Case-Oriented Alert Correlation

JIDONG LONG and DANIEL G. SCHWARTZ
Department of Computer Science
Florida State University
Tallahassee, Florida 32303
U.S.A.
jidolong@cs.fsu.edu, schwartz@cs.fsu.edu

Abstract: - Correlating alerts is of importance for identifying complex attacks and discarding false alerts. Most popular alert correlation approaches employ some well-defined knowledge to uncover the connections among alerts. However, acquiring, representing and justifying such knowledge has turned out to be a nontrivial task. In this paper, we propose a novel method to work around these difficulties by using case-based reasoning (CBR). In our application, a case, constructed from training data, serves as an example of correlated alerts. It consists of a pattern of alerts caused by an attack and the identity of the attack. The runtime alert stream is then compared with each case, to see if any subset of the runtime alerts are similar to the pattern in the case. The process is reduced to a matching problem. Two kinds of matching methods were explored. The latter is much more efficient than the former. Our experiments with the DARPA Grand Challenge Problem attack simulator have shown that both produce almost the same results and that case-oriented alert correlation is effective in detecting intrusions.

Key-Words: - Alert Correlation, Case-Based Reasoning, Data Mining, Intrusion Detection

1 Introduction

Intrusion detection has been studied for more than 20 years. Although many intrusion detection techniques have been explored, all apply essentially the same methodology, namely, looking for patterns from a single data source. Typical such sources are network traffic, the sequence of system calls on a host, and the sys logs on a host. Intrusion detection techniques can be classified into misuse detection or anomaly detection, based on the types of patterns for which the detector is searching. In misuse detection, one has a knowledge base of previously observed abnormal behavior (attacks) and searches for matches of current behavior with records currently in the knowledge base. In anomaly detection, one has a knowledge base of records of normal behavior and determines whether the current behavior fails to match all the records currently in the knowledge base (a mismatch indicates a possible attack). Intrusion detection techniques can also be classified as being host-based or network-based depending on the type of data source. Host-based systems monitor events on a host, such as audit data or sys logs. Network-based systems monitor packets flowing through a network.

Different techniques have both advantages and drawbacks. Misuse detection cannot detect previously unknown attacks, whereas anomaly detection tends to generate large quantities of false alerts. Moreover, neither host-based nor network-based intrusion detection can provide a reasonable coverage of all attacks, as some attacks, especially distributed ones, can only be discovered by looking into multiple data sources at the same time. Consequently, a large number of false reports (both positives and negatives) has become a common and critical issue in the intrusion detection community.

Traditional intrusion detection systems (IDSs) are typically limited by using a single intrusion detection technique. This inherent imper-

fection makes them more likely to generate false reports. A practical idea that tries to solve this problem is to have complementary IDSs working together. This has been proposed in many works. However, mainly due to the different formats of data sources and alerts, most IDSs have been developed independently and were not intended to work collaboratively.

To address this issue, a common intrusion detection framework (CIDF) has been proposed [1]. This entails a common alert format that can be used for all intrusion detection environments. Some ideas in CIDF encouraged the creation of an intrusion detection working group (IDWG) of the Internet Engineering Task Force (IETF). Two important proposals, the intrusion detection message exchange format (IDMEF) and the intrusion detection exchange protocol (IDXP), have been increasingly accepted as standards in the intrusion detection community. Our work, in particular, has concentrated on correlating IDMEF alerts.

A common alert format makes it much easier to build a system incorporating different IDSs. Such a system is often referred to as multi-sensor IDS or meta IDS. In such a system, different IDSs complementary to each other are strategically deployed at different locations watching different data sources. In this context, the notion of sensor is more general than that of an IDS and includes any security device (in hardware of software), such as a firewall or antivirus system, that can fire alerts. As sensors differ in many aspects, such as their capabilities, input data sources, and working environments, alerts from different sensors can be very different. For example, alerts from a host sensor may contain process information that a network sensor otherwise cannot provide. The essential idea is to make the sensors somehow work together. For this it is necessary to determine the logical connections between alerts, i.e., to determine which alerts are being caused by the same attack. If this can be achieved then one obtains a global view of attacks and at the same time can reduce the volumes of false alerts. The general approach is often referred to as alert correlation.

The effectiveness of an alert correlation approach determines the effectiveness of the associated IDS. For this reason, many approaches have been explored. Most popular alert correlation approaches employ some well-defined knowledge such as rules or probabilistic models, to uncover the deep connections among alerts. However, acquiring and representing such knowledge has turned out to be a nontrivial task that requires high quality training data, and such data is unavailable for most intrusion detection applications. In addition, the correctness of the resulting model is also difficult to determine, since some attack behaviors cannot be expressed naturally in rigid terms.

Case-based reasoning (CBR) is a decision-making technology that can work around these difficulties. The expertise of a CBR system is embodied in a library of past cases. A *case* consists of a description of a *problem* together with its *solution*. The design of a case-based reasoning system includes two important tasks. One is to characterize the relevant problems. This amounts to identifying the *features* that can be used to describe the problems. The other task is to define a *similarity measure* that can be used to estimate the similarity between any two problems. In contrast with other types of intrusion detection systems, the CBR methodology makes knowledge acquisition and presentation much easier. Moreover, a case library can often be built on sparse training data and still be effective.

A CBR system is assumed to be operating in the context of some environment that can give rise to problems. Given such a problem, the system applies its similarity measure to locate those cases in the case library whose problem components are most similar to it. Then the solution parts of those retrieved cases are used to suggest a solution for the problem.

In our application, problems are attacks. Each such problem is accordingly represented as a pattern of alerts from the various sensors. When dealing with the problem of alert correlation, the problem component of a case can be viewed as a concrete example of a set of correlated alerts. More exactly, our approach can be briefly described as follows. From training data, we can know exactly what true alerts the sensors fire for certain attacks. All alerts for a particular attack are aggregated as an alert pattern. That pattern, along with its associated attack type, forms a single case. Then, given a stream of runtime

alerts as input, if we can find a case whose problem component closely matches some subset of the input stream, it is reasonable to believe that the detected subset constitutes a set of correlated alerts caused by an attack of the same type as the case.

Given a set of run-time alerts, however, a large amount of computation may be required to determine whether the problem components of cases in the library match any subsets of this set. For example, if the input set has 10 alerts, and the problem component of some case in the library has 3 alerts, then one has 10-choose-3, or 120, comparisons to determine whether there are any matchings with that particular case; and this must be repeated for every case in the library.

Thus arises the issue of how to effectively find the most likely candidates in the library. Although the relation among alerts may be complicated, alerts can sometimes be correlated simply through session information, namely, source IP address and port and destination IP address and port. In fact, session information has been considered as key attributes in most intrusion detection approaches used for building intrusion detection models. Using session information suggests what we call *explicit* alert correlation. Although it is straightforward, it does have drawbacks. The important prerequisite to make this approach work is that the alerts themselves contain the session information. Unfortunately, this cannot be satisfied in the following circumstances.

- The data sources do not contain session information. If they are used for intrusion detection, sensors are not able to associate attacks with sessions. For example, antivirus software can act as a host-based sensor. It works on the binary executables and may fire alerts when virus patterns are found. But executables cannot be associated with sessions. Viruses embedded in executables can be evoked in an unpredictable manner. Other typical data sources without session information include application logs and system logs that only record high-level data such as user commands.

- The sensors do not provide session information. Different sensors have different capabilities. Depending on the environment, one may install a lightweight sensor that provides little information in its alerts, or a more complex sensor that will have a greater impact on the running system but provide more detailed alert information. Thus, even though the data source may have session information, the sensor may not use it.

- Session information is implicit in the data sources. Even in network traffic, session information sometimes is not apparent. IP packets must include both IP addresses and ports. However, packets in protocols lower than the TCP/IP protocol may not have complete session information. For example, ICMP packets only have IP addresses. An attacker may first perform an "IP sweep" (sending a series of ICMP packets) to find live machines in a particular network and then follow with attack sessions to break into those hosts by exploiting some vulnerabilities. In such cases, the ICMP packets can only be associated with sessions from the attack context.

- Not every intrusion occurs over a network. The system can be compromised by insiders, such as attacks involving an abuse of privileges or previously installed backdoor software.

The drawbacks of explicit alert correlation drove us to look for a more general approach. This lead to the idea of the case-oriented alert correlation. The basic idea is to turn the alert correlation problem into a matching problem. More specifically, for each case in the case library, the task is to find a subset of the set of run-time alerts that closely match the alerts in that case. If such a subset of alerts shows sufficient similarity to the alerts in the case, it is reasonable to believe that the set of alerts are correlated with respect the same type of attack as represented by the case. This amounts to what is known as a maximal matching problem, which can be perfectly solved by the well-known Hungarian algorithm [13]. But the downside is that the complexity of the algorithm is too high for it to be a practical solution. To work around this problem, we introduce the concept of order-preserving matching.

This is based on the observation that alerts are often fired in a sequential manner. A dynamic programming solution for the order-preserving matching is given, which has linear time complexity. Our experiment shows that this is as good as the Hungarian algorithm in solving the problem of alert correlation. Our case-oriented alert correlation has also demonstrated great potential in detecting attacks.

The rest of this paper is organized as follows. Some related work will be discussed in Section 2. Then some basic data representations employed in our approach are introduced in Section 3. We discuss case-oriented alert correlation and the Hungarian algorithm in Section 4. Sections 5 and 6 detail the dynamic programming solution for order-preserving matching and the associated algorithm. Section 7 gives our experimental results. Section 8 provides some concluding remarks.

2 Related Work

There have been several proposals that address the problem of alert correlation. Approaches such as probabilistic alert correlation [17] and alert clustering methods [2] and [7] do not use predefined knowledge other than alerts themselves and are based on the similarity between alert attributes. Although they are effective in finding similar alerts (e.g., alerts with the same source and destination IP addresses), approaches of this type are criticized for not being able to discover deep logical connections between alerts.

Some approaches, such as [5] and [11], correlate alerts according to the attack scenarios specified by human users or learned from training data by knowledge discovery approaches (e.g., data mining and machine learning). A limitation of these methods is that they are restricted to known attacks or variants of known attacks.

Approaches based on prerequisites and consequences of attacks, such as [3], [14], and [16], may discover novel attack scenarios. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful, while the consequence of an attack is the possible outcome of the attack. However, in practice, it is impossible to predefine the complete set of prerequisites and consequences. In fact, some relationships cannot be expressed naturally with the given set of terms.

Some approaches, such as [12] and [15], apply additional information sources, e.g., firewalls and system state monitors, to facilitate alert correlation. System state information may include such items as number of users, amount of memory usage, and amount of CPU usage. In particular, [12] has proposed a formal model, M2D2, for alert correlation using such multiple information sources. Because of this multiplicity of sources, their method can potentially lead to better results than those that look at alerts only. However, it invites more human involvement and makes the development of the intrusion detection system more time-consuming and error-prone in practice.

In some relevant fields such as network management and supervision, CBR has been applied to alert correlation [10]. In addition, CBR can also be used as a detection approach for a standalone system [6]. Our approach significantly differs in that it applies an XML description of problems and a generic XML distance measure.

3 Problems and Cases

Problems and cases have to be clearly defined for a specific domain where CBR is applied. Our domain is meta intrusion detection. For this we define a problem as a pattern of alerts representing an attack and a case as a previously known problem together with information regarding its identity and possibly some prescribed response. Normally, an intrusion is a series of attacks. It seems simpler to let a problem represent the entire intrusion. As the same type of attacks can be employed to launch different types of intrusions, defining a problem at the attack level allows a finer-grained analysis. The conventional approach to characterize problems is through a set of predefined features. In other words, a problem is described by a set of features together with their values. This is basically the same as the attribute-value representation employed by other knowledge or learning systems. Under such a representation, data can be put into a single table.

The information the meta IDS can obtain during an attack consists of IDMEF alerts from deployed sensors. Alert information is inherently heterogeneous. Some alerts are defined with very little information, such as origin, destination,

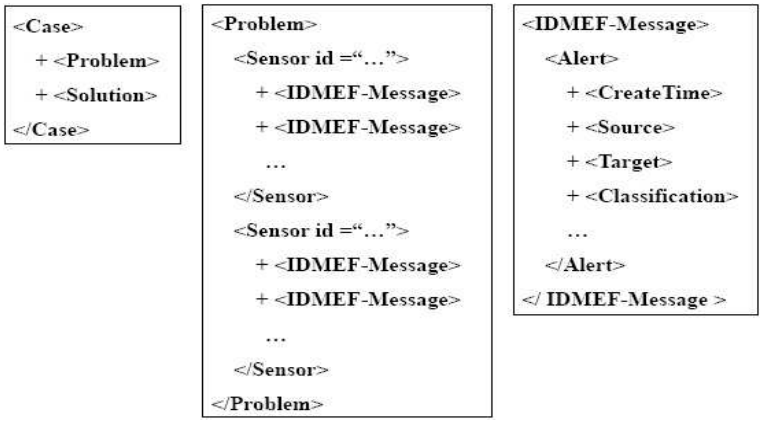

Figure 1: The XML representations of a case, a problem and an alert.

name, and time of the event. Other alerts provide much more information, such as ports or services, processes, user information, and so on. In addition, alert information may contain extended information due to the adoption of new detection approaches. Thus, predefining a fixed set of features that can cover all important information in alerts is almost impossible. Instead of attempting this, we choose a more flexible approach to describe cases in our design. The collection of alerts generated from different sensors during the attack, which forms a pattern of alerts, constitutes the description of an attack (or a problem). Since an alert is an XML object, in order to facilitate the aggregation of alerts, a problem is also represented in XML. The alerts comprising a problem are organized according to the sensors that produced them, and alerts from the same senor are sorted in chronological order. As mentioned, a case consists of the description of an attack and its identity. Figure 1 shows the representations and structures of a case, a problem, and an alert.

The XML representation of objects is inherently different from the attribute-value representation applied by conventional CBR systems. In an attribute-value representation, an object is described by a fixed number of attributes (attribute names and attribute values; attributes are called 'features' in CBR). Although the attribute-value representation is popular and used in a number of knowledge systems, it has a limitation when describing complex objects, such as trees. An XML document is a tree structure. If an XML object is transformed into a set of attributes (the process is called 'flattening a tree'), some information in it may be lost. Thus, in our approach, data analysis is performed directly over the alerts, given as XML objects, rather than working on a set of attributes (features) extracted from the original alerts.

4 The Core Methodology

We have called our approach case-oriented alert correlation because it makes use of a case library. We assume that there is a case library that contains descriptions of known attacks, where, as before, cases are identified by their patterns of alerts represented in XML. The underlying idea is, given some collection of alerts raised by the various sensors during some given time frame, to find those cases in the case library whose alert patterns are sufficiently similar to some subset of the given set to suggest that the attack described by that case has taken place.

To be more specific, the problem of alert correlation has been reduced to a matching problem as follows. The pattern of alerts in a given case and the collection of runtime alerts form two sets of alerts. An alert in one set can be matched with up to one alert in the other set. The distance between these two alerts is associated with that match. Suppose all the pair-wise distances between alerts in these two sets are available. A matching problem can be defined as one of looking for matches with the overall minimal sum of their distances. The resulting matching determines a pattern of alerts that can be found from the runtime alerts and is most similar to one in the given case. Given a set of runtime alerts, a matching problem is formulated for each case in

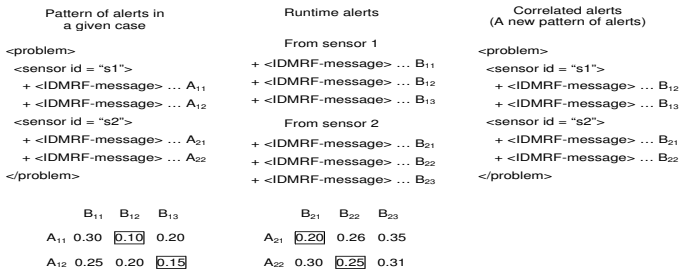

Figure 2: An example of case-oriented alert correlation.

the case library.

The manner in which these subsets are identified and applied is illustrated in Figure 2. A fragment of some given set of alerts is shown in the middle column, where the alerts are organized according to the sensors that produced them. Here there are 6 alerts, with 3 coming from sensor $S1$, denoted B_{11}, B_{12}, and B_{13}, and 3 from sensor $S2$, denoted B_{21}, B_{22}, and B_{23}. On the left is shown the alert pattern from some case in the case library. For the purposes of the illustration, this pattern also involves the same sensors, $S1$ and $S2$, although in general this need not be the case. Some cases might involve other sensors, and might not involve either $S1$ or $S2$. The pattern of alerts in the given case consists of 4 alerts, A_{11} and A_{12} from $S1$, A_{21} and A_{22} from $S2$.

We use the pattern in the case to extract a subset of the alerts in the given set and build a new (derived) pattern as shown on the right. This is done as follows. First, for each sensor, we compute the pair-wise distance between the alerts in the case and those in the given set, using the distance measure described in [8]. These are then recorded in a distance matrix as shown at the bottom of Figure 2.

To these matrices we next apply the Hungarian algorithm, to find the optimal matching, i.e., the set of alert pairs with the minimum sum of the distances. These are shown in the boxes in the matrices. The matched alerts from the given set are then extracted to form the derived set. Last we apply the distance measure again to find the distance between the derived pattern and the pattern of the case. If this distance is below some specified threshold, we take this is as meaning what was explained above, i.e., that alerts extracted from the given set are correlated in the same manner as their corresponding alerts in the case and that the attack represented by the case is likely to have occurred.

It should be noted that this is done for every case in the case library. Thus it is possible that more than one case will be matched in this manner, indicating that possibly more than one attack has taken place.

5 Order-Preserving Correlation

The methodology described in the previous section does not consider the temporal ordering of the sequences of alerts. This, however, can serve as valuable additional information. Intrusions are often carried out in phases, through a series of attacks. An earlier attack normally prepares the conditions for following attacks. In the different phases, different types of alerts are generated. As a result, alerts are fired in a temporal sequence that corresponds to the attack phases. When matching two sets of alerts, if the order of alerts in one set is the same as the order of their matches in the other set, the matching between these two sets is said to be order-preserving. A graphical illustration of this notion is given in Figure 3. If the elements in the sets are ordered, (a) and (b) show two matchings where the ordering has been broken, while (c) and (d) are two matchings where the ordering is preserved.

By imposing an order constraint on the

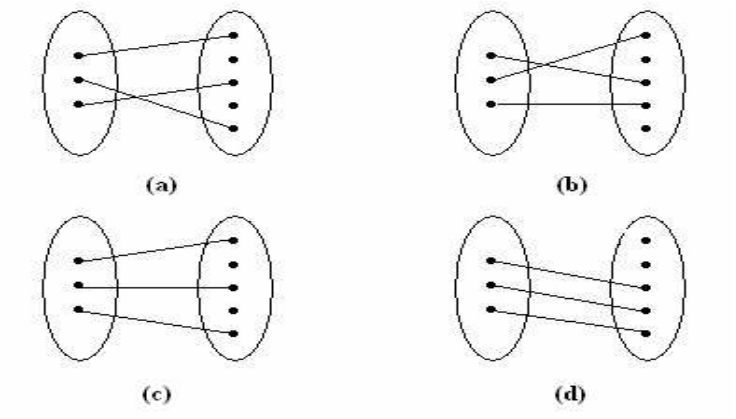

Figure 3: Arbitrary matching (a) and (b) and order-preserving matching (c) and (d).

matching, the possible matches for one element in one set is no longer all the elements in the other set. As a result, we can significantly reduce the solution search space by concentrating on a smaller set of possible matchings.

To illustrate the method, suppose we are given two sets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3, y_4, y_5\}$, with cost matrix C,

$$\begin{bmatrix} 2 & 1 & 5 & 4 & 3 \\ 6 & 7 & 8 & 9 & 10 \\ 3 & 6 & 4 & 5 & 2 \end{bmatrix}$$

where the element $c_{i,j}$ is the cost $c(x_i, y_j)$ for x_i matching y_j, which, in our application, is the distance between the two alerts. We wish to find a minimal cost matching of the members of X with those in Y that preserves their orderings. The possible matchings for each member of X, given this ordering constraint, are shown in Figure 4. The left diagram shows that x_1 can be matched with any of y_1, y_2, or y_3. The reason that x_1 cannot be matched with either y_4 or y_5 is that then there would be no order-preserving matchings for x_2 and x_3. Similar considerations apply to give the possible matchings indicated in the center and right diagrams for x_2 and x_3.

How one determines this minimal cost is the crux of dynamic programming (DP). In DP, a problem is divided into many sub-problems that can be solved easily. The solutions of those sub-problems are then used to find the overall solution of the original problem. Our method will first be explained in terms of the foregoing example, and then the general formulation will be presented. We consider the process of determining a set of matchings for the x_i as progressing through a series of stages. In the present example, the first stage is to choose a matching for x_1, the second stage is to choose one for x_2, and the third is to choose one for x_3. Each stage s has a possible cost that is determined by the possible matches for x_s. Thus, in our example, state 1 may have cost 2, 1, or 5, according to which of, y_1, y_2, or y_3 is matched with x_1. In the language of dynamic programming, if the action at stage s is to match x_s with y_k, then that stage is said to be in state k_s. The sum of the costs of all states is the cost of an order-preserving matching. Our goal is to find the minimal overall cost. Given these notations, then the minimum cost of getting to this stage and state is denoted $f_s(k_s)$. To illustrate, these costs for the stages and states in the foregoing example are determined as follows.

$$f_1(1) = 2,\ f_1(2) = 1,\ f_1(3) = 5$$

These values are simply the costs of matching x_1 respectively with y_1, y_2, y_3, as given by the cost matrix.

$$f_2(2) \quad = 7 + f_1(1) = 7 + 2 = 9$$

In stage 2, one can match x_2 with y_2 only if, in stage 1, x_1 was matched with y_1. Otherwise order would not be preserved. Hence the total accumulated cost at this stage and state is the sum of the earlier choice, given as $f_1(1)$, and the cost of the new matching, as given by the cost matrix.

$$\begin{aligned} f_2(3) \quad &= 8 + min\{f_1(1), f_1(2)\} \\ &= 8 + min\{2, 1\} = 9 \end{aligned}$$

Here the reasoning is similar, except if we are matching x_2 with y_3, then there were two possible matchings in stage 1 for x_1, namely, with y_1

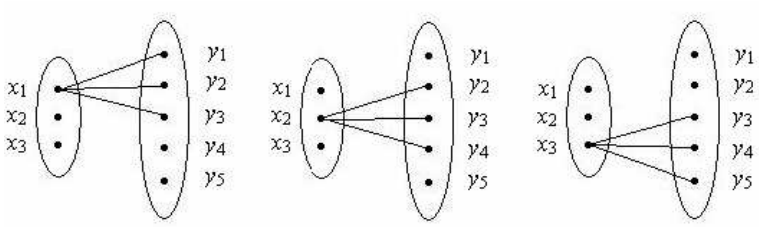

Figure 4: Possible matchings for the x_i.

or y_2. So the minimum accumulated cost at this state and stage is determined as the minimum of the previous two possibilities plus the cost of the new matching as given by the cost matrix. The remaining steps are determined in this same manner.

$$\begin{aligned} f_2(4) &= 9 + min\{f_1(1), f_1(2), f_1(3)\} \\ &= 9 + min\{2, 1, 5\} = 10 \\ f_3(3) &= 4 + f_2(2) \\ &= 4 + 9 = 13 \\ f_3(4) &= 5 + min\{f_2(2), f_2(3)\} \\ &= 5 + min\{9, 9\} = 14 \\ f_3(5) &= 2 + min\{f_2(2), f_2(3), f_2(4)\} \\ &= 2 + min\{9, 9, 10\} = 11 \end{aligned}$$

Thus the minimal possible overall cost is determined to be the minimum of the values for f_3, namely, 11.

The matchings that provide this solution are not explicitly evident from the computational process, but can be determined as part of the implementation of the dynamic programming algorithm. This will be discussed in the section below.

The general case illustrated by this example can now be described as follows. Suppose we are given two sets $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$. For purposes of this discussion, assume that X is the smaller of the two, i.e., that $n \leq m$. Then an element in X can only match with $m - n + 1$ elements in Y; the candidate matches for the ith element in X are the ith to $(i + m - n)$th elements in Y.

As in the example, we can envision this problem as being decomposed along a tree, where here the tree has n levels. Moreover, again each path upwards through the tree represents a series of stages in constructing a complete matching for the x_i. As above, where s is the stage and y_k is the element matched with x_s at this stage, the stage is said to be in state k_s, and the minimum cost of getting to this stage and state is denoted $f_s(k_s)$. The recursion formulas for dynamic programming, which compute these costs are:

$$f_1(k_1) = c_{1,k_1} \tag{1}$$

for all possible values of k_1, and

$$f_i(k_i) = c_{i,k_i} + \min_{i-1 \leq j < k_i} f_{i-1}(j), \tag{2}$$

where $1 < i \leq n$, for all possible values of each k_i. The constraint $i - 1 \leq j < k_i$ on states over which we minimize in (2) ensures the requirement for ordered matching. The computation produces different costs on different states at the final stage n, namely, $f_n(n), f_n(n+1), \ldots, f_n(m)$. The solution for the overall problem is the minimal cost, namely, $\min_{n \leq k_n \leq m} \{f_n(k_n)\}$.

6 The Algorithm for Order-Preserving Matching

From formulas (1) and (2), it seems natural that there should be a recursive solution for our algorithm. However, the implementation of our dynamic algorithm is by iteration instead of recursion. This is because a recursive solution may cause many common sub-problems to be computed repeatedly. For example, in order to compute $f_3(4)$ and $f_3(5)$, some sub-problems, such as $f_2(2)$ and $f_2(3)$ have to be computed more than once, if recursion is applied. As an optimization, however, one can compute such sub-problems once and then store the results to read back later. The iteration solution thus is introduced to reduce call overhead. In fact, such iterative implementations of dynamic programming algorithms are quite common in dynamic programming research, cf. [9].

```
Procedure initialize(C)
begin
   [n,m] = dimension(C)
   for i=1 to n
      for j = 1  to m-n+1
         C'(i, j) = C(i, i-1+j)
      end
   end
   return C'
end
```

Figure 5: Initialize the cache table

```
Procedure ordered-min-cost(C)
begin
   [n,m] = dimension(C)
   C' = initialize(C)
   for i=2 to n
      for j = 1  to m-n+1
         val = min{C'(i-1, k), 1 ⩽ k ⩽ j}
         Let idx be the index of minimal value
         C'(i, j) = val + C'(i, j)
         Attach idx to C'(i, j)
      end
   end
   return C'
end
```

Figure 6: Algorithm of order-preserving matching

The algorithm is presented in Figures 5 and 6, and the result of applying this algorithm to the 3 by 5 matrix in Section 6 is shown in 7. The routine 'initialize' in 5 is invoked by 'ordered-min-cost' in 6, and has the purpose of extracting the data of interest from the initial matrix and inserting this into a cache table. In this example, $n = 3$ and $m = 5$, and the result of executing the line "C′ = initialize(C)" is to create the cache table shown on 7(a). The result of the first iteration of the outer for loop, i.e., when $i = 2$ is shown in 7(b). The code is mostly self-explanatory. The line "Attach idx to C′(i,j)" means to create the superscript shown in Figure 7. The result of the second iteration, when $i = 2$, is shown in Figure 7(c).

This matrix can then be used to determine the ordered matchings that yield the minimal overall cost, together with this cost. These are indicated by the gray boxes in 7(d). First, the minimal value is selected (grayed). This value (i.e., 11) is the desired minimal overall cost. Then the superscript on the value (i.e., 1) is taken as the index of the cell to be grayed on the immediately preceding row (i.e., row 2). In turn, the superscript on this value (i.e., 1) is taken as the index of the cell to be grayed on its immediately preceding row (row 1). The interpretation of the grayed boxes is that the best matching (with cost 11) is obtained by having x_1 match its first candidate match, namely y_1 (see 4), having x_2 match its first candidate match, namely y_2, and having x_3 match its third candidate match, namely y_5.

7 Experiments

We used the DARPA Cyber Panel Grand Challenge Problem (GCP) program [4], an attack simulator, to evaluate the case-oriented alert correla-

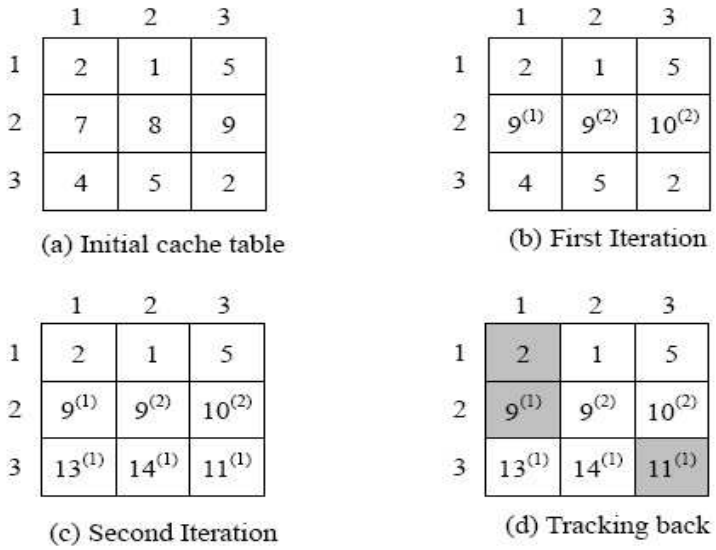

Figure 7: An example of changes in a cache table

tion methodology. The experiments assume that a meta IDS is set up to protect a corporation on the large scale. All the cases and problems for the experiments came from the attack scenarios simulated by the program. The experimental results have shown the enhanced case-oriented alert correlation is a promising approach.

7.1 The Grand Challenge Problem Simulator

The GCP models a fictitious shipping company, called Global, Inc., and allows one to simulate this company under different kinds of strategic coordinated attacks. As such, it exhibits the primary characteristics of many large, network centric organizations. The GCP scenario describes three strategic, coordinated cyber attacks that are conducted against Global. These attacks exemplify typical attack classes: life-cycle, insider, or distributed denial of service of attacks (denoted as A1, A2, and A3, respectively).

The Global corporate structure has 5 different types of network sites, namely Headquarters (HQ), Air Planning Center (APC), corporate transport ships (SHIP), Air Transport Hubs (ATH), and Partner Air Transport Hubs (PATH). Each site is defended by typical sensors, as well as postulated high-level sensors. There are in total 10 types of sensors in the networks. All the sensors generate the alerts in IDMEF messages. Simulation tools allow researchers to configure a network of arbitrary size and observe the alert streams that would be generated by sensors when attacks are perpetrated.

7.2 Experiment Setup

Running an attack scenario with the GCP simulator requires providing the following inputs:

- Specification of the number of sites of type SHIP, ATH, and PATH. This sets the size of the corporation being attacked.
- Three arbitrarily chosen integers, evidently used as seeds for random number generators applied during the attack simulation.
- Specification of the type(s) of attack. There are three choices, A1, A2, and A3. One can select any combination of these, e.g., one can simulate having simultaneous attacks of types A1 and A3.
- Specification of whether the attacks should, or should not, be accompanied with randomly generated background attacks (noise).

We experimented with a number of different site configurations, represented here as triples. To illustrate, $(1, 1, 1)$ indicates one site of each type, SHIP, ATH, and PATH, $(1, 5, 10)$ indicates one of type SHIP, 5 of type ATH, and 10 of type

PATH. For each such configuration, we built a simple case library as follows. For each attack type, A1, A2, A3, we ran one attack of this type without background noise. The alerts generated for each attack then became the problem part of a case, and the identity of the attack type formed the solution part of the case. Here it should be noted further that the simulator allows attacks of type A3 (distributed denial of service) to be generated only for configurations having one site of each type. Thus for configuration $(1, 1, 1)$, the case library has 3 cases, one for each of A1, A2, A3. For all other configurations, the case library has only two cases, one for each of A1 and A2. Different random number seeds were chosen for each run.

Attacks of combinations of A1, A2 and A3 were then run for each configuration, with background noise, and the alert streams were collected. These attacks are referred to as "problems". The alert streams were then input to our case-based reasoner, once using the Hungarian algorithm, and once using dynamic programming, as the underlying matching technique.

7.3 Experimental Results

The results of these experiments are shown in Tables 1 through 5. In Table 1, the configuration $(1, 1, 1)$ was used. The column heading C1(A1) indicates library case 1, created using an attack of type 1. Similarly for other columns. The secondary column headings "H" and "D" indicate results using the Hungarian algorithm and dynamic programming, respectively. The row labels indicate the problems. To illustrate, problem P1 involved an attack of type A1, and problem P4 involved simultaneous attacks of types A1 and A2.

The numbers in the cells are the computed distances between the given problem and case as determined by the indicated matching technique. Thus, the 0's for both H and D under case 1, for problem 1, represent an exact match, i.e., the problem represented by the case is reported to have occurred. The other cells in this table, as well as those in the other tables, are interpreted similarly.

From these tables, we can draw a few general conclusions. The closeness of the results for the Hungarian algorithm based case-oriented alert correlation and the dynamic programming based cased-oriented alert correlation approaches demonstrate that they are equally good in correlating alerts. However, the dynamic programming approach has much less complexity. The complexity of the Hungarian algorithm is given as $O((m+n)^3)$ for a m by n cost matrix. In contrast, with the same m by n cost matrix (suppose $n < m$), the complexity of the dynamic programming approach is $O(n \times (m - n + 1))$ because it only has to fill a cache table with $n \times (m - n + 1)$ cells in order to find the solution. Thus, the dynamic programming method is a more efficient solution for case-based alert correlation.

Given a problem and a case, one of two situations is possible, either the problem is represented by the case, or it is not. Whether the problem is represented by the case can be determined by means of a threshold on the distance between that problem and case. If the distance exceeds the threshold, the problem can be determined as not matching the case, and if the the distance is smaller than the threshold, it does match the case, i.e., the attack represented by the case can be assumed to have occurred.

One can determine such a threshold for each case as follows. We illustrate this for case C1 where the matching used the Hungarian algorithm. We scan through all the tables for the smallest distance between a problem and C1, where the problem does not match C1. Given the above tables, this is 0.7798, the distance between problem P6 and C1 in Table 1. We next scan through all the tables for the largest distance between a problem and C1, where the problem does match C1. This is 0.0588, the distance between problem P3 and C1 in Table 2.

Thus any value in the range $[0.0588, 0.7798]$ could be an acceptable threshold. It is desirable, however to have this be as insusceptible as possible to noisy data or variants of known attacks. A straightforward and practical approach is to use the median of the interval as the threshold. Thus, for cases representing attack A1, the threshold is given as $(0.7798 + 0.0588)/2 = 0.4193$.

Using this same technique one can compute a threshold for cases representing attack A2 as $(0.5002 + 0.3283)/2 = 0.4142$, and a threshold for cases representing attack A3 as $(0.3078 + 0.0)/2 = 0.1539$. Since the table values corresponding to

Table 1: Experimental results with (1,1,1)

	C1(A1)		C2(A2)		C3(A3)	
	H	D	H	D	H	D
P1(A1)	0	0	0.5386	0.5484	0.3078	0.3078
P2(A2)	0.7828	0.7847	0.0185	0.0185	0.3078	0.3078
P3(A3)	0.8126	0.8137	0.5697	0.5697	0	0
P4(A1,A2)	0	0	0.0185	0.0185	0.3078	0.3078
P5(A1,A3)	0	0	0.5002	0.51070	0	0
P6(A2,A3)	0.7798	0.7818	0.0185	0.0185	0	0
P7(A1,A2,A3)	0	0	0.0185	0.0185	0	0

Table 2: Experimental results with (5,5,5)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0577	0.0595	0.5030	0.5047
P2(A2)	0.8943	0.8951	0.3327	0.3327
P3(A1,A2)	0.0588	0.0576	0.3001	0.3018

the dynamic programming matching method are essentially identical to those based on the Hungarian algorithm, we will restrict our discussion to just those involving the latter.

As can be seen from the tables, these threshold values effectively identify the attacks contained in all the given problems. Thus these results suggest that the overall technique can be effective. One could, if one wished, add further tables based on further network configurations to the mix, but the present choice of configurations is sufficiently varied that such further additions will not significantly change the threshold values from those computed here.

To further validate our method for computing thresholds, one can apply the well-known leave-one-out analysis technique. To illustrate, let us first leave out problem P1 in Table 1. This means that we recompute the thresholds using all the values in the tables except those in the row P1 of Table 1. We then see if these values can be used to effectively identify the attack present in P1 of Table 1. It will be seen that it does. (Coincidentally, in this case the new computation gives the same three threshold values as computed above.) This is then repeated for every problem (row) in every table. We found that in all cases the method succeeds; the newly computed thresholds effectively identify the attacks present in each problem.

Note that a large threshold tends to make false positives and a small threshold tends to make false negatives. Thus, if in some application it is determined that false positives are less desirable than false negatives, one can lower the threshold values by some appropriate amount. Conversely, if false negatives are less desirable that false positives, one can raise the thresholds.

8 Concluding Remarks

Alert correlation is a difficult and critical issue for any multi-sensor intrusion detection environment. This paper has presented an approach using case-based reasoning for alert correlation. We call this case-oriented alert correlation. The underlying idea is to transform alert correlation into a matching problem, using either a maximal matching or an order-preserving matching. The Hungarian algorithm provides the solution for maximal matching, whereas dynamic programming is used for order-preserving matching. These were determined to be equally effective, while the latter is much more efficient. Experiments with the DARPA Grand Challenge Prob-

Table 3: Experimental results with (10,10,10)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0298	0.0298	0.5002	0.5107
P2(A2)	0.9449	0.9454	0.3282	0.3293
P3(A1,A2)	0.0298	0.0302	0.2973	0.3079

Table 4: Experimental results with (1,5,10)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0	0	0.5002	0.5107
P2(A2)	0.8596	0.8607	0.3282	0.3293
P3(A1,A2)	0	0	0.2973	0.3079

lem program have shown that our case-oriented approach is very effective in finding correlated alerts and the corresponding attacks.

The CBR methodology has several advantages. One is that it does not require extensive training data. This is significant inasmuch as large amounts of training data often is not available. In CBR, effective performance can often be obtained with only a small case library. A second advantage is that the knowledge acquisition process is generally easier. Obtaining case data and incorporating this into the case library is quite straightforward, whereas other approaches often require complex learning techniques. A third advantage is that the methodology can be applied to intrusions that occur as a result of a series of attacks, as well as those occurring in a single attack. This only requires that some expert be able to identify the relevant attacks in the sequence and write these into a case for the library, i.e., so that a case becomes a description of the entire sequence of attacks. Since CBR allows for approximate matches between problems and cases, this has the added benefit that the system can detect variants of the same attack, as long as the variance is not too great. A fourth advantage is that this methodology is easier to learn and use than approaches which, like the well-known Snort system, require that one learn a new special purpose language. A fifth is that the CBR approach is not thwarted by false/noisy alerts and quite effective in detecting multiple attacks, since the presence of such alerts seldom affects the best-matching process.

One downside of this approach is the high computation cost when there is a storm of alerts in the network. Our experiment used all the available runtime alerts, i.e., everything in the given data set. In real applications, however, one realistically can only keep the alerts occurring in some time window and shift the window over time. How big the window should be, and whether its size should be fixed or adjustable, are questions for future study. Another drawback is that the effectiveness of the intrusion detection can seriously degrade if one or more of the relevant sensors are removed from the network. This could cause cases in the library to not be matched, even though the attacks specified by those cases are actually occurring. For best performance is it necessary that all sensors corresponding to the problem features used in the case library be operational.

References

[1] A. Bundy. *Artificial Intelligence Techniques: A Comprehensive Catalogue*. Springer-Verlag New York, Inc, 4th edition, 1995.

[2] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security*

Table 5: Experimental results with (10,5,1)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0563	0.0576	0.5030	0.5047
P2(A2)	0.9304	0.9311	0.0185	0.0185
P3(A1,A2)	0.0822	0.0949	0.2973	0.3079

Application Conference, page 22. IEEE Computer Society, 2001.

[3] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202, Oakland, CA, 2002. IEEE Computer Society.

[4] DARPA. Cyber panel grand challenge problem specification version 4.1. Technical report, June 2004.

[5] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Lecture Notes In Computer Science, Proceedings of the 4th International Symposium on Recent Advance in Intrusion Detection*, pages 85–103. Springer-Verlag, 2001.

[6] M. Esmaili, B. Balachandran, R. Safavi-Naini, and J. Pieprzyk. Case-based reasoning for intrusion detection. In *ACSAC '96: Proceedings of the 12th Annual Computer Security Applications Conference*, pages 214–137. IEEE Computer Society, 1996.

[7] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 12–21. IEEE Computer Society, 2001.

[8] J. Long, D. G. Schwartz, and S. Stoecklin. An XML distance measure. In *Proceedings of the 2005 International Conference on Data Mining*, pages 119–125, 2005.

[9] R. Luus. *Iterative Dynamic Programming*. CRC Press, 2000.

[10] L. Lewis. A case-based reasoning approach to the resolution of faults in communication networks. In *Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management with participation of the IEEE Communications Society CNOM and with support from the Institute for Educational Services*, pages 671–682. North-Holland, 1993.

[11] B. Morin and D. Debar. Correlation of intrusion symptoms: an application of chronicles. In *Lecture Notes In Computer Science, Proceedings of the 6th International Symposium on Recent Advance in Intrusion Detection*, pages 94–112. Springer-Verlag, 2003.

[12] B. Morin, L. Me, H. Debar, and M. Ducasse. Correlation of intrusion symptoms: an application of chronicles. In *Lecture Notes In Computer Science, Proceedings of the 5th International Symposium on Recent Advance in Intrusion Detection*, pages 115. Springer-Verlag, 2002.

[13] J. Munkres. Algorithms for the assignment and transportation problems. *Journal Of SIAM*, 5(1):32–38, March 1957.

[14] 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*, pages 245–254, Washington, D.C., November 2002.

[15] M. W. Fong P. A. Porras and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Lecture Notes in Computer Science, Proceedings Recent Advances in Intrusion Detection*, pages 95–114, Zurich, Switzerland, October 2002.

[16] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *New Security Paradigms Workshop, Proceedings of the 2000 workshop on New security paradigms*, Ballycotton, County Cork, Ireland, October 2001. ACM Press.

[17] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Lecture Notes In Computer Science, Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, number 2212, pages 54–68. Springer-Verlag, 2001.