Alert Correlation for Extracting Attack Strategies

Bin Zhu and Ali A. Ghorbani
(Corresponding author: Ali A. Ghorbani)

Faculty of Computer Science, University of New Brunswick
Fredericton, New Brunswick, Canada (Email: ghorbani@unb.ca)

(Received Oct. 7, 2005; revised and accepted Nov. 5, 2005)

Abstract

Alert correlation is an important technique for managing large the volume of intrusion alerts that are raised by heterogenous Intrusion Detection Systems (IDSs). The recent trend of research in this area is towards extracting attack strategies from raw intrusion alerts. It is generally believed that pure intrusion detection no longer can satisfy the security needs of organizations. Intrusion response and prevention are now becoming crucially important for protecting the network and minimizing damage. Knowing the real security situation of a network and the strategies used by the attackers enables network administrators to launches appropriate response to stop attacks and prevent them from escalating. This is also the primary goal of using alert correlation technique. However, most of the current alert correlation techniques only focus on clustering inter-connected alerts into different groups without further analyzing the strategies of the attackers. Some techniques for extracting attack strategies have been proposed in recent years, but they normally require defining a larger number of rules.

This paper focuses on developing a new alert correlation technique that can help to automatically extract attack strategies from a large volume of intrusion alerts, without specific prior knowledge about these alerts. The proposed approach is based on two different neural network approaches, namely, Multilayer Perceptron (MLP) and Support Vector Machine (SVM). The probabilistic output of these two methods is used to determine with which previous alerts this current alert should be correlated. This suggests the causal relationship of two alerts, which is helpful for constructing attack scenarios. One of the distinguishing feature of the proposed technique is that an Alert Correlation Matrix (ACM) is used to store correlation strengthes of any two types of alerts. ACM is updated in the training process, and the information (correlation strength) is then used for extracting high level attack strategies.

Keywords: Alert correlation, attack graph, intrustion detection, neural networks

1 Introduction

Given the immense popularity of the Internet and widespread use of automated attack tools, attacks against Internet-connected systems have become commonplace. According to the Computer Emergency Response Team Coordination Center (CERT/CC) of Carnegie Mellon University, the number of incidents reported rapidly increased from 82,094 in 2002 to 137,529 in 2003. Not only has the number of incidents increased, but the methods used by the attackers are getting more and more sophisticated. Security issues have become major concern for organizations that have networks connected to the Internet.

Intrusion detection is one of the major techniques for protecting information systems. It has been an active research area for over 20 years since it was first introduced in the 1980s. Generally, intrusion detection systems can be roughly classified as anomaly detection and signature detection systems. Anomaly detection involves building the normal behavior profile for system. The behaviors that deviate from the profile are considered as intrusions. Signature detection looks for malicious activities by searching particular patterns on the data against a predefined signature database. Recently, a new type of intrusion detection has emerged. It is called specification based intrusion detection. Normally, this kind of detection technique defines specifications for network protocols, and any activities that violate specifications are considered as being suspicious. All intrusion detection techniques have their own strengthes and weaknesses. For example, signature-based intrusion detection has a lower false positive rate, but it is intended for detecting known attacks. Anomaly-based detection has the potential to detect novel attacks, but at the same time it suffers from a high false positive rate. Moreover, it is very hard to define normal behavior for a system.

Intrusion detection systems can also be classified as host-based and network-based, depending on the data source. A host-based intrusion detection system collects data such as system calls, log files and resource usage

from a local machine. A network-based intrusion detection system detects intrusions by looking at the network traffic. These two types of intrusion detection system are complementary, neither can be replaced by the other one.

Even though intrusion detection systems play an important role in protecting the network, organizations are more interested in preventing intrusion from happening or escalating. The intrusion prevention largely relies on the knowledge about the protected network and the understanding of attack behaviors. However, studying the attack behavior is a challenging job as the adversaries tend to change their behavior in order not to be identified. And at the same time, as new vulnerabilities are continually being discovered, the attacks may use new attack strategies. One of the approaches to study the attack strategies is to extract them from the alerts that are generated by IDSs.

However, as intrusion detection systems are increasingly deployed in the network, they could generate large number of alerts with true alerts mixed with false ones. Manually managing and analyzing these alerts is time-consuming and error-prone. Alert correlation allows for automatic alert clustering, which groups logically interconnected alerts into one groups and allows easy analysis of attacks.

1.1 Related Work

Alert correlation is defined as a process that contains multiple components with the purpose of analyzing alerts and providing high-level insight on the security state of the network under surveillance. One important use of alert correlation is to recognize the strategies or plans of different intrusions and infer the goal of attacks. Suppose that the next step or the ultimate goal of an attacker can be identified by looking at the pattern of the intrusive behavior, we can take action to prevent the attack from escalating and therefore minimize the damage to the asset. Alert correlation provides a means to group different logically-connected alerts into attack scenarios, which allows the network administrator to analyze the attack strategies.

In the past few years, a number of alert correlation techniques have been proposed. Generally, they can be classified into the following categories:

- **Alert Correlation Based on Feature Similarity:** These class of alert correlation approaches correlates alert based on the similarities of some selected features, such as source IP address, target IP address, and port number. Alerts with higher degree of overall feature similarity will be correlated. One of the common weakness of these approaches is that they cannot fully discover the causal relationships between related alerts [11].

- **Alert Correlation Based on Known Scenario:** This class of methods correlate alerts based on the known attack scenarios. An attack scenario is either specified by an attack language such as STATL [5] or LAMDBA [3], or learned from training datasets using data mining approach [4]. Such approaches can uncover the causal relationship of alerts, however, they are all restricted to known situations.

- **Alert Correlation Based on Prerequisite and Consequence Relationship:** This class of approaches are based on the observation that most alerts are not isolated, but related to different stages of attacks, with the early stages preparing for the later ones. Based on this observation, several work (e.g., [9, 15, 2]) propose to correlate alerts using prerequisites and consequences of corresponding attacks. Such approaches require specific knowledge about the attacks in order to identify their prerequisites and consequences. Alerts are considered to be correlated by matching the consequences of some previous alerts and the prerequisites of later ones. In addition to the correlation method proposed by Ning et al., JIGSAW[9] and the MIRADOR[15] are other two approaches that use the similar paradigm. These approaches target recognition of multistage attacks and have the potential of discovering unknown attacks patterns. However, such approaches have one major limitation, that is, they cannot correlated unknown attacks (not attack patterns) since its prerequisites and consequences are not defined. Even for known attacks, it is difficult to define all prerequisites and all of their possible consequences.

1.2 Paper Overview

In this paper, we propose a new alert correlation technique that is based on a neural network approach. The distinguishing feature of this approach is that it uses a supervised learning method to gain knowledge from the training examples. Once trained, the correlation engine can determine the probability that two alerts should be correlated. Assigning correlation probability can help constructing hyper-alert graph and attack graphs that represent the real attack scenario. We also introduce an Alert Correlation Matrix (ACM) that can encode correlation knowledge such as correlation strength and average time interval between two types of alerts. This knowledge is gained during the training process and is used by correlation engine to correlate future alerts. And besides, different attack graphs can be generated out of the ACM, which can help security analyst to study the strategies or plans of attackers. One other benefit of using ACM is that it is incrementally updated after the training process, which enables it to discover the changes in the existing attack patterns or the newly emerging patterns.

The DARPA 2000 dataset was used to evaluated the proposed technique and system. The result shows that our approach can successfully extract attack strategies from raw alerts. The ACM is proved to be effective for improving the hyper-alert graph generation. And the in-

formation it provides is also helpful for attack trend analysis.

The rest of this paper is organized as follows. Section 2 provides details of the proposed techniques for alert correlation and attack graph generation. Two neural network approaches for alert correlation are proposed. In addition, a correlation knowledge representation scheme called Alert Correlation Matrix (ACM) that provides enhance features for alert correlation and attack graph construction is also presented. Section 3 reports the experimental results that are obtained from the DARPA 2000 dataset. We show the graphical representation of hyper-alerts generated by correlation engine, as well as the attack graphs constructed from ACM. Finally, the conclusions and some suggestions for future work are given in Section 4.

2 Proposed Alert Correlation Techniques for Extracting Attack Strategy

2.1 Overview

Attack strategies are normally represented as attack graphs. These attack graphs can be manually constructed by security experts using knowledge such as topology and vulnerabilities of the protected network. But, this approach is time-consuming and error-prone[14]. Consequently, a number of alert correlation techniques have been introduced in order to help security analysts to learn strategies and patterns of the attackers [9, 10, 4, 16]. However, all of these approaches have their own limitations. They either cannot reveal the causal relationship among the alerts (they simply group the alerts into scenarios), or require a larger number of predefined rules in order to correlate alerts and generate attack graphs.

In this paper, we propose a complementary alert correlation method that targets the automated construction of attack graphs from a large volume of raw alerts. The proposed correlation method is based on multi-layer perceptron (MLP) and Support Vector Machine (SVM). To the best of our knowledge, the support vector machine has not yet been used in alert correlation. One similar work is done by Dain et al. [4], but a different set of features are used in their report. More importantly, we use the probability output of MLP or SVM to connect correlated alerts in a way that they represent the corresponding attack scenarios.

We also introduce an alert correlation matrix into the correlation process. This is basically a knowledge base that encodes statistical correlation information of alerts. It is possible to infer causal relationship based on this information.

The following subsections describe the selection and quantification of features that are used for alert correlation, and then explain the proposed correlation technique.

2.2 Alert Correlation Matrix (ACM)

The correlation strength between two types of alert plays an important role in attack pattern analysis. It reveals the causal relationship of the two alerts. However, deciding how strong two alerts are correlated is difficult because it requires extensive knowledge about a wide range of attacks and their relations. Moreover, defining the correlation strength for all of them seems to be impossible. In [16], Valdes et al. use a matrix to define the correlation strengthes for eight classes of alerts instead of alert signatures. This approach eases the problem of having a huge number of different alerts. But it is not sufficient for attack strategy analysis and intrusion prediction. The Alert Correlation Matrix (ACM) that is proposed in this paper includes correlation weights (correlation strengths are calculated based on correlation weights) between any two alerts, and therefore provides more information for alert correlation and attack strategy analysis .

	a1	a2	a3	a4	a5
a1	*C(a1,a1)*	*C(a1,a2)*	*C(a1,a3)*	*C(a1,a4)*	*C(a1,a5)*
a2	*C(a2,a1)*	*C(a2,a2)*	*C(a2,a3)*	*C(a2,a4)*	*C(a2,a5)*
a3	*C(a3,a1)*	*C(a3,a2)*	*C(a3,a3)*	*C(a3,a4)*	*C(a3,a5)*
a4	*C(a4,a1)*	*C(a4,a2)*	*C(a4,a3)*	*C(a4,a4)*	*C(a4,a5)*
a5	*C(a5,a1)*	*C(a5,a2)*	*C(a5,a3)*	*C(a5,a4)*	*C(a5,a5)*

Figure 1: Alert correlation matrix

The formal definition of ACM is given in the following.

Definition 1 *An Alert Correlation Matrix (ACM) for n alerts $a_1, a_2, \cdots, a_n$ is a matrix with $n \times n$ cells, each of which contains a correlation weight of two types of alert.*

Figure 1 shows an ACM example of five alerts a_1, a_2, a_3, a_4,and a_5. We use $C(a_i, a_j)$ to denote a cell in ACM for alerts a_i and a_j. Note that the ACM is not symmetric. It encodes the temporal relation between two alerts. As shown in Figure 1, $C(a_1, a_2)$ and $C(a_2, a_1)$ represent two different temporal relationships. $C(a_1, a_2)$ suggests that alert a_2 arrives after a_1, while $C(a_2, a_1)$ indicates that alert a_2 arrives before a_1. By distinguishing these two situations, one can gain better understanding of the relationship of these two types of attacks.

Each cell in ACM holds a correlation weight. It is

computed as follows:

$$W_{C(a_i,a_j)} = \sum_{k=1}^{n} p_{i,j}(k)$$

n is the number of times these two types of alert have been directly correlated, and $p(k)$ is the probability of the k_{th} correlation, which is decided by a correlation engine. These correlation weights are incrementally updated during the training process.

Certain knowledge can be inferred based on the structure of the ACM and correlation weights in the ACM:

1) **Correlation Strength (Π):** Unlike correlation weight, which is an absolute value, Π is a normalized value between 0 and 1. The larger the value is, the stronger the two alerts are correlated, i.e. it is more likely that they are going to happen one after another. The values on the diagonal represent the strength of the self-correlation. In other words, it implies the probability that one type of alert may repeat itself in one scenario. For example, in DARPA LLDOS1.0 [7], the attacker tried the *SadmindBOF* attack several times with different parameters in order to exploit the vulnerability of sadmind service. So the strength of self-correlation of *SadmindBOF* alert in the ACM is high, indicating that normally the attacker will perform such attack multiple times in order to succeed.

   The ACM also encodes the temporal relationship between two types of alerts. Accordingly, two types of correlation strength are defined, namely, Backward Correlation Strength(Π^b) and Forward Correlation Strength (Π^f). The reason for having two different types of correlation strength is that, for alert correlation, we are more interested in finding which previous alerts should be correlated with the current one. And for intrusion prediction and attack strategy recognition, we are concerned about what alert is most likely to happen next. Π^b and Π^f are respectively used for those two cases. The calculation of Π^b is basically the normalization of the correlation weight of vertical elements in ACM, while normalizing the correlation weights of horizontal elements gives the Π^f.

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

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

2) **Temporal Relationship:** Given two alerts a_1 and a_2 in the cell $C(a_1, a_2)$ of ACM, we interpret it as a_2 comes after a_1. This temporal relationship is important for inferring the causal relationship. For example, for two alerts, the corresponding correlation weight with respect to different temporal relations is given by $W_{C(a_1,a_2)}$ and $W_{C(a_2,a_1)}$. If these two values turn out be to very close after training, then it is more likely that they do not have any causal relationship. On the contrary, causal relationship can be observed if, for example, one of $W_{C(a_1,a_2)}$ or $W_{C(a_2,a_1)}$ is much greater than the other one. This suggests that a_2 might be the consequence of a_1.

3) **Causal Relationship:** The causal relationship can be inferred from temporal relationship given that the training data is a good representative of the real world data. Identifying the causal relationship can help the administrator to recognize the strategies or the intention of an attacker. The ultimate goal of alert analysis is to understand the security landscape of the protected network, prevent further attack and therefore minimize the damage. All these require full or at least partial understanding of attack strategies by identifying the logical connection among the alerts.

   The ACM is incrementally updated using the output of the correlation engine. Each time the correlation engine correlates one alert with other alerts, it also provides the correlation probability between this alert and each of the alerts in the same group. This correlation probability is then used to update the corresponding cells in the ACM.

2.3 Feature Selection

When an intrusion detection system raises an alert, it also provides information associated with that alert, such as timestamp, source IP address, destination IP address, source port, destination port, and type of the attack. All of these can be used to construct the features for alert correlation. The approach presented in this report is similar to the probabilistic alert correlation technique [16] in a sense that they all involve selecting a set features, computing the probability based on these features and deciding if two alerts should be correlated. The essential difference between these two approaches is the way they compute the probability. For probabilistic correlation, the probability is computed directly from the feature similarity, while data mining alert correlation involves a statistical leaning process. For the proposed correlation technique , the following 6 features are selected.

F1: *The similarity between two source IP addresses of two alerts*

In a network-based attack, the source IP address can be viewed as the identity of an attacker. In some cases, two alerts with same source IP addresses are likely to belong to the same attack scenario and therefore could be correlated. The reason is that these two alerts might have been triggered by the malicious behavior of the same attacker. However, an attacker may use different IP addresses to perform different attacks against the target system or network. Or even worse, the attackers may spoof their IP address and then attack the target. Therefore, the source IP address cannot always be used to identify the attacker. The value of this feature indicates the likelihood that two

alerts come from the same attacker and is calculated as follows:

$$sim(ip_1, ip_2) = n/32, \quad (1)$$

where n is the maximum number of high order bits that these two IP addresses match and 32 is the number of bits in an IP address. For example:

ip_1 = **192.168.0**.001
$\Longrightarrow$ **11000000 10101000 00000000**00000001
ip_2 = **192.168.0**.201
$\Longrightarrow$ **11000000 10101000 00000000**10000001

then $n = 24$ and $sim(ip_1, ip_2) = 0.75$. This indicates that two IP addresses are from same network and could be used by the same attacker in different stage of an attack.

F2: *The similarity between two target IP addresses of two alerts*
The similarity of two target IP addresses is more important than the the one of two source IP addresses. The correlation probability are normally low in the case that the two target IP addresses are not the same (even if they belong to the same network). However, if the two source IP addresses are different, the corresponding two alerts can still have a high chance to be correlated. In order to properly demonstrate these two situations, appropriate training patterns should be selected. Again, Equation 1 is used to calculate the similarity.

F3: *If the target port numbers of two alerts are matched*
Before an attacker can exploit the vulnerability of the service that is running on a particular port, he or she must first know if this port is opened. This feature is important for correlating two attacks that target the same port. A value 0 or 1 is used to indicate the matched case and unmatched case, respectively.

F4: *If the source IP address of the current alert matches the target IP address of a previous alert*
This feature indicates that whether or not the source of an alert is the target of a previous alert. We include this feature because of the fact that, in many cases, attackers may compromise a host and use it to attack another target.

F5: *The backward correlation strength (Π^b) between two types of alert*
The value of this feature is between 0 and 1 indicating the backward correlation strength between two types of alert. For example, *SadmindPing* alerts are usually closely correlated with *SadmindBOF* alerts. Such knowledge is helpful when deciding if two alerts should be correlated. However, defining such correlation strength for each pair of alerts is almost impossible. The best way to build such knowledge base is to learn from examples, i.e. use correlation probability to update the correlation weight and correlation strength between two types of alerts.

Due the nature of alert correlation, two alerts can still be correlated without knowing the Π^b, given that other information is sufficient. Π^b can enforce the correlation when there is uncertainty in some situations such as different source or target IP addresses. To make this more clear, consider correlating two alerts a_1 and a_2 without knowing the real Π^b of these two types of alert. It is easy to conclude that they can be correlated if the following conditions hold:

```
a_1.srcIP=a_2.srcIP AND a_1.dstIP=a_2.dstIP AND
a_1.dstPort=a_2.dstPort
```

The correlation probability can be used to update the Π^b of these two types of alert. Next time when we have a similar situation, but there is uncertainty in other condition such as $sim(a_1.srcIP, a_2.srcIP) = 0.5$. The Π^b can enforce the correlation for a_1 and a_2. One problem with the use of Π^b is that, initially the information provided by Π^b may not be accurate because the correlation engine may sometimes produce incorrect result, which means that the use of Π^b may lead to incorrect correlation in the future. To address this problem, we introduce another feature ($F6$) to limit the effect of Π^b.

F6: *The frequency that two alerts are correlated*
The value of this feature is between 0 and 1. If the value of $F6$ is low, it implies that the corresponding two alerts are seldom correlated, and therefore the Π^b value is not reliable at this time due to the existence of miscorrelation. When the value of $F6$ becomes large meaning that these two alerts are frequently correlated, the percentage of miscorrelation should be low. Once Π^b has been updated many times and becomes "mature", then we can trust Π^b because these two types of alert are now statistically correlated.

2.4 Alert Correlation Using MLP and SVM

In this section, an alert correlation approach based MLP and SVM is explained. Overall, the task of such a technique is to determine:

- Whether or not two alerts should be correlated.
- If yes, the probability with which they are correlated.

Both MLP and SVM can fulfill these requirements, given that appropriately labeled training data is provided. The correlation probability is required because it is useful for practical recognition cases, and also reflects the uncertainty of correlation.

2.4.1 Alert Correlation Using Multi-Layer Perceptron

Figure 2 shows the MLP structure that is used for alert correlation. It contains six input, one hidden layer with seven neurons, and one output. The input is a vector of

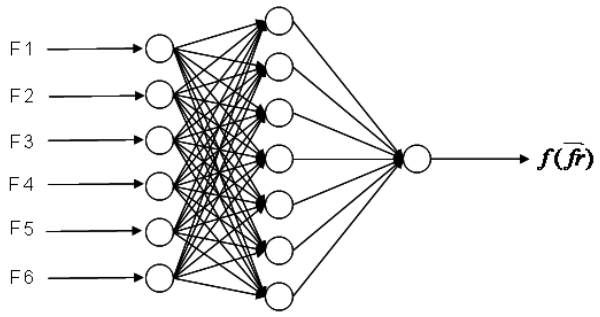

Figure 2: MLP structure for alert correlation

6 elements, each of which represent one of the features defined above.

$$\overrightarrow{fr} = [F1, F2, F3, F4, F5, F6]$$

The output of this MLP is a value between 0 and 1, indicating the probability that two alerts are correlated.

2.4.2 Alert Correlation Using Support Vector Machine

Support Vector Machines was first introduced by Vapnik and his coworkers in 1992 [1]. It is an approximate implementation of the method of structural risk minimization [6]. It has been proved to be very robust for pattern classification and nonlinear regression, especially when the dimension of the input space is very large. The conventional SVM is a binary classifier. Alert correlation can be viewed as a binary classification problem. For two given alerts, we determine if they are correlated or not. This is the main motivation of using SVM for alert correlation.

The same input used for the MLP is also used to train the SVM, the labels of training patterns are mapped to bipolar format where 1 represents *correlated* and -1 represents *not correlated*. Unlike MLP, SVM does not require probability to be assigned to the training patterns, the labels that are assigned to the training patterns are simply the class labels (in this case, 1 and -1). This is one major advantage of using SVM. However, the problem is, the output of a conventional SVM is not probability [13]. Recall that the output of a conventional SVM with respect to certain input is either 1 or -1 indicating the class of that input. The output of the SVM (before applying a threshold) is calculated as follows:

$$f(x) = \sum_{i=1}^{l} y_i \alpha_i \cdot K(x, x_i) + b.$$

This is the distance between input vector and optimal hyperplane, which is an unbounded value. As we can see, both of these two outputs are not probabilities. Platt [13] shows that fitting a Sigmoid that maps SVM output to posterior probability produces good mapping from SVM margins to probability. According to Platt, the probability output of SVM is given by:

$$p_i = \frac{1}{1 + exp(Af_i + B)},$$

where f_i is the output of the conventional SVM. A and B can be obtained by minimizing the corresponding cross-entropy error function in the following [13]:

$$\sum_i -t_i log(p_i) - (1 - t_i) log(1 - p_i)$$

where the t_is are target probabilities. The probability output of SVM is a continues value between 0 and 1. A higher value indicates the higher probability with which two alerts are correlated.

2.4.3 Comparison of MLP and SVM

We use both MLP and SVM for alert correlation. MLP uses an error-correction rule to update the weight matrixes. Once it converges, the outputs it produces will be close to the desired outputs. The SVM is based on structural risk minimization principal. It needs only class labels to be assigned to the training examples. Therefore, its probabilistic outputs might not be as accurate as the ones produced by MLP. However, it is not easy to precisely assign desired outputs for MLP. Furthermore, MLP also suffers from the slow training speed and the potential over-fitting problem. On the other hand, in order to enable SVM to produce precise probabilistic outputs, appropriate training patterns have to be selected, which is also a difficult task. Therefore, there is no absolute preferable solution. A better way may be to make a decision based on the outputs of both of these two methods.

2.4.4 Correlation Method

In order to use MLP or SVM as a correlation engine, they have to be trained first. In the correlation approach proposed by Dain [4], the training examples are taken from the real alerts. The author manually classified and labeled 16,250 alerts. This requires expertise knowledge of attack scenarios and huge amount of labor work. In this work, both MLP and SVM are trained with small number of patterns, which are manually generated and labeled. Table 1 shows the 18 training patterns that are used in the training. These are some typical cases. MLP and SVM are expected to generalize other cases by learning from these examples. For the training of MLP, the desirable correlation probabilities are assigned to each training pattern, and for SVM only the class labels are assigned. The label values for the MLP training examples is determined by comparing one example with others. The extreme case is that all the features of two alerts are the same (except F3 because it contradicts with F1 and F2). In this case, it is reasonable to assign a maximum correlation probability. And a value of 0 is assigned to the other extreme case where the feature values are all zero. The desire correlation probability for the rest examples are determined

Table 1: Training results for MLP and SVM

							MLP		SVM	
	F1	F2	F3	F4	F5	F6	Label	Output	Label	Output
1	1	1	0	1	1	1	1	0.9900	1	0.9926
2	1	1	0	0	0	0	0.75	0.7497	1	0.8668
3	1	1	0	0	0.5	0.5	0.85	0.8503	1	0.9222
4	0.5	1	0	0	0.5	0.5	0.8	0.8001	1	0.8668
5	0.5	0.5	0	0	0.1	0.3	0.1	0.1007	-1	0.1982
6	0	1	0	0	0.1	0.2	0.2	0.2000	-1	0.2270
7	1	0.5	0	1	0.5	0.3	0.65	0.6500	1	0.9031
8	0	0	0	0	0	0	0	0.0002	-1	0.1252
9	0.5	1	0	0	1	1	0.85	0.8500	1	0.9315
10	0.5	0.5	0	1	1	1	0.8	0.8005	1	0.9482
11	1	1	0	1	0	0	0.9	0.9003	1	0.9263
12	0.5	0.5	0	0	0.5	0	0.15	0.1499	-1	0.2270
13	0	0	0	1	1	1	0.65	0.6498	1	0.8668
14	0	0	1	1	1	1	0.9	0.9002	1	0.9782
15	0	0	1	0	0.5	0	0.8	0.8000	1	0.8808
16	0	0	1	1	0.5	0.5	0.85	0.8498	1	0.9577
17	0	0	1	0	0	0	0.8	0.6900	1	0.8668
18	0.5	0.5	0	0	0.5	1	0.3	0.2998	-1	0.1727

by comparing them with the ones that have been previously labeled ones. Labeling training example for SVM is easier, since we only need to assign 1 to the example if we conclude that in that case, two alerts should be correlated, otherwise, -1 is assigned. Labeling training examples for both MLP and SVM require certain degree of knowledge about the similarity measurement. But it is not directly related to any particular alerts.

The output of the correlation engine is used to determine whether or not two alerts are correlated. The next step is to define a representation for the group of correlated alerts. The representation used in this report is called *hyper-alert*. The goal of the correlation process is to construct a list of hyper-alerts. The following gives the formal definition of a hyper-alert.

Definition 2 A hyper-alert $H = (V, E, C)$ is a directed graph where V is the vertex set and E is the edge set, each $v_i \in V$ corresponding to one primitive alert a_i. An edge $(v_i, v_j) \in E$ represent the temporal relationship between alert a_i to a_j (a_i arrives earlier than a_j), and $c_{i,j} \in C$ represents correlation probability.

Since hyper-alerts are defined as graphs, in the rest of this paper hyper-alerts and hyper-alert graphs are used interchangeably to represent the correlated alerts. Note that a new alert is not always linked to the latest alert in the hyper-alert. Instead, it connects to the alerts with which it has a high correlation probability. So, the representation is useful for inferencing the multiple goals of attackers. The intention of using graph representation for correlated alerts is to give the network administrator intrinsic view of attack scenarios.

In order to construct the hyper-alert graphs, two thresholds are defined. The first threshold is called the *correlation threshold*. Since the probabilistic output of MLP or SVM is between 0 and 1, intuitively, a value of 0.5 is a suitable threshold to decide if two alerts should be correlated or not. If the correlation probability between two alerts is greater than 0.5, then they are considered to be correlated alerts. Otherwise, they are considered to be independent alerts. The other threshold is called the *Correlation Sensitivity*. Normally, when constructing a hyper-alert graph, the new alert is connected to a previous alert that has the highest correlation probability with it. But, in some cases, it is possible for this new alert to connect to more than one previous alert. This represents the situation that an alert is the direct consequence of more than one previous alert. Assume that $Cmax$ is the maximum correlation probability the correlation engine produces for a new alert and all the alerts in a hyper-alert. The correlation sensitivity is a threshold that determines whether or not an alert in the hyper-alert should be connected to the new alert. Only those whose correlation probability with the new alert is close to $Cmax$ such that the difference is less than the correlation sensitivity will be connected to the new alert.

The correlation process is described in Algorithm 1. When a new alert is received, it is compared with all previous alerts in existing hyper-alerts, and the output of correlation engine and the correlation sensitivity are used to determine which hyper-alert it should join, and to which primitive alerts it should connect. If the correlation probability with respect to the new alert and all primitive alerts is lower than a threshold (correlation threshold), a new hyper-alert will be created to include this new alert. Once two alerts are correlated, the corresponding cell in ACM will be updated accordingly. ACM is incrementally updated so that the new correlation strength can be ap-

plied to next correlation immediately.

2.5 Generating Attack Graph Using ACM

The approach that is proposed in this report to generate attack graphs is different from the ones that use vulnerability and topology information about the protected network. In our approach, the attack graph are generated based on the training data and the output of the alert correlation algorithms. The ACM provides enough information for generating different attack graphs that represent different typical attack strategies.

As mentioned above, one of the main purposes of using the ACM is to construct attack graphs in order to study the strategies of attackers. By knowing the strategies, it is possible for network administrators to predict what is most likely to happen when they observe a part of a pattern from the received alerts. An attack graph is defined as follows:

Definition 3 An attack graph $G = (V, E, C)$ is a directed graph where V is the vertex set and E is the edge set, each $v_i \in V$ corresponding to one type of alert a_i. An edge $(v_i, v_j) \in E$ represent the state transition from alert a_i to a_j, and $c_{i,j} \in C$ represents Π_f of alerts a_i and a_j.

Note that there are essential differences between a hyper-alert graph and an attack graph that is generated from the ACM:

- First, a hyper-alert graph is a directed acyclic graph. There are no cycles in the graph because of the temporal constraints of the alerts. On the other hand, an attack graph is a directed graph that can have cycles. The navigation of an edge represents the direction of the state transition rather than a temporal relationship. Therefore, a hyper-alert graphs can be considered as a specialized instance of an attack graph. Attack graphs are a more general representations of attack strategies, which encode causal relationship among alerts.
- As a hyper-alert is an instance of the corresponding attack graph, it can consist of multiple alerts of the same type, which are normally distinguished by their time-stamps. But, for an attack graph, each type of alert can only appear once in the graph. If certain type of alert occurs multiple times in a scenario, it is represented as a cycle in the attack graph.

The process of constructing attack graphs from ACM is described in Algorithm 2. The graph construction starts from a given type of alert that represent a particular type of attack. The algorithm performs a horizontal search in the ACM to find the alerts that are most likely to happen after this alert. Then these alerts become new starting points to search for alerts that are more likely to happen next. The process is repeated until no other alerts are found to follow any existing alerts in the graph.

Algorithm 2.2: ConstruGraphsFormACM(a_i, r)

```
a: the starting alert of the attack graph
r: the Π^f threshold
initialize graph G
initialize queue q
q ← a_i
G ← a_i
while not q.isEmpty()
do { a ← q.dequeue()
     for j ← 0 to number of alerts in ACM
     do { if Π^f_{Cell(a,a_j)} > r
          then { if a_j has not been visited
                 then { q ← a_j
                        Visit a_j
                        G ← G ⋃ a_j
                        G ← G ⋃ (a, a_j)
                        c_{i,j} ← Π^f_{Cell(a,a_j)}
                        G ← G ⋃ c_{i,j} } } } }
return (G)
```

3 Test and Evaluation

3.1 Experiment with DARPA 2000 Dataset

The proposed Alert Correlation Scheme (ASC) has been tested using the DARPA 2000 dataset[7]. DARPA2000 is a well-known IDS evaluation dataset created by the MIT Lincoln Laboratory. It consists of two multistage attack scenarios, namely LLDDOS1.0 and LLDOS2.0.2. The purpose of this experiment is to evaluate ACS's ability to correlate alerts, discover these scenarios and extract attack strategies.

3.1.1 LLDOS 1.0 - Scenario One

The LLODS1.0 scenario can be divided into five phases as follows.

- **Phase 1:** The attacker scans the network to determine which hosts are "up".
- **Phase 2:** The attacker then uses the *ping* option of the *sadmind* exploit program to determine which hosts selected in Phase 1 are running the Sadmind service.
- **Phase 3:** The attacker attempts the sadmind Remote-to-Root exploit several times in order to compromise the vulnerable machine.

Algorithm 2.1: CORRELATION(A, r, s)

A: List of alerts
r: Correlation threshold
s: Correlation sensitivity
initialize hyper-alert list H
for all each alert a_i in A
do
- **for** all hyper-alerts in H
- **do**
  - find an hyper-alert h_j that contains an alert a_j
  - such that the correlation probability of a_i and a_j is maximum
  - $m \leftarrow$ this maximum correlation probability
  - **if** $m > r$
  - **then for** each alert a_k in h_j
  - **do**
    - **if** m-(probability between a_k and a_i) $< s$
    - **then** connect a_i with a_k
    - **else**
      - create a new hyper-alert
      - put a_i in new hyper-alert

- **Phase 4:** The attacker uses *telnet* and *rpc* to install a DDoS program on the compromised machines.
- **Phase 5:** The attacker telnets to the DDoS master machine and launches the *mstream DDOS* against the final victim of the attack.

Table 2: 19 types of alerts reported by RealSecure in LLDOS1.0

ID	Alert Name
1	Sadmind_Ping
2	TelnetTerminaltype
3	Email_Almail_Overflow
4	Email_Ehlo
5	FTP_User
6	FTP_Pass
7	FTP_Syst
8	HTTP_Java
9	HTTP_Shells
10	Admind
11	Sadmind_Amslverify_Overflow
12	Rsh
13	Mstream_Zombie
14	HTTP_Cisco_Catalyst_Exec
15	SSH_Detected
16	Email_Debug
17	TelnetXdisplay
18	TelnetEnvAll
19	Stream_DoS

We use an alert log file [12] generated by RealSecure IDS. As a result of replaying the "Inside-tcpdump" file from DARPA 2000, Realsecure produces 924 alerts. After applying the proposed correlation approach (for both LLDOS1.0 and LLDOS2.0.2, with correlation threshold $r = 0.5$ and correlation sensitivity $s = 0.1$), the ACM contains 19 different types of alerts ahown in Table 2. The correlation weights of these alerts are given in Table 3. One of the hyper-alert graphs is given in Figure 3. It correctly represents the DARPA LLDOS1.0 scenario that is described above. Phase 1 is missing in this hyper-alert graph because RealSecure dos not raise any alert for the ICMP probing activity executed by the attacker, although in phase 2, the attacker triggers multiple *Sadmind_Ping* alerts. By using *AlertAggregator*, the ACS aggregates the same type of probing attacks from the same source into one *Sadmind_Ping* alert.

Phase 3 consists of multiple $Sadmind\_Amslverify\_Overflow$ and *Admind* alerts. All these alerts have the same source IP and destination IP implying that the attacker tries several times to break into the system running *Sadmind* service by using a buffer-overflow attack. According to the description of Realsecure, the *Admind* is also an unauthorized access attempt to remote administration of Solaris machines. Thus, we believe that RealSecure raises two alerts for a single attack. This can also be observed from the number of these two attacks and their timestamp. In Phase 4, the attacker uses *rsh* and *telnet* to install and start *mstream* daemon and master program, which triggered the *Rsh*, $MStream\_Zombie$ and some telnet alerts. The telnet alerts are not correctly correlated because the correlation engine produces low correlation probabilities for them. Moreover, the *mstream_DDOS* alert in phase 5 has not been correlated into the hyper-alert graph shown in Figure 3 because the attacker was using spoofed IP addresses, and initially, the correlation weights for *mstream_DDOS* and any other alerts is set to a very small value. Therefore, if the connection between *mstrean_DDOS* and *Mstream_Zombie* is desired, the corresponding correlation weight must be initialized with a large value.

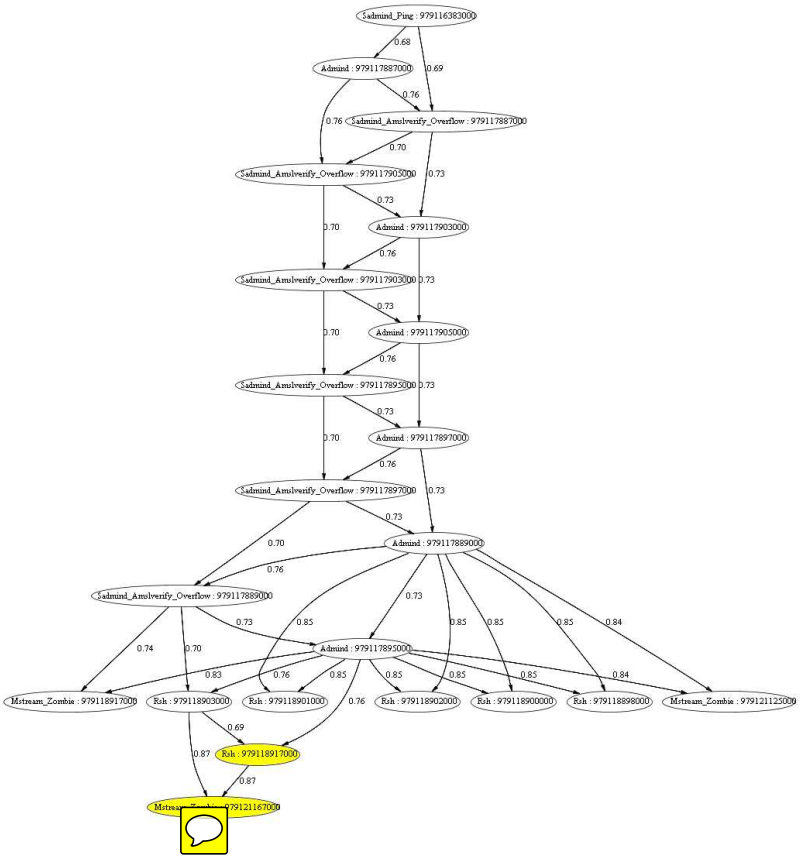

Figure 3: LLDOS1.0 hyper-alerts generated by using updated ACM

Three such hyper-alert graphs are generated by the ACS, representing the scenarios that the attacker uses the same method to compromise three hosts in the network. This is consistent with the description in the DARPA LLDOS1.0 documentation. In addition to the LLDOS1.0 attack scenario described above, the ACS also identifies some other scenarios. For example, an attacker tries to gain unauthorized access to several hosts inside the network. This activity triggers multiple *Email_Ehol* alerts. These alerts are correlated by the ACS to generate the corresponding hyper-alert. One other similar scenario discovered by the ACS contains multiple FTP alerts such as FTP_User, FTP_Pass and FTP_Syst. However, all of these scenarios are not documented in DARPA2000 dataset. One possible explanation is that, the attacker performed multiple unauthorized accesses to several machines, but failed to break in, and thus the intrusion does not escalate.

Figure 3 illustrates the specific attack scenarios. It is possible for a security analyst to recognize the attack strategy by analyzing the hyper-graph. However, attackers might try to confuse the correlation system in order to cover their attack plans. They can generate some "noise alerts" and change the order of the steps. Therefore, hyper-alert graphs that are generated by correlation system could be very large and complex and contain many irrelevance alerts. It might not be very useful for the security analyst to understand the attackerś strategies or plans.

The attack strategy extraction method proposed in this paper is based on the statistical information in the ACM. It has the potential to filter out the "noise alert" and find the sequence of actions that attacker use to achieve the intrusive goal. Figure 4 is one of the attack graphs extracted from the ACM, while Table 4 shows the corresponding forward correlation strengths in ACM. This is a representation of the attack strategy used by the attacker in the LLDOS1.0 scenario. It shows the major steps and all the possible transitions that the attack followed in order to achieve its goal. Note that the self-looped cycles in the graph represent the situation that the attacker may repeat these steps in order to succeed. As shown in the hyper-alert graph (Figure 3), the attacker successfully breaks in the machine after several tries of Sadmind buffer-overflow attack. The cycle contain only two nodes $(Sadmind\_Amslverify\_Overflow, Admind)$ also implies that these two alerts are actually triggered by the same attack. Recall the discussion about causal relationship in the ACM. We can conclude that there is no causal relation between these two alerts. They can be viewed as two parallel events. On the other hand, the *Sadmind_Ping* attack can be considered as the preparation of $Sadmind\_Amslverify\_Overflow$ attack. Figure 4 only shows the most frequent paths or transitions since Π^f threshold is set to a high value, as described in $ConstruGraphsFormACM$ algorithm.

Table 3: Correlation weights in ACM

Alert	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0.3	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	9.3	8.16	6.27	2.37	0.01	0.01	0.01	0.69	0.69	0.01
2	1.75	29.87	19.54	139.09	16.1	19.29	16.11	0.01	0.01	3.79	2.17	2.33	0.6	0.01	3.49	1.29	0.01	0.01	0.01
3	0.87	45.74	34.84	228.07	29.52	25.27	24.86	12.25	0.65	1.68	1.12	1.23	0.32	0.98	0.92	1.1	0.01	0.01	0.01
4	1.75	782.27	628.25	3533.93	550.71	528.85	527.02	13.49	0.65	4.37	2.2	2.39	0.62	4.16	17.87	48.31	0.01	0.01	0.01
5	0.01	9.03	5.32	29.57	19.71	27.31	26.21	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.27	1.48	0.01	0.01	0.01
6	0.01	9.03	1.09	29.05	20.79	19.71	27.33	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.27	1.48	0.01	0.01	0.01
7	0.01	8.76	1.09	29.05	21.22	20.15	19.35	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.27	1.48	0.01	0.01	0.01
8	0.01	11.82	0.01	29.34	3.06	3.06	3.06	11.53	3.88	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
9	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.64	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
10	0.7	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	30.79	31.53	72.87	26.25	0.01	0.01	0.01	4.11	4.12	0.01
11	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	31.4	22.74	32.14	16.96	0.01	0.01	0.01	4.11	4.12	0.01
12	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	20.88	2.79	0.01	0.01	0.01	3.29	3.29	0.01
13	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.97	0.01	0.01	0.01	0.01	0.01	0.01
14	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.35	0.01	0.01	0.01	0.01	0.01
15	0.01	0.88	0.01	2.64	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.32	1.17	0.01	0.01	0.01
16	0.01	1.16	0.01	4.24	0.28	0.28	0.28	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.27	0.01	0.01	0.01
17	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	1.08	0.01	0.01	0.01	0.01	0.69	0.01
18	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	1.08	0.01	0.01	0.01	0.01	0.01	0.01
19	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

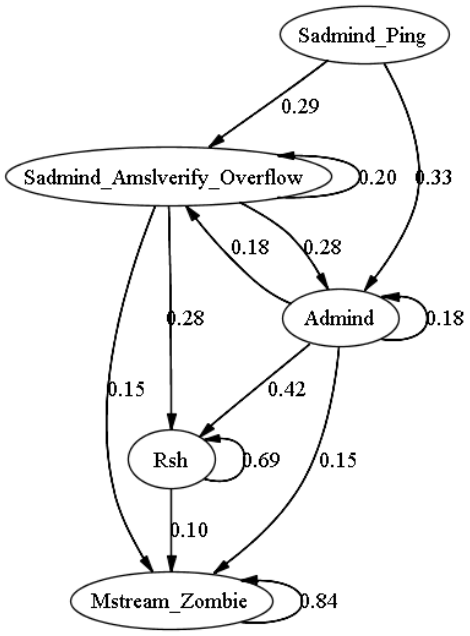

Figure 4: An attack graph for LLDOS1.0 extracted from ACM

3.1.2 LLDOS 2.0.2 - Scenario Two

The second scenario in DARPA 2000 dataset also consists of five phases that starts from probing and ends with a DDoS attack. The main difference between LLDOS1.0 and LLDOS2.0.2 is that, in LLDOS2.0.2, the attack uses a HINFO query instead of ICMP echo reply to discover live hosts. Moreover, the attacker uses a more stealthy way to compromise several hosts and install trojan mstream DDoS software. To do so, he/she first breaks in a host, and then uses it to compromise other hosts in the network, instead of compromising them individually from the same source as in LLDOS1.0 scenario. The attack scenario consists of the following five phases:

- **Phase 1:** The attacker probes a machine (172.016.115.020), which is a DNS server in the network.
- **Phase 2:** The attacker breaks in this machine via exploiting the Sadmind vulnerability.
- **Phase 3:** The attacker uses FTP to upload mstream DDoS software and attack script to the compromised machine.
- **Phase 4:** The attacker tries to break in two more machines, but only one attempt is successful .
- **Phase 5:** The attacker telnets to the DDoS master machine (the first compromised machine) and launches the *mstream DDOS* against the final victim of the DDoS attack.

The experimental result is similar to the one in LLDOS1.0. Realsecure generates 17 different types of alerts (see Table 5). The total number alerts in this scenario is 494. Again, Realsecure does not raise alerts for the probing activity. Therefore Phase 1 is not shown in the corresponding hyper-alert graph. Phase 5 is also missing in Figure 5 since the attacker again uses spoofed IP addresses. Instead, the *Stream_DoS* is correlated with a *Port_Scan* alert as a separate hyper-alert graph. The IDS identifies other phases by raising the following alerts: $Sadmind\_Amslverify\_Overflow$, *Admind*, Ftp_Put and $MStream\_Zombie$. These alerts are correctly correlated by the ACS. Figure 5 is the corresponding hyper-alert graph that is generated to represent this scenario.

As we mentioned above, one of the main difference between LLDOS1.0 and LLDOS2.0.2 is the way in which the attacker comprises several machine in the network. This difference can only be observed by looking at the IP addresses of $Sadmind\_Amslverify\_Overflow$ and *Admind* alerts. ACS allows user to explore more information about a particular alert by selecting them. With this feature, one can identify that, in the hyper-alert graph shown in Figure 5, the target of first

Table 4: Forward correlation strength in ACM

Alert	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0.011	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.292	0.225	0.085	0.000	0.000	0.000	0.025	0.025	0.000
2	0.007	0.117	0.076	0.544	0.063	0.000	0.063	0.000	0.000	0.015	0.008	0.009	0.002	0.000	0.014	0.005	0.000	0.000	0.000
3	0.002	0.112	0.085	0.557	0.072	0.062	0.061	0.030	0.002	0.004	0.003	0.003	0.001	0.002	0.002	0.003	0.000	0.000	0.000
4	0.000	0.118	0.095	0.532	0.083	0.080	0.079	0.002	0.000	0.001	0.000	0.000	0.000	0.001	0.003	0.007	0.000	0.000	0.000
5	0.000	0.076	0.045	0.248	0.166	0.229	0.220	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.012	0.000	0.000	0.000
6	0.000	0.083	0.010	0.267	0.191	0.181	0.251	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.014	0.000	0.000	0.000
7	0.000	0.086	0.011	0.286	0.209	0.199	0.191	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.003	0.015	0.000	0.000	0.000
8	0.000	0.179	0.000	0.445	0.046	0.046	0.046	0.175	0.059	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.780	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012
10	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.181	0.185	0.427	0.154	0.000	0.000	0.000	0.024	0.024	0.000
11	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.281	0.204	0.288	0.152	0.000	0.000	0.000	0.037	0.037	0.000
12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.687	0.092	0.000	0.000	0.000	0.108	0.108	0.000
13	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.843	0.009	0.009	0.009	0.009	0.009	0.009
14	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.019	0.660	0.019	0.019	0.019	0.019	0.019
15	0.002	0.171	0.002	0.512	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.062	0.227	0.002	0.002	0.002
16	0.002	0.175	0.002	0.639	0.042	0.042	0.042	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.041	0.002	0.002	0.002
17	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.557	0.005	0.005	0.005	0.005	0.356	0.005
18	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.857	0.008	0.008	0.008	0.008	0.008	0.008
19	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053

Sadmind_Amslverify_Overflow alert is actually the source of the second *Sadmind_Amslverify_Overflow* alert.

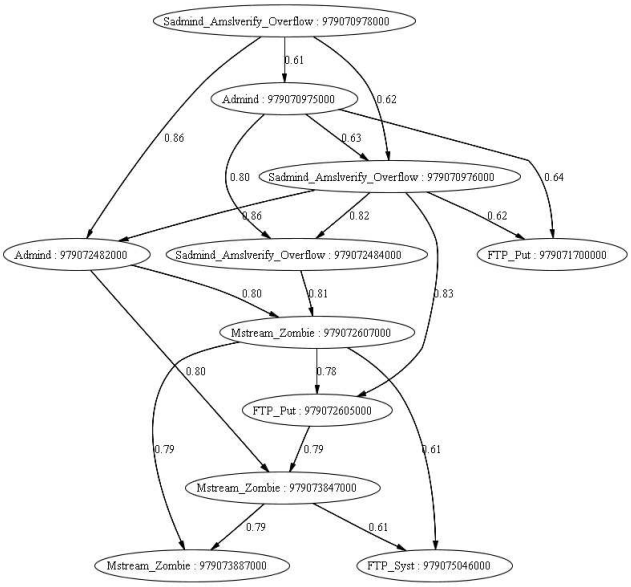

Figure 5: A hyper-alert graph for LLDOS2.0.2 scenario

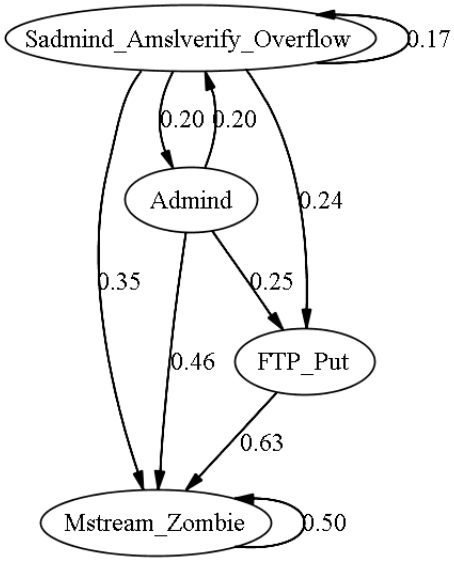

Figure 6: An attack graph for LLDOS2.0.2 extracted from ACM

Figure 6 is the attack graph extracted from LLDOS2.0.2. It is easy to find out that it has a similar pattern with the one extracted from LLDOS1.0. In both cases, the attacker compromises several machines (in different ways) by exploiting the vulnerability of the Sadmind service, and installs DDoS daemons on these machines by using either *rsh* or *ftp*. We have already mentioned that the self-looped cycles represent repeating steps performed by the attacker, such as the *Sadmind_Amslverify_Overflow* attack in both scenarios in DARPA 2000 dateset. In our experiments, we identify that there are two types of repetition. The attacker can repeat one type of attack on a particular machine (e.g in LLDOS1.0 scenario), or repeat it on different machines in the network (e.g., in LLDOS2.0.2 scenario). Encoding this information in attack graphs provides good representation of the nature of attack strategies.

As previously mentioned, Ning et al. [8, 10] propose a correlation method to extract attack strategies from intrusion alerts, which is similar to ours. The experimental results on the DARPA 2000 dataset show that both approaches produce the similar graph representation for attack strategies. However, our approach is different than theirs in that it does not need to define a large number of rules in order to correlate the alerts. Moveover, the ACS is adaptive to the emerge of new attack patterns because new alerts are automatically added to the ACM, which allows the administrator to generate new attack graphs in order to study new attack strategies.

4 Conclusions and Future Work

4.1 Conclusions

Alert correlation is an important technique to aggregate the outputs from multiple IDSs, filter out spurious alerts, and provide a high-level view of the security state of the network. The research in this area is getting active recently because of the fact that generating huge number of alerts has become a major problem of traditional IDSs. A number of correlation approaches have been suggested. However, very few of them provide the capability of automatic extracting attack strategies from alerts. Most of them simply cluster the alerts into different groups.

Table 5: 17 types of alerts reported by RealSecure in LLDOS2.0.2

ID	Alert Name
1	RIPAdd
2	Email_Ehlo
3	Email_Almail_Overflow
4	FTP_Syst
5	FTP_Pass
6	FTP_User
7	Sadmind_Amslverify_Overflow
8	Admind
9	HTTP_Cisco_Catalyst_Exec
10	FTP_Put
11	Email_Turn
12	HTTP_Java
13	Mstream_Zombie
14	HTTP_ActiveX
15	Port_Scan
16	Stream_DoS
17	EventCollector_Info

This paper presents an alert correlation technique based on two neural network approaches: Multilayer Perceptron and Support Vector Machine. The goal of the proposed correlation technique is not only to group alerts together, but also to represent the correlated alerts in a way that they reflect the corresponding attack scenarios. The probabilistic outputs of MLP and SVM is proved to be helpful for constructing such attack scenarios. Both MLP and SVM have their own strengthes for alert correlation. When knowledge for assigning accurate probabilities to training data is available, MLP can produce more precise correlation probabilities. Labeling training patterns for SVM is much easier but the outputs are less accurate than the ones produced by MLP. Another advantage of using SVM is that its training speed is fast and it is possible to incrementally update it in a real time environment.

An alert correlation system is implemented to demonstrate the effectiveness of proposed technique. Experiments performed using the DARPA 2000 intrusion detection scenario specific datasets shows that the proposed technique can successfully correlate a large number intrusion alerts into scenarios. The hyper-alert graphs that are generated by correlation engine correctly reflect the multi-stage attacks in the dataset. The use of ACM is also proved to be effective for extracting high level attack strategies.

4.2 Future Work

Some extensions that can be make to this work are summarized in the following:

Identifying more features for correlation
In this work we only used six features for the alert correlation. MLP and SVM have the ability to handle high dimension input. So, the proposed alert correlation technique can be improve by introducing more features. Recent research shows that vulnerability and topology information about the protected network can be incorporated into the correlation process to produce more meaningful results. But the challenge is how to extract the features from those information and quantify and normalize them.

Real-time correlation
Correlation techniques will become more useful if they can be used in a real time environment and provide instant information about the attacks that are happening in the network. The alert correlation system that is developed in this paper can be extended and used for real-time correlation. However, some other issues such as performance and user interface need to be addressed before it becomes a viable alternative.

Recognizing the variations of attack strategies
Attackers often change attack patterns to achieve their goal. For example, they may use different attacks to gather information and compromise the target system to gain root access. Attack graphs provide useful information for analyst to study the variations of attack pattern. But automated analysis techniques on top of attack graphs can greatly reduce the system administrator/analyst's workload and therefore should be further investigated. Some graph theories can be used to analyze the similarity of attack graphs, and knowledge about the similarity of different type of alerts are also important for studying the variations of attack strategies.

Target recognition and risk assessment
Attack graphs enable network administrators to understand the strategies of attackers. In order to protect the network, the administrators normally will perform risk assessments against the attack strategies. By incorporating the vulnerability and topology information, it is also possible to identify the potential victims of a particular attack. How to combine all this information and come up with effective response plan may prove to be a very valuable research.

References

[1] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers,", *COLT*, pp. 144-152, 1992.

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

[3] F. Cuppens and R. Ortalo, "Lambda: A language to model a database for detection of attacks," in *Proceedings of Recent Advances in Intrusion Detection, 3rd International Symposium, (RAID 2000)*, LNCS 1907, pp. 197-216, Springer-Verlag, Toulouse, France, Oct. 2000.

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

[5] S. T. Eckmann, G. Vigna, and R. A. Kemmerer, "STATL: an attack language for state-based intrusion detection," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 71-103, 2002.

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, Prentice Hall, July 1998.

[7] MIT Lincoln Laboratory, *2000 Darpa Intrusion Detection Scenario Specific Data Sets*, 2000.

[8] P. Ning and Y. Cui, *An Intrusion Alert Correlator Based on Prerequisites of Intrusions*, Tech. Report TR-2002-01, North Carolina State University, USA, Jan. 2002.

[9] 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 communication security*, ACM Press, pp. 245-254, Washington D.C., USA, Nov. 2002.

[10] P. Ning and D. Xu, "Learning attack strategies from intrusion alerts," in *Proceedings of the 10th ACM conference on Computer and communication security*, ACM Press, pp. 200-209, Washington D.C., USA, Oct. 2003.

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

[12] North Carolina State University Cyber Defense Laboratory, *Tiaa: A Toolkit for Intrusion Alert Analysis*, http://discovery.csc.ncsu.edu/software/correlator /ver0.4/index.html, Apr. 10th, 2005, last accessed.

[13] J. Platt, *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods*, MIT Press, 1999.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs", in *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 273, Washington, DC, USA, 2002.

[15] K. L. S. J. Templeton, "A requires/provides model for computer attacks," in *Proceedings of the 2000 Workshop on New Security Paradigms*, pp. 31-38, Feb. 2001.

[16] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Proceedings of Recent Advances in Intrusion Detection, 4th International Symposium, (RAID 2001)*, LNCS 2212, pp. 54–68, Springer-Verlag, Davis, CA, USA, Oct. 2001.

Bin Zhu received his Master degree in Computer Science from University of New Brunswick (UNB) in year 2005. He received his B.Eng in Industrial automation from Beijing Institute of Clothing Technology in 1997. He is currently with the Network Security Lab / University of New Brunswick as security software developer since July 2005. Previously, he was with Privacy, Security & Trust (PST) research group in National Research Council (NRC) from July 2004 to June 2005. He also worked in the industrial as software developer for 5 years before his study at UNB. His research interests include Intrusion Detection, Machine Learning, Software Engineering and Data Visualization.

Ali A. Ghorbani (M'95) received his PhD (1995) and Master's (1979) from the University of New Brunswick, and the George Washington University, Washington D.C., USA, respectively. He was on the faculty of the Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran, from 1987 to 1998. Since 1999 he has been at the faculty of Computer Science, University of New Brunswick (UNB), Fredericton, Canada, where he is currently a Professor of Computer Science. He is also a member of the Privacy, Security and Trust (PST) team at the National Research Council (NRC) of Canada.

He has held a variety of positions in academia for the past 24 years. His research originated in software development, where he designed and developed a number of large-scale systems. His current research focus is Neural Networks, Web intelligence, agent systems, complex adaptive systems, and trust and security. He established the Intelligent and Adaptive Systems (IAS) and Network Security research groups in 2002 and 2004, respectively, at the faculty of Computer Science, UNB. The IAS group (http://ias.cs.unb.ca) pursues research on machine and statistical learning, data mining, intelligent agents and multiagent systems and Web intelligence. The NSL group (http://nsl.cs.unb.ca) is home to R&D in computer and network security.

He authored more than 100 research papers in journals and conference proceedings and has edited four volumes.

He is on the editorial board of the Computational Intelligence (CI), an international journal.

Dr. Ghorbani a member of ACM, IEEE Computer Society, and ANNS.