Anomaly Detection Using Layered Networks Based on Eigen Co-occurrence Matrix

Mizuki Oka[1], Yoshihiro Oyama[2,5], Hirotake Abe[3], and Kazuhiko Kato[4,5]

[1] Master's Program in Science and Engineering, University of Tsukuba
mizuki@osss.is.tsukuba.ac.jp
[2] Graduate School of Information Science and Technology, University of Tokyo
oyama@yl.is.s.u-tokyo.ac.jp
[3] Doctoral Program in Engineering, University of Tsukuba
habe@osss.is.tsukuba.ac.jp
[4] Graduate School of Systems and Information Engineering, University of Tsukuba
kato@is.tsukuba.ac.jp
[5] Japan Science and Technology Agency (JST) CREST

Abstract. Anomaly detection is a promising approach to detecting intruders masquerading as valid users (called masqueraders). It creates a user profile and labels any behavior that deviates from the profile as anomalous. In anomaly detection, a challenging task is modeling a user's dynamic behavior based on sequential data collected from computer systems. In this paper, we propose a novel method, called Eigen co-occurrence matrix (ECM), that models sequences such as UNIX commands and extracts their principal features. We applied the ECM method to a masquerade detection experiment with data from Schonlau et al. We report the results and compare them with results obtained from several conventional methods.

Keywords: Anomaly detection, User behavior, Co-occurrence matrix, PCA, Layered networks

1 Introduction

Detecting the presence of an intruder masquerading as a valid user is becoming a critical issue as security incidents become more common and more serious. Anomaly detection is a promising approach to detecting such intruders (masqueraders). It first creates a *profile* defining a normal user's behavior. It then measures the *similarity* of a current behavior with the created profile and notes any behavior that deviates from the profile. Various approaches for anomaly detection differ in how they create *profiles* and how they define *similarity*.

In most masquerade detection methods, a profile is created by modeling sequential data, such as the time of login, physical location of login, duration of user session, programs executed, names of files accessed, and user commands issued [1]. One of the challenging tasks in detecting masqueraders is to accurately model user behavior based on such sequential data. This is challenging because the nature of a user's behavior is dynamic and difficult to capture completely. In this paper, we propose a new method, called Eigen co-occurrence matrix (ECM), designed to model such dynamic user behavior.

E. Jonsson et al. (Eds.): RAID 2004, LNCS 3224, pp. 223–237, 2004.

One of the approaches to modeling user behavior is to convert a sequence of data into a *feature vector* by accumulating measures of either unary events (histogram) or n-connected events (n-grams) [2–4]. However, the former approach only considers the number of occurrences of observed events within a sequence, and thus sequential information will not be included in the resulting model. The latter approach considers n-connected neighboring events within a sequence. Neither of them considers any correlation between events that are not adjacent to each other.

Other approaches to modeling user behavior are based on converting a sequence into a network model. Such approaches include those based on an automaton [5–8], a Bayesian network [9], and an Hidden Markov Model (HMM) [10,11].

The nodes and arcs in an automaton can remember short- and long-range transition relations between events by constructing rules within a sequence of events. To construct an automaton, we thus require well-defined rules that can be transformed to a network. However, it is difficult to construct an automaton based on a set of user-generated sequences with various contexts, which does not have such well-defined rules. When an automaton can indeed be obtained, it is computationally expensive to learn on the automaton when a new sequence is added.

A node in a Bayesian network associates probabilities of the node being in a specific state given the states of its parents. The parent-child relationship between nodes in a Bayesian network indicates the direction of causality between the corresponding variables. That is, the variable represented by the child node is causally dependent on those represented by its parents. The topology of a Bayesian network must be predefined, however, and thus, the capability for modeling a sequence is dependent on the predefined topology.

An HMM can model a sequence by defining a network model that usually has a feed-forward characteristic. The network model is created by learning both the probability of each event emerging from each node and the probability of each transition between nodes by using a set of observed sequences. However, it is tough to build an adequate topology for an HMM by using ad hoc sequences generated by a user. As a result, the performance of a system based on an HMM varies depending on the topology and the parameter settings.

We argue that the dynamic behavior of a user appearing in a sequence can be captured by correlating not only connected events but also events that are not adjacent to each other while appearing within a certain distance (*non-connected* events). Based on this assumption, to model user behavior, the ECM method creates a so-called *co-occurrence matrix* by correlating an event in a sequence with any following events that appear within a certain distance. The ECM method then creates so-called *Eigen co-occurrence matrices*. The ECM method is inspired by the Eigenface technique, which is used to recognize humans facial images. In the Eigenface technique, the main idea is to decompose a facial image into a small set of characteristic feature images called eigenfaces, which may be thought of as the principal components of the original images. These eigenfaces are the orthogonal vectors of a linear space. A new facial image is then reconstructed by projecting onto the obtained space. In the ECM method, we consider the co-occurrence matrix and the Eigen co-occurrence matrices analogous to a facial image and the corresponding eigenfaces, respectively. The Eigen co-occurrence

matrices are characteristic feature sequences, and the characteristic features of a new sequence converted to a co-occurrence matrix are obtained by projecting it onto the space defined by the Eigen co-occurrence matrices.

In addition, the ECM method constructs the extracted features as a *layered* network. The distinct principal features of a co-occurrence matrix are presented as layers. The layered network enables us to perform detailed analysis of the extracted principal features of a sequence.

In summary, the ECM method has three main components: (1) modeling of the dynamic features of a sequence; (2) extraction of the principal features of the resulting model; and (3) automatic construction of a layered network from the extracted principal features.

The reminder of the paper is organized as follows. In Section 2, the ECM method is described in detail by using an example set of UNIX commands. Section 3 applies the ECM method to detect anomalous users in a dataset, describes our experimental results, and compares them with results obtained from several conventional methods. Section 4 analyzes the computational cost involved in the ECM method. Section 5 discusses possible detection improvements in using the ECM method. Section 6 gives our conclusions and describes our future work.

2 The Eigen Co-occurrence Matrix (ECM) Method

The purpose of this study is to distinguish malicious users from normal users. To do so, we first need to model a sequence of user commands and then apply a pattern classifica-

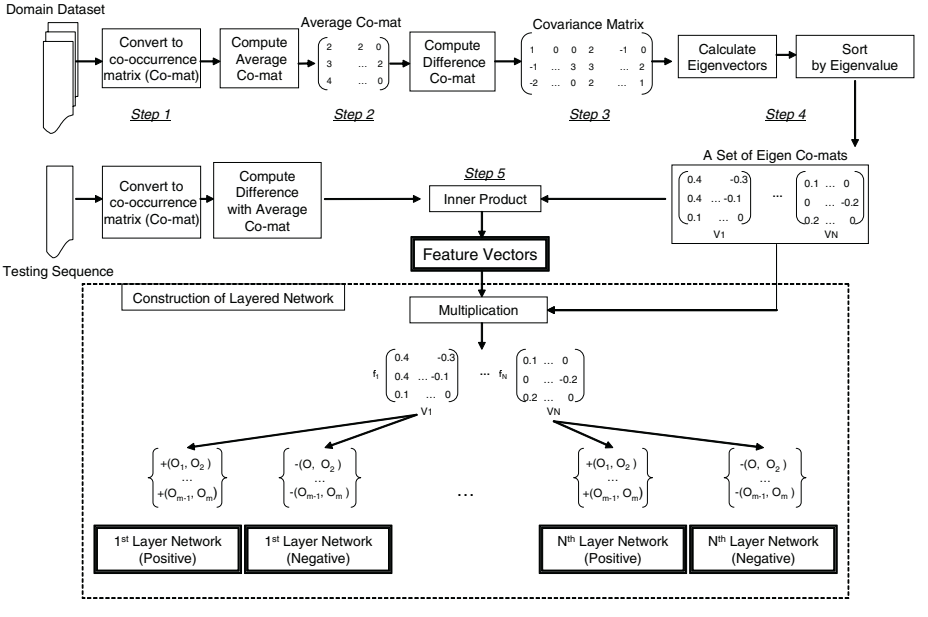

Fig. 1. Overall procedure of the ECM method

Table 1. Notation and terminology

l	length of an observation sequence
s	maximum distance over which correlations between events are considered (scope size)
O	set of observation events
m	number of events in O
D	set of sample observation sequences (*domain* dataset)
n	number of sample sequences in D
M	a co-occurrence matrix
V_i	ith Eigen co-occurrence matrix
F	a feature vector
f_i	ith component of F
N	dimension size of F
X_i	a matrix for producing ith *positive* network layer
Y_i	a matrix for producing ith *negative* network layer
h	threshold of elements in X_i (or Y_i) to construct a network layer
R	number of elements in f_iV_i for constructing the ith network layer
r	number of nodes in a subnetwork

$\overrightarrow{time}$

User1	cd	ls	less	ls	less	cd	ls	cd	cd	ls
User2	emacs	gcc	gdb	emacs	ls	gcc	gdb	ls	ls	emacs
User3	mkdir	cp	cd	ls	cp	ls	cp	cp	cp	cp

Fig. 2. Example dataset of UNIX commands

tion method. To accurately classify a sequence as normal or malicious, it is necessary to extract its significant characteristics (*principal features*) and, if necessary, convert the extracted features into a form suitable for detailed analysis. In this section, we explain how the ECM method models a sequence, how it obtains the principal features, and how it constructs a model for detailed analysis, namely, a network model. The overall procedure of the ECM method is illustrated in Figure 1 and the notation and terminology used in the ECM method are listed in Table 1.

In the following sections, we explain each procedure in the ECM method by using a simple example of UNIX command sequences. Figure 2 shows an example dataset of UNIX commands for three users, designated as User1, User2, and User3. Each user issued ten UNIX commands, which are shown truncated (without their arguments) in the interest of simplicity.

2.1 Modeling a Sequence

The ECM method models a sequence by correlating an event with any following events that appear within a certain distance. The strength of the correlation between two events is defined by (a) the distance between events and (b) the frequency of their occurrence. In other words, when the distance between two events is short, or when they appear more frequently, their correlation becomes stronger. To model such strength of corre-

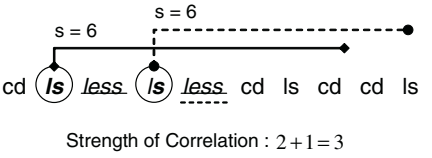

Fig. 3. Correlation between *ls* and *less* for User1

	cd	*ls*	*less*	*emacs*	*gcc*	*gdb*	*mkdir*	*cp*
cd	4	7	2	0	0	0	0	0
ls	7	5	*3*	0	0	0	0	0
less	6	4	1	0	0	0	0	0
emacs	0	0	0	0	0	0	0	0
gcc	0	0	0	0	0	0	0	0
gdb	0	0	0	0	0	0	0	0
mkdir	0	0	0	0	0	0	0	0
cp	0	0	0	0	0	0	0	0

Fig. 4. Co-occurrence matrix of User1

lation between events, we construct a so-called *co-occurrence matrix* by counting the occurrence of every event pair within a certain distance (*scope size*). Thus, the correlations of both connected and non-connected events are captured for every event pair and subsequently represented in the matrix.

We define M_X as the co-occurrence matrix of a sequence X ($= x_1, x_2, x_3, \ldots, x_l$) with length l. We define the unique events appearing in the sequence as a set of observation events, denoted as O ($= o_1, o_2, o_3, \ldots, o_m$). In the example dataset of Figure 2, O is `cd ls less emacs gcc gdb mkdir cp`. The correlation between the ith and jth events in M_X, o_i and o_j, is computed by counting the number of occurrences of the event-pair within a scope size of s. Here, we did not change the strength of the correlations between events depending on their distance, but instead used a constant value 1 for simplicity. Doing this for every event pair generates a matrix representing all of the respective occurrences. Each element in the matrix represents the perceived strength of correlation between two events. For example, as illustrated in Figure 3, the events *ls* and *less* are correlated with a strength of three when s and l are defined as 6 and 10, respectively. Figure 4 shows the matrix generated from the sequence of User1.

2.2 Extracting the Principal Features

As explained earlier, to distinguish a malicious user from a normal user, it is necessary to introduce a pattern classification method. Measuring the distance between co-occurrence matrices is considered the simplest pattern classification method. A co-occurrence matrix is highly dimensional, however, and to make an accurate comparison, it is necessary to extract the matrix's principal features.

The ECM method uses principal component analysis (PCA) to extract the principal features, so-called *feature vectors*. PCA transforms a number of correlated variables

into a smaller number of uncorrelated variables called principal components. It can thus reduce the dimensionality of the dataset while retaining most of the original variability within the data. The process for obtaining a feature vector is divided into the following five steps:

(Step 1) Take a domain dataset and convert its sequences to co-occurrence matrices: As a first step (*Step 1* in in Figure 1), we take a set of sample sequences, which we call a *domain* dataset and denote as D, and convert the sequences into corresponding co-occurrence matrices, $M_1, M_2, M_3, ..., M_n$, where n is the number of sample observation sequences and M is an $m \times m$ matrix (m: number of observation events). In the current example, the domain dataset consists of all the three users' sequences ($n = 3$), and M is an 8×8 matrix ($m = 8$).

(Step 2) Subtract the mean: We then take the set of co-occurrence matrices $M_1, M_2, M_3, ..., M_n$ and compute its mean co-occurrence matrix M_{mean} (*Step 2* in Figure 1). Here we introduce two different ways to compute M_{mean}. The first way is to compute it normally:

$$M_{mean} = \frac{1}{n} \sum_{k=1}^{n} M_k. \tag{1}$$

The second way is to compute M_{mean} by taking into account the fact that a co-occurrence matrix can be sparse. Let $m_{mean}(i, j)$ be the ith-row jth-column element of the mean co-occurrence matrix M_{mean}. We then compute $m_{mean}(i, j)$ by taking the sum of all the values in $m_1(i, j), m_2(i, j), m_3(i, j), \ldots, m_n(i, j)$ and dividing by the number of those values that are non-zero. In summary,

$$m_{mean}(i, j) = \frac{1}{K(i, j)} \sum_{k=1}^{n} m_k(i, j), \tag{2}$$

where $m_k(i, j)$ is the ith-row jth-column element of the kth co-occurrence matrix, and $K(i, j)$ and $\delta[x]$ are defined as

$$K(i, j) = \sum_{k=1}^{n} \delta[m_k(i, j)] \tag{3}$$

and

$$\delta[x] = \begin{cases} 1 & \text{if } x \text{ is not equal to zero} \\ 0 & \text{otherwise} \end{cases}, \tag{4}$$

respectively. The mean co-occurrence matrix M_{mean} is then subtracted from each event co-occurrence matrix,

$$A_k = M_k - M_{mean} \qquad \text{for } k = 1, 2, 3, \ldots, n, \tag{5}$$

where A_k is the kth co-occurrence matrix with the mean subtracted.

(Step 3) Calculate the covariance matrix: We then construct the covariance matrix as

$$P = \sum_{k=1}^{n} \hat{A_k} \hat{A_k}^T, \tag{6}$$

where $\hat{A_k}$ is created by taking each row in A_k and concatenating its elements into a single vector (*Step 3* in Figure 1). The dimension of $\hat{A_k}$ is $1 \times m^2$. In the example dataset, the dimension of $\hat{A_k}$ is 1×64.

The components of P, denoted by p_{ij}, represent the correlations between two event pairs q_i and q_j, such as the event pairs (*ls less*) and (*ls cd*) in the example dataset. An event pair q_i $(= o_x, o_y)$ can be obtained by

$$\begin{aligned} x &= \gamma[(i-1)/m] + 1 \\ y &= i - \gamma[(i-1)/m] \times m, \end{aligned} \tag{7}$$

where $\gamma[z]$ is the integer part of the value. The variance of a component indicates the spread of the component values around its mean value. If two components q_i and q_j are uncorrelated, their variance is zero. By definition, the covariance matrix is always symmetric.

(Step 4) Calculate the eigenvectors and eigenvalues of the covariance matrix: Since the covariance matrix P is symmetric (its dimension is $m^2 \times m^2$, or 64×64 in the example dataset), we can calculate an orthogonal basis by finding its eigenvalues and eigenvectors (*Step 4* in Figure 1). The eigenvector with the highest eigenvalue is the first principal component (the most characteristic feature) since it implies the highest variance, while the eigenvector with the second highest eigenvalue is the second principal component (the second most characteristic feature), and so forth. By ranking the eigenvectors in order of descending eigenvalues, namely $(v_1, v_2, ..., v_{m^2})$, we can create an ordered orthogonal basis according to significance. Since the eigenvectors belong to the same vector space as the co-occurrence matrices, v_i can be converted to an $m \times m$ matrix (8×8 in the example dataset). We call such a matrix an *Eigen co-occurrence matrix* and denote it as V_i.

Instead of using all the eigenvectors, we may represent a co-occurrence matrix by choosing N of the m^2 eigenvectors. This compresses the original co-occurrence matrix and simplifies its representation without losing much information. We define these N eigenvectors as the *co-occurrence matrix space*. Obviously, the larger N is, the higher the contribution rate of all the eigenvectors becomes. The contribution rate is defined as

$$\text{contribution rate} = \frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{m^2} \lambda_i}, \tag{8}$$

where λ_i denotes the ith largest eigenvalue.

(Step 5) Obtain a feature vector: We can obtain the *feature vector* of any co-occurrence matrix, M, by projecting it onto the defined co-occurrence matrix space (*Step 5* in Figure

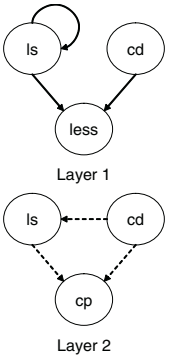

Fig. 5. Positive layered network for User1

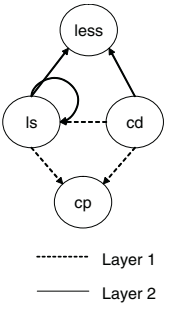

Fig. 6. Combined positive layered network of User1. The solid lines and dotted lines correspond to layer 1 and 2, respectively

1). The feature vector $F^T = [f_1, f_2, f_3, ..., f_N]$ of M is obtained by the dot product of vectors v_i and $\hat{A}$, where f_i is defined as

$$f_i = v_i^T \hat{A} \qquad \text{for } i = 1, 2, 3, \ldots, N \tag{9}$$

The Components $f_1, f_2, f_3, ..., f_N$ of F are the coordinates within the co-occurrence matrix space. Each component represents the contribution of each respective Eigen co-occurrence matrix. Any input sequence can be compressed from m^2 to N while maintaining a high level of variance.

2.3 Constructing a Layered Network

Once a feature vector F is obtained from a co-occurrence matrix, the ECM method converts it to a so-called *layered network* (shown as *construction of layered network* in Figure 1) . The ith layer of a network is constructed from the corresponding ith Eigen co-occurrence matrix V_i multiplied by the ith coordinate f_i of F. In other words, the ith *layer* of the network represents the ith principal feature of the original co-occurrence matrix.

The layered network can be obtained from equation (9). Recall that this equation for obtaining a component f_i (for $i = 1, 2, 3, \ldots, N$) of a feature vector is

$$f_i = v_i^T \hat{A} \qquad \text{for } i = 1, 2, 3, \ldots, N,$$

where $\hat{A}$ is the vector representation of $A = (M - M_{mean})$. We can obtain an approximation to the original co-occurrence matrix M' with the mean M_{mean} subtracted from the original co-occurrence matrix M by isolating $\hat{A}$ from equation (9). In summary,

$$(M - M_{mean}) \simeq \sum_{i=1}^{N} f_i V_i = M', \tag{10}$$

where $f_i V_i$ can be considered an adjacency matrix labeled by the set of observation events O. The ith network layer can be constructed by connecting the elements in the obtained matrix M'.

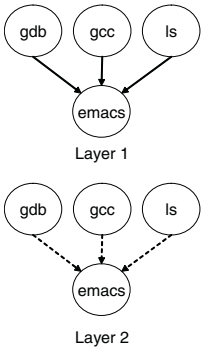

Fig. 7. Negative layered network for User1

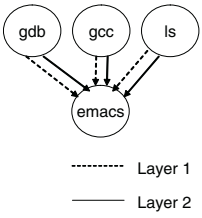

Fig. 8. Combined negative layered network for User1. The solid and dotted lines correspond to layers 1 and 2, respectively

Each layer of the network constructed by $f_i V_i$ (for $i = 1, 2, 3, \ldots, N$) provides the distinct characteristic patterns observed in the approximated co-occurrence matrix. We can also express such characteristics *in relation to the average co-occurrence matrix* by separating it as

$$\sum_{i=1}^{N} f_i V_i = \sum_{i=1}^{N} (X_i + Y_i) = \sum_{i=1}^{N} X_i + \sum_{i=1}^{N} Y_i \tag{11}$$

where X_i (or Y_i) denotes an adjacency matrix whose elements are determined by the corresponding positive (or negative) elements in $f_i V_i$. The matrix X_i (or Y_i) represents the principal characteristic of M' in terms of frequency (or rarity) in relation to the average co-occurrence matrix. We call the network obtained from X_i (or Y_i) a *positive* (or *negative*) network.

There may be elements in X_i (or Y_i) that are too small to serve as principal characteristics of M'. Thus, instead of using all the elements of X_i (or Y_i), we set a threshold h and choose elements that are larger (or smaller) than h (or $-h$) in order to construct the ith layer of the *positive* (or *negative*) network. Assigning a higher value to h reduces the number of nodes in the network and consequently creates a network with a different topology.

Figure 5 shows the first and second layers of the positive networks, obtained for User1 in the example dataset with h assigned to 0. We can combine these two layers to describe User1's overall patterns of principal frequent commands. The combined network is depicted in Figure 6, which indicates strong relations between the commands `ls`, `cd`, and `less`. This matches our human perception of the command sequence of User1 (i.e., `cd ls less ls less cd ls cd cd ls`).

Similarly, the first and second layers of the negative network and the combined network obtained for User1 are shown in Figures 7 and 8, respectively. These negative networks indicate the rarely observed command patterns in the command sequence of User1 relative to the average observed command patterns. We can observe strong correlations in the commands `gdb`, `gcc`, `ls`, and `emacs`. These relations did not appear in the command sequence.

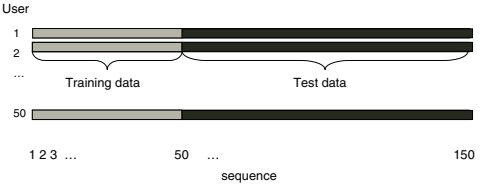

Fig. 9. Composition of the experimental dataset

3 Application of the ECM Method

3.1 Overview of the Experimental Data

We applied the ECM method to a dataset for masquerade detection provided by Schonlau et al. [12]. The dataset consists of 50 users' commands entered at a UNIX prompt, with 15,000 commands recorded for each user. Due to privacy arguments, the dataset includes no reporting of flags, aliases, arguments, or shell grammar. The users are designated as User 1, User 2, and so on. The first 5000 commands are entered by the legitimate user, and the masquerading commands are inserted in the remaining 10,000 commands. All the user sequences were decomposed into a sequence length of 100 commands ($l = 100$). Figure 9 illustrates the composition of the dataset.

3.2 Creation of a User Profiles (Offline)

For each user, we created a profile representing his normal behavior. Each decomposed sequence was converted into a co-occurrence matrix with a scope size of six ($s = 6$). We did not change the strength of the correlations between events on depending on their distance but instead used a constant value 1 for simplicity. We took all of the users' training dataset, consisting of 2500 (50 sequences $\times$ 50 users) decomposed sequences, and defined it as the *domain* dataset ($n = 2500$). The set of observation events ($O = o_1, o_2, o_3, \ldots, o_m$) was determined by the unique events appearing in the *domain* dataset, which accounted for 635 commands ($m = 635$). We took 50 Eigen co-occurrence matrices ($N = 50$), whose contribution rate was approximately 90%, and defined this as the co-occurrence matrix space.

The profile of a user was created by using his training dataset. We first converted all of his training sequences to co-occurrence matrices and obtained the corresponding feature vectors by projecting them onto the defined co-occurrence matrix space. Each feature vector was then used to reconstruct an approximated original co-occurrence matrix. This co-occurrence matrix was finally converted into a *positive* (or *negative*) layered network with a threshold of 0 ($h = 0$). We only used the *positive* layered network to define each user's profile.

3.3 Recognition of Anomalous Sequences (Online)

When a sequence seq_i of the User u was to be tested, we followed this procedure:

1. Construct a co-occurrence matrix from seq_i.
2. Project the obtained co-occurrence matrix on the co-occurrence matrix space and obtain its feature vector.

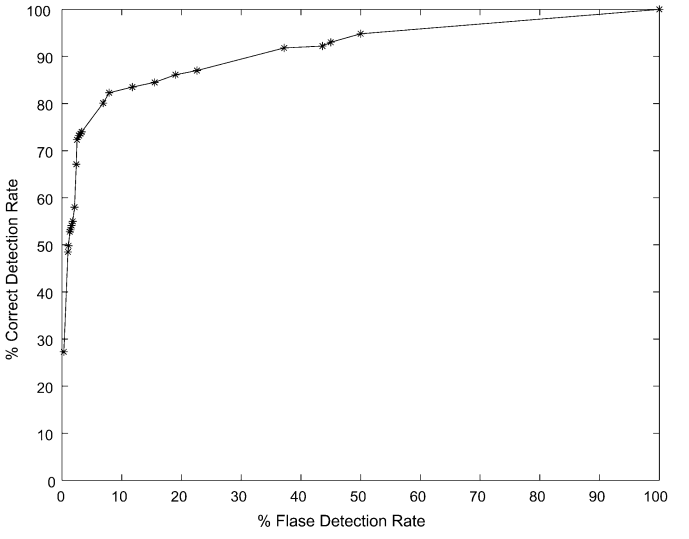

Fig. 10. ROC curves for the ECM method

3. Multiply the feature vector by the Eigen co-occurrence matrices to obtain a layered network.
4. Compare the layered network with the profile of User u.
5. Classify the testing sequence as anomalous or normal based on a threshold ϵ_u.

To classify a testing sequence seq_i as anomalous or normal, we computed the similarity between each network layer of seq_i and each networks layer in the user profile, where we chose the largest value as the similarity. If the computed similarity of seq_i was under a threshold ϵ_u for the User u, then the testing sequence was classified as anomalous; otherwise, it was classified as normal. We defined the similarity between the networks of two sequences, seq_i and seq_j, as,

$$Sim(seq_i, seq_j) = \sum_{k=1}^{N} \Gamma(T_k(i), T_k(j)), \tag{12}$$

where $T_k(i)$ is the obtained network at the kth layer of seq_i and $\Gamma(T_k(i), T_k(j))$ is the number of subnetworks that $T_k(i)$ and $T_k(j)$ have in common. We extracted the 30 largest values to form a network ($R = 30$) and employed 3 connected nodes as the unit of a subnetwork ($r = 3$).

3.4 Results

The results illustrate the trade-off between correct detection (true positives) and false detection (false positives). A receiver operation characteristic curve (ROC curve) is often used to represent this trade-off. The percentages of true positives and false positives are shown on the y-axis and x-axis of the ROC curve, respectively. Any increase in

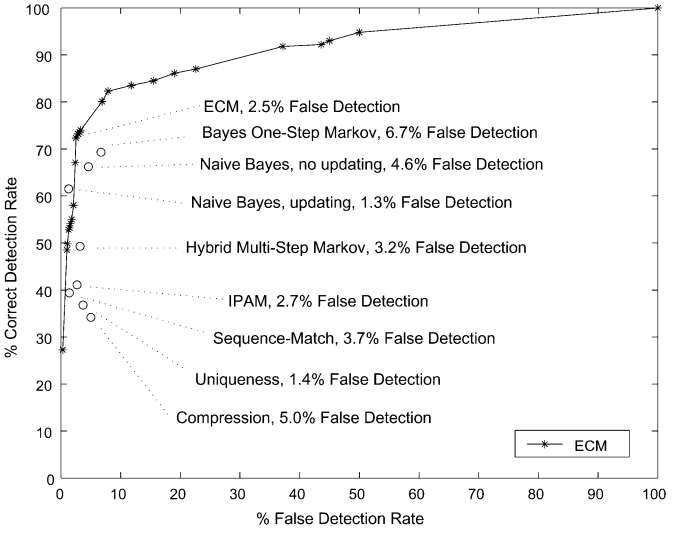

Fig. 11. ROC curve for the ECM method with the best results from other methods shown for comparison

the true positive rate will be accompanied by an increase in the false positive rate. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the results are, since they indicate high true positive rates and, correspondingly, low false positive rates.

Figure 10 shows the resulting ROC curve obtained from our experiment with the ECM method. We have plotted different correct detection rates and false detection rates by changing α in the expression:

$$\epsilon_u^{opt} + \alpha,$$

where ϵ_u^{opt} is the optimal threshold for User u. The optimal threshold ϵ_u^{opt} is defined by finding the largest correct detection rate with a false detection rate of less than $\beta\%$. We set β to 20 in this experiment and used the same values of ϵ_u throughout all the test sequences (no updating). As a result, the ECM method achieved a 72.3% correct detection rate with a 2.5% false detection rate.

Schonlau et al. [12] and Maxion et al. [13] have applied a number of masquerade detection techniques, including Bayes 1-Step Markov, Hybrid Multi-Step Markov, IPAM, Uniqueness, Sequence-Match, Compression, and Naive Bayes, to the same dataset used in this study. (See refs. [12] and [13] for detailed explanations of each technique.) Their results are shown in Figure 11 along with our results from the ECM method. As one can be seen from the data, the ECM method achieved one of the best scores among the various approaches.

4 Computational Cost

The ECM method has two computational phases, the offline and online phases. For the offline phase, the required computation processes are the following: transforming a set

Table 2. Changeable parameters in obtaining a feature vector

O	set of observation events.
l	length of sequence to be tested.
s	scope size
D	domain dataset.

Table 3. Changeable parameters in obtaining a layer of network

h	threshold of elements in X_i (or Y_i) for constructing a network.
R	number of elements in $f_i V_i$ for constructing the ith network layer
r	number of nodes in a subnetwork.

training sequences of length w to co-occurrence matrices, calculating the N eigenvectors of the covariance matrix, projecting co-occurrence matrices onto the co-occurrence matrix space to obtain feature vectors, constructing layered networks with R nodes in each layer, and generating a lookup table containing subnetworks with r connected nodes.

We used the Linux operating system (RedHat 9.0) for our experiments. We implemented the conversion of a sequence to a co-occurrence matrix in Java SDK 1.4.2 [14] and the remaining processes in Matlab Release 13 [15]. The hardware platform was a Dell Precision Workstation 650 (Intel(R) Xeon (TM) CPU 3.20GHz, 4GB main memory, 120GB hard disk). With this environment, for the online phase, it took 26.77 minutes to convert all the user training sequences ($l = 100$, $s = 6$) to the co-occurrence matrices (average of 642 ms each), 23.60 minutes to compute the eigenvectors ($N = 50$), 6.76 minutes to obtain all the feature vectors (average of 162 ms each), 677.1 minutes to construct all the layered networks with 30 nodes in each layer (average of 16.25 s for each feature vector), and 106.5 minutes to construct the lookup table ($r = 3$).

For the online phase, the required computations are the following: transforming a sequence to a co-occurrence matrix, projecting the obtained co-occurrence matrix to the set of N Eigen co-occurrence matrices, obtaining the feature vector of the co-occurrence matrix, constructing a layered network with R nodes, generating subnetworks with r connected nodes, and comparing the obtained layered network with the corresponding user profile. For one testing sequence, using the same environment described above, it took 642 ms to convert the sequence ($l = 100$, $s = 6$) to the co-occurrence matrix, 162 ms to obtain the feature vector ($N = 50$), 16.25 s to construct the layered network ($R = 30$), 2.60 s to generate the subnetworks ($r = 3$), and 2.48 s to compare the subnetworks with the profile. In total, it took 22.15 s to classify a testing sequence as normal or anomalous.

5 Discussion

As noted above, we have achieved better results than the conventional approaches by using the ECM method. Modeling a user's behavior is not a simple task, however, and we did not achieve very high accuracy with false positive rates near to zero. There is room to improve the performance by varying the parameters of the ECM method, as shown in Tables 2 and 3.

Table 2 lists the parameters that can be changed when computing a feature vector from a co-occurrence matrix. The parameter O determines the events for which correlations with other events are considered. If we took a larger number of events (i.e., UNIX commands), the accuracy of the results would become better but the computational cost cost would increase. Thus, the number of events represents a trade-off between accuracy and computational cost.

Changing the parameter l results in a different length of test sequence. Although we set l to 100 in our experiment in order to compare the results with those of conventional methods, it could be changed by using a time stamp, for example. The parameter s determines the distance over which correlations between events are considered. If we assigned a larger value to s, two events separated by a longer time interval could be correlated. In our experiment, we did not consider the time in determining the values of l and s, but instead utilized our heuristic approach, as the time was not included in the dataset. Moreover, we did not change the strength of the correlations between events depending on their distance for simplicity. Considering the aspect of dividing the number of occurrences by the distance between events, for example, would influence the results.

Choosing more sequences for the *domain* dataset D would result in extracting of more precise features from each sequence, as in the case of the Eigenface technique. This aspect could be used to update the profile of each user: updating the *domain* dataset would automatically update its extracted principal features, since they are obtained by using Eigen co-occurrence matrices.

Table 3 lists the parameters that can be changed in constructing a network layer from a co-occurrence matrix. In our experiment, we set $h = 0$ and chose the largest 30 elements ($R = 30$) to construct a positive network. Nevertheless, the optimal values of these parameters are open for discussion.

Additionally, the detection accuracy would be increased by computing the mean co-occurrence matrix M_0 by using equation (2) instead of equation (1), since each original co-occurrence matrix is sparse. Moreover, normalization of $\Gamma(T_k(i), T_k(j))$ by the number of arcs (or nodes) in both $T_k(i)$ and $T_k(j)$ may improve the accuracy: let $|T_k(i)|$ be the number of arcs (or nodes) in network $T_k(i)$. Then the normalized $\Gamma(T_k(i), T_k(j))$ would be simply obtained by $\Gamma(T_k(i), T_k(j))/(|T_k(i)||T_k(j)|)$.

6 Conclusions and Future Work

Modeling user behavior is a challenging task, as it changes dynamically over time and a user's complete behavior is difficult to define. We have proposed the ECM method to accurately model such user behavior. The ECM method is innovative in three aspects. First, it models the dynamic natures of users embedded in their event sequences. Second, it can discover principal patterns of statistical dominance. Finally, it can represent such discovered patterns via layered networks, with not only frequent (*positive*) properties but also rare (*negative*) properties, where each layer represents a distinct principal pattern.

Experiments on masquerade detection by using UNIX commands showed that the ECM method achieved better results, with a higher correct detection rate and a lower

false detection rate, than the results obtained with conventional approaches. This supports our assumption that not only connected events but also non-connected events within a certain scope size are correlated in a command sequence. It also shows that the principal features from the obtained model of a user behavior are successfully extracted by using PCA, and that detailed analysis by using layered networks can provide sufficient, useful features for classification.

Although we used the layered networks to classify test sequences as normal or malicious in our experiment, we should also investigate classification by using only the feature vectors. Furthermore, we need to conduct more experiments by varying the method's parameters, as described in Section 5, in order to improve the accuracy for masquerade detection. We must also try using various matching network algorithms to increase the accuracy.

References

1. Lunt, T.F.: A survey of intrusion detection techniques. Computers and Security **12** (1993) 405–418
2. Ye, N., Li, X., Chen, Q., Emran, S.M., Xu, M.: Probablistic Techniques for Intrusion Detection Based on Computer Audit Data. IEEE Transactions on Systems Man and Cybernetics, Part A (Systems & Humans) **31** (2001) 266–274
3. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion Detection using Sequences of System Calls. Journal of Computer Security **6** (1998) 151–180
4. Lee, W., Stolfo, S.J.: A framework for constructing features and models for intrusion detection systems. ACM Transactions on Information and System Security (TISSEC) **3** (2000) 227–261
5. Sekar, R., Bendre, M., Bollineni, P.: A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland (2001) 144–155
6. Wagner, D., Dean, D.: Intrusion Detection via Static Analysis. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland (2001) 156–168
7. Abe, H., Oyama, Y., Oka, M., Kato, K.: Optimization of Intrusion Detection System Based on Static Analyses (in Japanese). IPSJ Transactions on Advanced Computing Systems (2004)
8. Kosoresow, A.P., Hofmeyr, S.A.: A Shape of Self for UNIX Processes. IEEE Software **14** (1997) 35–42
9. DuMouchel, W.: Computer Intrusion Detection Based on Bayes Factors for Comparing Command Transition Probabilities. Technical Report TR91, National Institute of Statistical Sciences (NISS) (1999)
10. Jha, S., Tan., K.M.C., Maxion, R.A.: Markov Chains, Classifiers and Intrusion Detection. In: Proc. of 14th IEEE Computer Security Foundations Workshop. (2001) 206–219
11. Warrender, C., Forrest, S., Pearlmutter, B.A.: Detecting Intrusions Using System Calls: Alternative Data Models. In: IEEE Symposium on Security and Privacy. (1999) 133–145
12. Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades. In: Statistical Science. (2001) 16(1):58–74
13. Maxion, R.A., Townsend, T.N.: Masquerade Detection Using Truncated Command Lines. In: Prof. of the International Conference on Dependable Systems and Networks (DSN-02). (2002) 219–228
14. (Java) http://java.sun.com/
15. (Matlab) http://www.mathworks.com/