selection policies in a pairwise alert correlation system named, iCorrelator[8]. It is an AIS-inspired[12] alert correlation system in which a three-layer alert correlation architecture is used. It is able to assign a correlation probability to each pair of alerts and uses this probability to extract the attack scenario. We use different selection policies along with iCorrelator and report the results.

Datasets which are used to evaluate *ERDTW*and other policies are DARPA2000[10] and netForensicshoneynet data[11]. The first dataset is a well known dataset for alert correlation and, there are different works that use it as their evaluation dataset. The second dataset is a relatively large dataset and, it is more appropriate for performance evaluation. Completeness, soundness, false correlation rate (*FCR*) and execution time are reported for both datasets. The completeness, soundness and false correlation rate are defined as follow[13]:

$$Completeness = \frac{\text{correctly correlated alerts}}{\text{related alerts}} \quad (4)$$

$$Soundness = \frac{\text{correctly correlated alerts}}{\text{total correlated alerts}} \quad (5)$$

$$FCR = \frac{\text{incorrectly correlated alerts}}{\text{related alerts}} \quad (6)$$

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

DARPA2000 contains two multi-step attack scenarios LLDoS1.0 and LLDoS2.0. These scenarios are placed in two traffic data file Inside1 and Inside2 and, we examine both traffic data with different selection policies. All datasets are examined ten times with each selection policies and results are reported based on the mean value. The most important parameters that are related to different selection policies are the number of time slots, *n*,the width of each time slot, W_s, the minimum acceptable correlation, min_{accept} and the reduction factor, ω. The values of these parameters are 20, 300, 0.75 and 0.5 respectively.

Alerts in the first attack scenario (LLDoS1.0) are from six different types: *SadmindPing*, *SadmindAmslverifyOverflow*, *Admind*, *Rsh*, *MstreamZombie* and *StreamDoS*. The first five alert types are appeared in extracted scenarios by all selection 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 1 shows generated results for Inside1 data. It shows that the completeness of the *select all* policy is the best among all policies. It is predictable that by using the time window some steps of attacks are ignored, but the main problem with *select all* policy is its performance. The running time for*select all* policy is 12.53 and for *random*, *RDTW*and *ERDTW*are 7.94, 7.81 and 4.95 seconds respectively. Although, the soundness and false correlation rate for different policies are very close the

generated results for *ERDTW*are the best. Therefore, by employing the context information and the classification tree the running time, soundness and false correlation rate are improved. The completeness is decreased comparing with *select all*, but it is improved comparing with two other policies. It shows the advantage of *ERDTW*and our classification method.

Different selection policies are also examined by using Inside2 data. Again all policies extract the attack scenario almost completely (except the last step). Alerts that appear in all extracted scenario are *Admind, SadmindAmslverify Overflow, FTP Put and Mstream Zombie*. The last step of the attack is not extracted in all experiments, and its related alert (*Stream DoS*) is placed in a hyper-alert with only one alert.

Table 1. The results generated for LLDoS1.0 with different policies

	All	Random	RDTW	ERDTW
Completeness	0.941	0.745	0.816	0.878
Soundness	0.950	0.928	0.943	0.953
FCR	0.950	0.060	0.052	0.045

Results generated by different selection policies are more close to each other for Inside2 (see Table 2). Its reason is that the duration of LLDoS2.0 is less than LLDoS1.0, and the size of the time windows that is used for both data are the same ($n \times W_s = 6000$ seconds). Hence, the time window contains almost all alerts of Inside2. Little differences in completeness for four polices are negligible. The soundness and false correlation rate for three time window-based policy are better than *select all*. It shows that these methods select alerts more wisely and less numbers of irrelevant alerts are selected by them. As expected, the execution time improved by using the time window-based polices. The best execution time belongs to *ERDTW*policy without any meaningful accuracy degradation. Thus, our new selection policy is the best method among all policies for Inside2 data.

Table 2. The results generated for LLDoS2.0 with different policies

	All	Random	RDTW	ERDTW
Completeness	0.607	0.600	0.600	0.600
Soundness	0.937	0.982	0.982	0.958
FCR	0.050	0.014	0.014	0.029

Snort generates 3419 alerts belonging to 43 different alert types for the first two days of netForensicshoneynet data. Results show that all 43 types of alerts in the input data are correlated with each other with different strengths. The most compelling evidence of compromise in this data is the outbound IRC communication, which implies that the intrusion succeeded. For this dataset there is not a unanimous agreement about the designated scenario and its related alerts. As a result, we report the extracted attack graph instead of three accuracy measures (completeness, soundness and false correlation rate). 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 all policies correlate these alerts. Fig.3 shows extracted scenarios with two different selection policies: *select all* and *ERDTW*. The first policy has the most computational cost and examines all alerts and, the second one has the least computational cost and examines the least number of alerts. Although the running time is very different for two policies (213.6 and 13.33 seconds), both policies extract the same scenario (see Fig.3). Probabilities that are assigned to edges are a little different, but their general logic is the same.

The goal of a selection policy is to improve the performance of a correlation system without accuracy degradation. Reported results in Table 3 and Fig.3 show that *ERDTW*meets this goal perfectly.

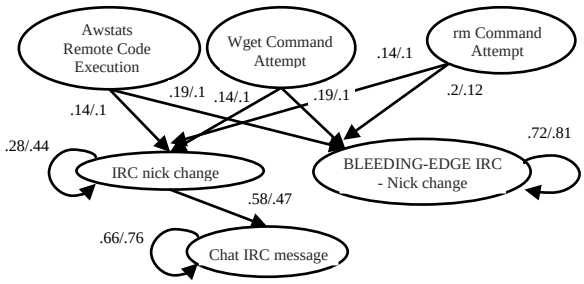

Figure 3. The attack graphs generated for netForensicshoneynet with *Select All* and *ERDTW* (separated by slash)

Table 3. The running time for different policies

	All	Random	RDTW	ERDTW
LLDos1.0	12.53 s	7.94 s	7.81 s	4.95 s
LLDos2.0	3.27 s	2.80 s	2.85 s	1.64 s
netForensics	213.60s	23.98 s	23.11 s	13.33 s

V. Concludsion

In pairwise alert correlation, each new alert is examined with several previous alerts to find possible correlation. Alert selection policy defines the scope and method of the search in previous alerts for selecting some of them. The performance of the correlation is considerably affected by its alert selection policy. A good policy must select the more relevant previous alerts and ignore the irrelevant ones. As a result of this good selection policy, the computational cost is decreased without accuracy degradation. A new alert selection policy named, *Enhanced Random Directed Time Window* (*ERDTW*) is introduced in this paper. It is a time window-based alert selection. The time window contains several sliding time slots. It gathers some context information about each time slot and uses this information later to recognize the importance of the slot. A dataset of training data is generated and used by the Classification and Regression Tree algorithm to classify slots to *Relevant*

and *Irrelevant*. More alerts are selected from *Relevant*slots and less or no alerts are selected from *Irrelevant* slots. Experimental results on different datasets with different alert selection policies show that *ERDTW*is able to achieve two goals simultaneously: the performance and the accuracy.

By using *ERDTW*for LLDoS1.0 and LLDoS2.0, running times are decreased about 60 and 50 percent respectively in comparing with *select all* policy. While reductions for *random*and *RDTW*are 36 and 37 percent for LLDoS1.0 and 14 and 13 percent for LLDoS2.0. The completeness, soundness and false correlation rate for *ERDTW*are comparable with other more time consuming policies. For larger dataset like netforensicshoneynet, the performance improvement is more considerable. Running times for *select all*, *random*, *RDTW*and *ERDTW*are 213.6, 23.98, 23.11 and 13.33 Seconds respectively, while the accuracy is almost the same for all of them.

REFERENCES

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

[2] F. Valeur, G. Vigna, C.Kruegel and R. Kemmerer. A comprehensive approach to intrusion detection alert correlation. IEEE Transactions on Dependable and Secure Computing,2004.p.153-172.

[3] B. Zhu and A. Ghorbani. Alert correlation for extracting attack strategies.International Journal of Network Security, 2006. 3(3):p.244-258.

[4] P. Kabiri and A. Ghorbani. A rule-based temporal alert correlation system. International Journal of Network Security, 2007. 5(1):p.66-72.

[5] Z. Li, A. Zhang, J. Lei and L. Wang. Real-Time Correlation of Network Security Alerts. In proceeding of e-Business Engineering, ICEBE 2007, IEEE International Conference, p.73-80.

[6] H. Ren, N. Stakhanova and A. Ghorbani. An online adaptive approach to alert correlation. Volume 6201 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, p.153-172.

[7] H. Ahmadinejad and S.Jalili. Alert Correlation Using Correlation Probability Estimation and Time Windows. In proceedings of the 2009 International Conference on Computer Technology and Development, IEEE Computer Society ICCTD '09,2009. p.170-175.

[8] M. Bateni, A. Baraani, A. Ghorbani and A.Rezaei. An AIS-inspired Architecture for Alert Correlation. International Journal of innovative Computing, Information & Control, 2013. 9(1):p. 231-255.

[9] Y. Yohannes andJ.Hoddinott. Classification and Regression Trees: an introduction. Technical guide, International Food Policy Research Institute (IIFPRI), 1999.

[10] MIT Lincoln Laboratory. Darpa2000 intrusion detection scenario specific data sets. http://www.ll.mit.edu. (last accessed June 2013)

[11] netForensicsHoneynet team.Honeynet traffic logs. http://old.honeynet.org/scans/scan34. (last accessed June 2013)

[12] L.N. de Castro and J.Timmis. Artificial Immune Systems: A new computational intelligence approach. Springer-Verlag London Berlin Heidelberg, 2002.

[13] P. Ning, Y. Cui and D.S. Reeves. Techniques and Tools for Analyzing Intrusion Alerts. ACM Transactions on Information and System Security, 2004. 7(2):p.274–318.

Mehdi Bateni,is an assistant professor of computer engineering at the Faculty of Engineering of the Sheikhbahaee University (SHBU). He 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 received his Ph.D. in Computer Engineering in 2012 from University of Isfahan, Isfahan, Iran.

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). 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".

