4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp260-269)

Application of Case-Based Reasoning to Multi-Sensor Network Intrusion Detection*

Jidong Long, Daniel Schwartz, and Sara Stoecklin
Department of Computer Science
Florida State University
Tallahassee, Florida 32306
USA

Abstract: An intrusion detection system (IDS) is generally limited by having a single detection model and a single information source for detecting attacks. Multi-sensor (or meta) intrusion detection addresses this problem by combining results of multiple IDSs and providing global decisions. Nearly all current meta-IDSs are either statistics-based or logical rule-based and typically require substantial human involvement for setup. This paper reports two experiments that employ a case-based reasoning (CBR) approach, one using the well-known 1998 DARPA datasets, which contain a variety of different types of attacks, and one using the 2000 DARPA datasets, which contain distributed denial of service (DDOS) attacks. A critical issue with meta-IDS is alert correlation: determining when alerts from the various sensors are generated by the same attack. The first experiment uses explicit alert correlation based on session information contained in the alerts. In addition, it avoids human involvement in setup by employing data mining techniques to generate the case library automatically from training data. The results show that the CBR approach is very effective in distinguishing false alerts from real attacks, and in many of the latter cases can correctly identify the type of attack. The second experiment applies CBR to achieve a kind of implicit alert correlation. Explicit correlation is not possible here, since DDOS attacks span multiple network sessions. Here again the approach has proven effective. For the second experiment the case library is derived directly from the training data without data mining techniques.

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

1 Introduction

Although intrusion detection has been explored for more than two decades, current methods still suffer from large numbers of false reports. The typical intrusion detection system (IDS) not only tends to generate a large number false alerts (false positives), but it can also miss some attacks (false negatives). This high volume and low quality of alerts make it difficult for human users of such systems to determine when an attack is taking place and how to respond.

Two primary factors contribute to false reports. One is the imperfection of detection techniques. Intrusion detection techniques can be roughly classified as either misuse detection or anomaly detection. Misuse detection works from signatures of known attacks; hence it cannot detect new attacks (produces fales negatives). Anomaly detection looks for abnormal behavior, and is criticized inasmuch as the training data cannot be guaranteed to be either clean or thorough (so that it produces false positives). The other factor is the limitation of using a single information source. Some attacks span multiple hosts, operating systems, applications, and network sessions. The complete evidence of such attacks can only be attained by investigating multiple information sources and combining their results.

An IDS is also referred to as a sensor (and sometimes an analyzer). The concept of a multi-sensor (or meta) IDS comes from the idea of cooperation among multiple IDSs. A meta-IDS detects attacks by studying the results from other IDSs, where these sensors are deployed at different locations in a network. Because sensors may apply complementary detection approaches and work on complementary information sources, the meta-IDS potentially has the ability to make more comprehensive decisions. The main task of a meta-IDS is to correlate alerts, i.e., to determine when

* This research was supported by US Army Research Office, grant number DAAD 19-01-1-0502.

alerts from the various sensors arise from the same attack. This is necessary in order to generate global alerts and discard false ones. Accordingly, alert correlation has become one of the key issues in this field, and any meta-IDS has to rely on some alert correlation approach.

In this paper, we present case-based reasoning (CBR) techniques for meta intrusion detection. In a CBR system, the expertise is recorded in a library of past cases rather than, e.g., being encoded in a set of if-then rules. In general, a case is given as a problem-solution pair. In the present application, a problem is described as a pattern of alerts from the various IDSs that has in the past been known to occur during some attack, and a solution is given as the name of the attack together with a prescribed response, such as raise an alert or take some appropriate action. The CBR system contains a library of such cases. Then, when a pattern of alerts is observed in the system being protected, this pattern is matched against the patterns in the case library based on a similarity measure. If the most similar case found is sufficiently similar, then this indicates that the attack is real (not a false alert) as well as the nature or name of the attack.

Two experiments were conducted to validate this CBR approach, one using the well-known 1998 DARPA datasets, which contain a variety of different types of attacks, and one using the 2000 DARPA datasets, which contain distributed denial of service (DDOS) attacks [6]. Both experiments used two well-known sensors: Snort [9], a network-based IDS that monitors the packets entering and leaving a network, and STIDE [2, 3], a host-based IDS that monitors the system calls on a single node in a network

The experiments may be distinguished by the manner in which they deal with alert correlation. One relies on the availability of session information (source and target IP addresses and ports) in the alerts from the various sensors. We call this *explicit* alert correlation. It applies here inasmuch as it can be assumed that each attack occurs within one session. Snort alerts contain session information, extracted directly from the packets. STIDE does not provide session information, but its input stream of system calls can be preprocessed to obtain this. Explicit correlation facilitates the CBR process inasmuch as it allows one to gather together all the alerts generated by all the sensors in the current network session and then treat this collection of alerts as a pattern that can be matched against the cases in the library. If the pattern recorded in some case in the library is sufficiently similar, this is taken to mean the attack represented by the case has occurred

The other alert correlation method applies when the session information is unavailable or is otherwise insufficient for correlation. In the present case, even though session information can be made available it would not be useful, since DDOS attacks can span multiple sessions. Our method takes the current pattern of alerts gathered from the sensors within some time frame and compares this pattern with the cases in the library. For each case, those alerts that match alerts in the current pattern to some specified degree are extracted and assembled together to form a new pattern. The similarity of this new pattern with the pattern of the original case is then computed, and if this similarity degree is large enough, it is assumed that the current pattern represents a new occurrence of the same attack as described by the original case. Thus here alert correlation is achieved as part of the CBR process. We call this *implicit* alert correlation.

Another way in which the two experiments may be distinguished is in terms of how the case library is constructed. For purposes of the first experiment, a method was developed that applies data mining techniques to construct the case library automatically from historical data. This effectively eliminates the need for human involvement. The method is comprised of three steps. First, training data is input to the sensors, and some initial cases are generated according to labeled information in training data. Second, a clustering algorithm is applied to group the initial cases. Third, from each clean cluster a representative is selected to serve as a case in the case library. In the second experiment, the case library can be derived directly from the training data without data mining techniques.

In this research, all alerts are represented in the Intrusion Detection Message Exchange Format (IDMEF) [1], an XML representation that is becoming increasingly accepted as a standard alert format. Then an alert pattern is represented as an XML document containing a collection of such IDMEF alerts. In addition, a case in the CBR library is represented in XML, where the problem part of the case adopts this same XML alert pattern representation.

2 Case-Based Reasoning

In case-based reasoning systems, the expertise is embodied in a library of past cases. This may be contrasted with systems where the knowledge is

encoded as a set of if-then rules. A case consists of the description of a problem and its solution. The knowledge and reasoning process used by the experts to solve the problem is not recorded, but it is implicit in the solution.

In general, a CBR system works as follows. A problem is recieved from the environment and represented in a proper format. It is compared with cases in the case library. A similarity measure is applied to find the most similar case. That case is returned and used to suggest a solution for the current problem. Some feedback from the environment tells whether the given solution is good for the problem. If the solution is not good enough, a new case including the revised solution is created and entered into case library for future reference.

The present application uses only the case matching and retrieval components of CBR and does not contribute new cases to the library. The software implementation uses our own adaptive case-based reasoning framework, previously reported in [8].

3 Cases and Problems in Meta Intrusion Detection

Cases and problems 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 potential attack. The collection of alerts generated from different sensors during the attack 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. A case consists of the description of an attack and its solution. Figure 1 shows the representations and structures of a case, a problem, and an alert.

An XML representation of an object 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). 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, 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 extracted from the original alerts.

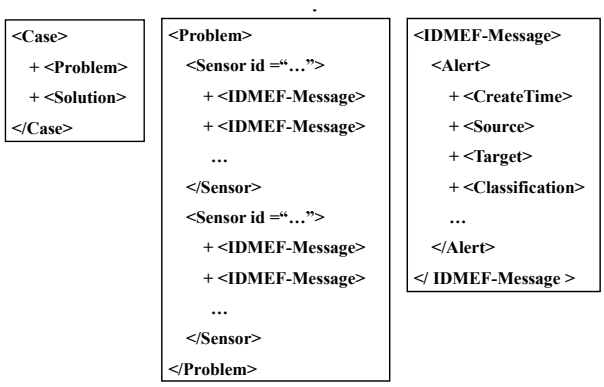

Figure 1. The XML representations of a case, a problem and an alert.

4 An XML Similarity Measure

Distance measures are used extensively in data mining and other types of data analysis. Such measures assume it is possible to compute for each pair of domain objects their mutual distance. Most distance measures work with attribute-value representations of the domain objects. In our work, however, since all objects (alerts, problems, and cases) are given in XML, is is more convenient to define a similarity measure that works directly with these XML representations. This led us to devise a general measure for the distance between two arbitrary XML documents, the details of which can be found in [5]. Briefly, it works as follows.

An XML document is a collection of XML elements organized in a tree structure in which there is one root element. An XML element can be represented as 3-tuple [e, A, E] where e is the name of the element, A is the attribute set of the element and E is the sub-element set of the element. An XML attribute consists of an attribute name and an attribute value. In case an XML element has content, its content is moved into a new attribute whose name is the element name and whose value is the content of the element content. An XML document obtained by moving the contents into attributes is called content-free. Each XML element has a sub-element set (which will be empty if the element is a leaf), and any element in that set will in turn have its own sub-element set, and so on. Thus the hierarchical organization of the original XML document is preserved in the 3-tuple representation.

Given two elements $[e_1, A_1, E_1]$ and $[e_2, A_2, E_2]$, their distance is given as $(dis(A_1,A_2)+dis(E_1,E_2))/2$,

if e_1=e_2, otherwise, it is the maximal value 1, where dis(A_1,A_2) is the distance between their attribute sets and dis(E_1,E_2) is the distance between their sub-elements. The problem of finding the distance between two sets (attribute sets or element sets) is cast into a maximal matching problem. An element in one set must have a one-on-one match in the other set. Each possible match has a cost (the distance). The maximal matching is the one that produces the minimal sum of their costs. If two sets have different sizes, a virtual element is created in the smaller set for every surplus element in the larger set. A virtual element has the maximal distance 1 to any element. The minimal sum of costs is obtained by applying the well-known Hungarian algorithm [4]. The normalized sum of distances is the final distance between two sets. Figure 2 presents an example of the computation of distance between a set of size 2 and a set of size 3.

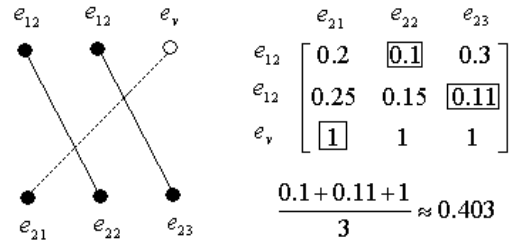

Figure 2. An example of distance between two sets.

As the basis for this distance measure, the type and distance measure for each possible kind of XML attribute must be defined. For example, the attribute 'address' is a discrete XML attribute, and the distance between two addresses is simply 0 is they are the same and 1 if they are different. As a further example, 'mismatch' (for mismatch rate) is a numeric XML attribute, and the distance between two mismatch rates r_1 and r_2 is given as $|r_1 - r_2|$ (where $0 \le r_1, r_2 \le 1$). For each attribute, the XML distance measure calls the appropriate function according to the attribute's type. Since an XML element consists of a set of attributes and a (possibly empty) set of sub-elements, the distance between two such elements is computed as noted above in terms of the distances between the attributes sets and the sub-element sets. But determining the distance between the two sub-element sets in turn requires determining the distances between their elements. Thus these two methods must call each other recursively until reaching an element with no sub-elements (a leaf). The distance between two XML objects is then defined simply as the distance between their root elements.

5 Experiment with the 1998 DARPA Datasets

5.1 Construction of the Case Library from Training Data

The library of past cases comprises the knowledge of a CBR system. A good case library should be representative of all possible problems and contain as little redundancy as possible. As was mentioned, a three-step procedure has been developed for the construction of the case library from training data: creating initial cases, clustering initial cases, and selecting final cases. These steps are illustrated in Figure 3.

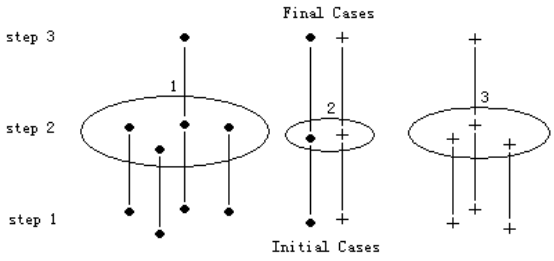

Figure 3. The three steps of case library construction.

In the first step, the training data is fed into the sensors. Sensors generate streams of IDMEF alerts. From the labeled information in training data, alerts corresponding to the same attack instance are aggregated into an XML object. Each instance of an attack is associated with an XML object of this kind if there is any alert generated for that instance. Such XML objects collected in the first step are called initial cases. The number of initial cases can be very large. It is possible that many initial cases correspond to the same type of attack and that most of them are very similar to each other. Our experiments with the 1998 DARPA datasets (more detail in Section 9) produced hundreds of similar cases associated with the type 'warezclient'. If the CBR system had to go through all these cases for every problem, the performance would be very poor.

Obviously, the use of large numbers of similar cases is redundant. In order to remove this redundancy, a clustering approach is applied in the second step. For this purpose, we developed a simple hierarchical agglomerative clustering

algorithm based on the XML distance measure discussed in the previous section. The hierarchical clustering is done in a bottom-up manner by starting with each case forming its own cluster. Next, the two clusters that are closest to each other are merged into a new cluster. Here, since the clusters are singletons, the distance between clusters is simply the distance between their two cases. For all subsequent clusterings, the step of merging the two most similar clusters is repeated until all cases are in a single cluster or the termination condition is satisfied. The termination condition is that there are no further clusters within a certain threshold distance of one another.

If there is more than one case in one cluster, the distance between two clusters is the maximal distance between a case in one cluster and a case in the other. More specifically, where $C = \{c_1, c_2, ..., c_n\}$, $C' = \{c'_1, c'_2, ..., c'_m\}$ are clusters of objects, $dis(C, C') = max\{ dis(c_i, c'_j), 1 \le i \le n, 1 \le j \le m \}$. Figure 4 gives an example. There are 5 objects (cases) with distance matrix shown on the left. When a threshold of 0.5 is set, the clustering algorithm will produce two clusters as shown on the right.

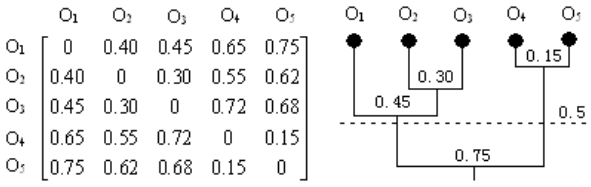

Figure 4. An example of clustering objects.

According to whether or not all objects in a cluster belong to the same type, a cluster can be marked as either pure or impure. For example, in Figure 3, clusters 1 and 3 are pure and cluster 2 is impure. A pure cluster indicates that its objects can be successfully distinguished from objects in other clusters. In contrast, an impure cluster indicates that its objects, although of different types, share some common patterns and cannot be effectively distinguished from each other. Since objects in a pure cluster are similar to each other, there is no need to compare all of them with a problem during reasoning.

Thus, in the third step, a representative of every pure cluster will be selected for entry into the case library. The method we applied to find the representatives is straightforward; we pick the one with minimal sum of distances to all other objects in the cluster. For example, a distance matrix of 4 objects in a pure cluster is given as in Figure 5. Because object 4 has the minimal sum of distances to the other objects, it is selected. For impure clusters, we do not make any change; each case in the cluster becomes its own case in the CBR library.

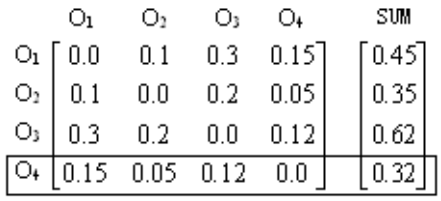

Figure 5. Example of selecting a representative case.

5.2 Method Using Explicit Alert Correlation

This employs a straightforward application of CBR. If the given alerts all contain the necessary session information, then they can be sorted according to their sessions, and then, within sessions, be sorted according to the sensors that produced them. Thus, for each session, we get a pattern of alerts that can be directly matched with the alert patterns appearing in the cases in the case library, and the detection process then amounts to retrieving the case that is most similar. The attack described by the retrieved case is assumed to have taken place.

This must be qualified, however, to cover the situation that no cases in the library are sufficiently similar to the given set to warrant this conclusion. To this end we add a requirement that the distance between the given set and the pattern in the case must be below some threshold. In our experiments, we used the same threshold as was used for the clustering algorithm, namely, 0.3.

We apply the same threshold that was used to construct the case library because that threshold can best distinguish the different types of cases. Patterns falling outside of the accepting zone are treated as normal behaviors. However, they can be instances of unknown attacks or variants of old attacks. Ignoring them causes the generation of false negatives. Generally, the solution suggested by a CBR system will be tested for success in the working environment. If a solution is not good enough to solve the problem, a case including the revised solution will be entered into case library for future reference.

5.3 The Experimental Results

The 1998 DARPA datasets provide nine weeks of data, seven weeks for training and two weeks for testing. Both training data and testing data have two

main information sources. One is BSM Audit Data for host-based IDS, the other is tcpdump data for network-based IDS. We selected STIDE [2, 3] and Snort [9] as the sensors to process these two information sources. STIDE is a host-based anomaly detection program developed to show that sequences of system calls can be used for anomaly detection. STIDE was trained with the normal sessions of some common services, such as ftp, telnet, and smtp. Most attacks in the 1998 DARPA datasets were launched against these types of services. STIDE, in its original version, produces mismatch rates and does not fire alerts. Some perl scripts were written to process the BSM audit files to find the session information. The outputs of STIDE, along with the session information found by those scripts, were represented as IDMEF messages that serve as the STIDE alerts. Snort is network-based misuse detection tool. Snort was configured as using the standard default rules and was set up with an IDMEF output plug-in [7] that can automatically transform default Snort alerts into IDMEF messages.

The labeled information of the 1998 DARPA datasets is given in list files. These list files contain the information that correlates the host sessions and network sessions and, for each session indicates whether it was normal or contained an attack, and in the latter case gives the name of the attack.

The experiment focused on the attacks against the host (named pascal) where the BSM audit data was collected. Network sessions not associated with the host have been ignored, since the information of their corresponding host sessions is not given in the training data. The case library was constructed through the three steps described in Section 5. First, the initial cases were collected. There are 440 initial cases generated for 13 types or variants of attacks. In the second step, the clustering algorithm produced 20 clusters for the initial cases when the threshold was set at 0.3. There is 1 impure cluster and 19 pure clusters, including 14 singletons (clusters having only one member). The distribution of initial cases over the clusters is shown in Table 1.

Note that instances of the 'warezclient' attack have been grouped into two pure clusters, 3 and 4. According to the method described in the Section 6, one representative from each of these two clusters was selected and entered into the case library. Two other types of attacks, 'ffb' and 'ftp_write', are processed in a similar way because all the instances of these two types have also fallen into pure clusters. Finding representatives of pure clusters greatly decreases the number of cases necessary for reasoning. For example, to find out how close a pattern of alerts is to the 'warezclient' attack, only 2, instead of 408, need to be compared. All objects in singletons and impure clusters go directly into the case library. This produced a case library having a total of 30 final cases.

type	amount	C1	C2	C3	C4	C5	C6	S
eject	5						5	
ffb	4					3		1
ffb_clear	1							1
format	1						1	
format_fail	1						1	
format_clear	1							1
ftp_write	4	2	2					
loadmodule	3						2	1
satan	8						1	7
spy	2						1	1
warez	1							1
warezclient	408			175	233			
warezmaster	1							1
Final Case	30	1	1	1	1	1	1	14

Table 1. Clustering results of the initial cases generated for the 1998 DARPA training data. The last column is singletons.

The BSM audit data and tcpdump data in the DARPA testing data were then fed into STIDE and Snort respectively. In this experiment, alerts were correlated by sessions. The IDMEF alerts generated by both STIDE and Snort for the same session were aggregated in a single XML document. Such XML documents were then sent to the CBR system as problems.

The overall results are presented in Table 2. The detection rate is the percentage of detected attacks in all attacks. An attack is detected if there is at least one alert generated for it. The false positive rate is the percentage of false alerts in all alerts. Snort may generate more than one alert for a session. We count all Snort alerts generated during normal sessions as false positives and all Snort alerts generated during attack sessions as true alerts. STIDE generates at most one alert for a session. Since it applies anomaly detection, a threshold is normally required to specify the condition of firing an alert. To be conservative, an alert is launched for any amount of detected anomalies. In other words, the threshold is zero for STIDE in the experiment. False negatives are undetected attacks. The number of false negatives can be computed as #attacks – #detected attacks. There are 6550 attack sessions (in a total of 8866

sessions) in the testing data. The CBR system fires 6537 alerts in which there are 6478 true alerts. Thus, the detection rate is 6478/6550 = 98.9%; the false positive rate is (6537 – 6478)/6537 = 0.90%; the number of false negatives is 6550 – 6478 = 72.

In our experiment, the detection threshold is set as 0.3 for the reasons discussed in Section 7.1. Problems that fall outside of the accepting zone are regarded as being normal because they do not match any known case. However, the system may miss a problem caused by a new attack or some variant of a known attack. For example, the first 'ps' attack on Monday of the first week was missed by the system. It happens that there are many types of attacks in the testing data, such as 'processtable' and 'mailbomb', that do not have any instance at all in the training data. Thus the system makes the wrong decision the first time it meets them.

We used this phenomenon to simulate updating the case library over time. A new case was entered into the case library whenever a wrong decision was made. Our experiment assumes that the update is done soon enough that the system will not make the same mistake again for identical problems. Although in practice a sequence of attacks of the same type can occur several times during a short period of time, and no update can be made on such short notice, our experiment was conducted for the purpose of demonstrating the potential of this approach.

From the results in Table 2, one can see that the system has a much better detection rate than Snort and almost the same detection rate as STIDE. It has a much lower false positives rate than either sensor. Only STIDE has a smaller number of false negatives than the CBR system. The fact that there are a large number of new attacks in the testing data contributes to most of the false negatives of the CBR system. The CBR system first missed a few new types of attacks, but after the case library was updated with these, it caught all subsequent occurrences of these attacks.

	#Alerts	Detection rate	False Positives rate	#False Negatives
STIDE	7084	99.6%	7.9%	27
Snort	6120	21.8%	76.6%	5118
CBR	6428	98.9%	0.9%	72

Table 2. Results of the first experiment.

Moreover, the system can indicate the exact types of most detected attacks. This is important in order to take appropriate actions in response to attacks. STIDE applies anomaly detection and thus is inherently unable to provide the detail of attacks. However, the alert correlation used in the CBR process allows the attack information to be obtained from the retrieved case. This is an important augmentation of STIDE.

Snort processes one network packet at a time and ignores the contexts of attacks. Although it applies misuse detection, it tends to generate a large number of false alerts, overwhelming its human observers. Our results show that the CBR system can almost exactly identify the attacks, thus greatly alleviating this problem.

7 Experiment with the 2000 DARPA Datasets

7.1 Method Using Implicit alert Correlation

This approach might be called 'case-oriented' alert correlation. Here 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. As mentioned above, 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.

The manner in which these subsets are identified and applied is described in Figure 6. 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 purpose of the illustration, this pattern also involves the same sensors, S1 and S2, although in general this need not be true. 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 Section 5. These are then recorded in a

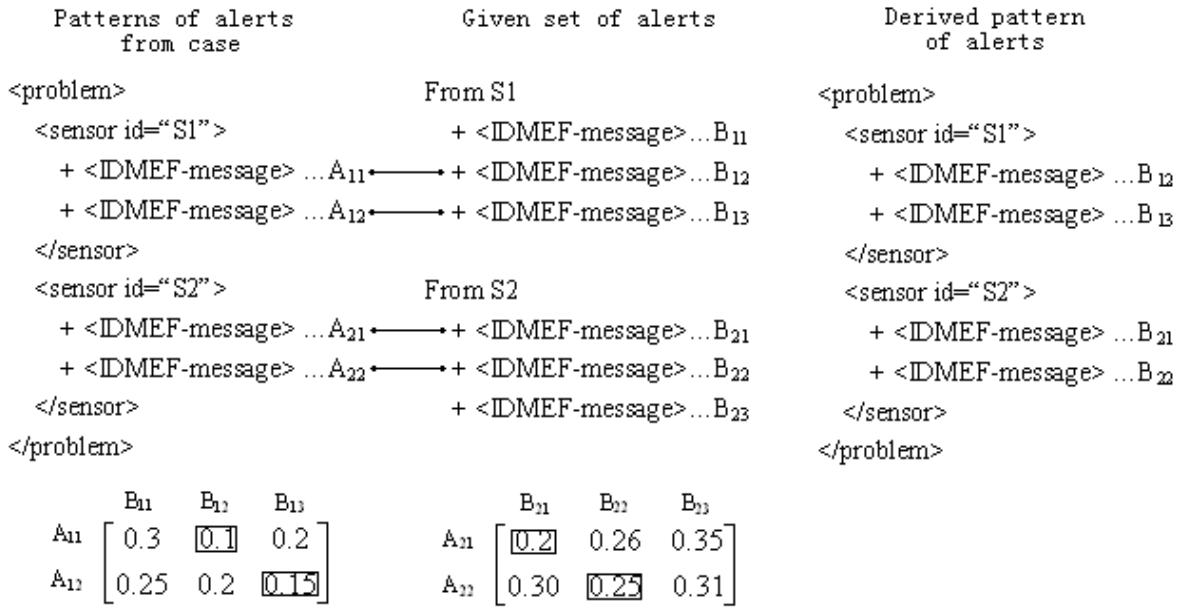

Figure 6. Example of case-oriented alert correlation.

distance matrix as shown at the bottom of Figure 6.

To these matrices we next apply the Hungarian algorithm described in Section 5, 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 of Section 5 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 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.

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 occurred.

7.2 Experimental Results

The 2000 DARPA intrusion detection scenario specific datasets include two sets entitled LLDOS 1.0 and LLDOS 2.0.2. LLDOS 1.0 contains a series of attacks in which an attacker probes, breaks in, installs the components necessary to launch a DDoS attack, and actually launches a DDoS attack against an off-site server. LLDoS 2.0.2 includes a similar sequence of attacks run by an attacker stealthier than the first one. The datasets come with low-level raw data, and mid-level labeled data. The low-level data consists of DMZ tcpdump data, inside tcpdump data, and BSM audit data from two Solaris machines, Pascal and Mill. The mid-level data consists of XML files containing IDMEF alerts generated by sensors. Our experiment was done only with mid-level data. In this experiment, LLDOS 1.0 was used to construct the case library. Then the CBR system was tested using the data in LLDOS 2.0.2. This is appropriate since LLDOS 2.0.2 is a variant of LLDOS 1.0.

The LLDOS 1.0 attack scenario is carried out over multiple network and audit sessions. These sessions have been grouped into 5 attack phases. A case corresponding to each attack phase is created for each of Pascal and Mill. All alerts that took place during an attack phase and were associated with a particular host form a case. The datasets provide each host with IDMEF alerts from three sources: DMZ, inside network, and the host itself. Accordingly, a case may have alerts from at most three sensors. Two case libraries were constructed for Pascal and Mill, respectively. Each has five cases corresponding to the five different attack phases. Since there is no redundancy in this experiment, clustering was not performed. The experiment applied case-oriented alert correlation. The correlated alerts for a particular case form a description of a problem representing a potential attack. The results are presented in Table 3.

The results have demonstrated that our system is able to effectively detect the attack described by LLDOS 2.0.2, especially on Mill. However, the first phase of the attack may not be easily recognized. The constructed problems for case-phase-1 on both machines have relatively large distances. This is because the attackers in LLDOS 1.0 and LLDOS 2.0.2 used different ways to find victim hosts. The attacker in LLDOS 1.0 performed a scripted of IPsweep to find 'live' IP addresses first and then sent 'sadmind' requests to find out possible victim hosts. The attacker in LLDOS 2.0.2 only performed an HINFO query of a public DNS server that directly returned possible victim hosts. Once the victims have been found, the rest of both attacks is almost the same. This is why the constructed problems show better similarities to their corresponding cases.

Pascal		Mill	
Case Library	Distance between case and constructed problem	Case Library	Distance between case and constructed problem
Case-phase-1	0.67	Case-phase-1	0.5
Case-phase-2	0.389	Case-phase-2	0.139
Case-phase-3	0.389	Case-phase-3	0.218
Case-phase-4	0.425	Case-phase-4	0.233
Case-phase-5	0.389	Case-phase-5	0.141

Table 3. Results of the second experiment.

There is another important difference between LLDOS 1.0 and LLDOS 2.0.2. In LLDOS 1.0, Pascal and Mill were attacked from outside of the network. As a result, alerts were generated from both the DMZ and inside the network. In contrast, in LLDOS 2.0.2, the attacker broke into Mill and fanned out from there. Mill is inside the network. The attacks from Mill to other hosts inside are invisible to the sensor at the DMZ. So, Pascal didn't accept any alert from the DMZ sensor in LLDOS 2.0.2. This difference made problems on Pascal have less similarity to their cases than problems on Mill. But if the appropriate threshold is set, say 0.4, most of the attack phases can be detected. More specifically, on Mill, all attack phases except phase 1 will be identified, and on Pascal, all attack phases except 1 and 4 will be identified.

8 Concluding Remarks

This paper has presented a CBR approach to meta intrusion detection. The basic idea behind the CBR approach is reasoning from concrete examples. The intrusion detection problem is cast into a problem of looking for the most similar example. Some common problems, such as construction of a knowledge base and alert correlation, arising in meta intrusion detection, can be dealt with in straightforward ways. The results of some experiments with the 1998 and 2000 DARPA datasets have demonstrated the potential of this approach.

References:

[1] Debar, H., France Telecom, Curry, D., Guardian, Feinstein, B. and TNT. IDMEF. http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-14.txt.

[2] Forrest, S., Hofmeyr, S., Somayaji, A. and Longstaff, T. A sense of self for Unix process. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, IEEE Computer Society Press, pp. 120-128.

[3] Forrest, S. Hofmeyr, S. and Somayaji, A. Computer Immunology, *Communication of the ACM*, v.40 no.40, pp. 88-96.

[4] Gould, R. *Graph Theory*, Benjamin/Cum-mings, 1988.

[5] Long, J., Schwartz, D., and Stoecklin, S., An XML distance measure, *The 2005 International Conference on Data Mining (DMIN'05)*, Las Vegas, Nevada, June 20-23, 2005.

[6] MIT Lincoln Laboratory. DARPA Intrusion Detection Evaluation datasets, http://www.ll.mit.edu/IST/ideval/.

[7] Poppi, S. 2004. Snort IDMEF output Plug-In, http://sourceforge.net/projects/snort-idmef.

[8] Stoecklin, S., Schwartz, D.G., Yilmaz, E., and Patel, M., A metadata architecture for case-based reasoning. In *Proceedings of The 2004 International Conference on Artificial Intelligence (IC-AI'04)*, Las Vegas, NV, June 21--24, 2004, pp. 790--794.

[9] Snort, available at http://www.snort.org.

4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp260-269)