

available at [www.sciencedirect.com](http://www.sciencedirect.com)journal homepage: [www.elsevier.com/locate/cose](http://www.elsevier.com/locate/cose)


---



---

**Computers  
&  
Security**


---



---



# Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data<sup>☆</sup>

Wei Wang<sup>a,\*</sup>, Xiaohong Guan<sup>a,b</sup>, Xiangliang Zhang<sup>c</sup>, Liwei Yang<sup>a</sup>

<sup>a</sup>Center for Networked Systems and Information Security (CNSIS) and State Key Laboratory for Manufacturing Systems Engineering (SKLMSE), Xi'an Jiaotong University, Xi'an 710049, China

<sup>b</sup>Center for Intelligent and Networked Systems, Tsinghua University, Beijing 100084, China

<sup>c</sup>Department of Electronic Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

## ARTICLE INFO

### Article history:

Received 8 August 2005

Revised 3 May 2006

Accepted 31 May 2006

### Keywords:

Intrusion detection

Anomaly detection

Computer security

Hidden Markov models

Self organizing maps

Profiling

Computer audit data

## ABSTRACT

Intrusion detection is an important technique in the defense-in-depth network security framework. In recent years, it has been a widely studied topic in computer network security. In this paper, we present two methods, namely, the Hidden Markov Models (HMM) method and the Self Organizing Maps (SOM) method, to profile normal program behavior for anomaly intrusion detection based on computer audit data. The HMM method utilizes the transition property of events while SOM method relies on the frequency property of events. Two data sets, CERT synthetic Sendmail system call data collected in the University of New Mexico (UNM) and Live FTP system call data collected in the CNSIS lab of Xi'an Jiaotong University, were used to assess the two methods. Testing results show that the HMM method using the transition property of events produces good detection performance while high computational expense is required both for training and detection. The HMM method is better than other two methods reported previously in terms of detection accuracy for the same data set. The SOM method considering the frequency property of events, on the other hand, is suitable for real-time intrusion detection because of its capability of processing a large amount of data with low computational overhead.

© 2006 Elsevier Ltd. All rights reserved.

## 1. Introduction

With rapidly growing of unauthorized activities on the network, Intrusion Detection Systems (IDS) have become very important because peripheral protection mechanisms such as firewalls and various authentication techniques cannot provide complete protection against intrusions. Intrusion detection is a technology for detecting hostile attacks against

computer network systems, both from outside and inside. Intrusion detection together with firewall, authentication and other technologies, constitutes the defense-in-depth or layered network security framework for computer network systems.

In general, the techniques for intrusion detection fall into two major categories depending on the modeling methods used: misuse detection and anomaly detection. Misuse

<sup>☆</sup> The research presented in this paper was supported in part by the National Outstanding Young Investigator Grant (6970025), National Natural Science Foundation (60243001, 60574087) and 863 High Tech Development Plan (2001AA140213, 2003AA142060) of China.

\* Corresponding author.

E-mail addresses: [wei.wang.email@gmail.com](mailto:wei.wang.email@gmail.com) (W. Wang), [xhguan@sei.xjtu.edu.cn](mailto:xhguan@sei.xjtu.edu.cn) (X. Guan), [xiangliangzhang@gmail.com](mailto:xiangliangzhang@gmail.com) (X. Zhang), [lwyang@sei.xjtu.edu.cn](mailto:lwyang@sei.xjtu.edu.cn) (L. Yang).

0167-4048/\$ – see front matter © 2006 Elsevier Ltd. All rights reserved.

doi:10.1016/j.cose.2006.05.005

detection usually identifies abnormal behavior by matching it against pre-defined descriptions of attacks. This is effective to detect known attacks but generally is very difficult for detecting new attacks. Anomaly detection, on the other hand, profiles normal behavior and attempts to identify patterns of activities that deviate from the defined profile. It has the advantage of being able to detect new attacks. However, anomaly detection may have a high rate of false alarms because of the difficulties of normal behavior profiling. Given that our adversaries will always develop and launch new types of attacks in an attempt to attack computer network systems and to defeat our deployed intrusion prevention and detection systems, and that anomaly detection is the key to the defense against novel attacks, we must develop significantly better anomaly detection techniques (Lee and Xiang, 2001).

Anomaly detection has been an active research area for more than a decade since it was originally proposed by Denning (1987). Many types of data can be used for anomaly detection, such as Unix shell commands, audit events, keystroke records, system calls, and network packages. Early studies (Smaha, 1988; Lunt et al., 1992; Anderson et al., 1995) on anomaly detection mainly focus on learning normal system or user behavior from monitored system logs or accounting log data. Examples of the information derived from these logs are: CPU usage, time of login, duration of user session and names of files accessed. However, since user behavior changes frequently, obtaining complete descriptions of user behavior is often difficult and this may cause a high rate of false alarms during the procedure of intrusion detection.

In recent years, much research in anomaly detection focused on profiling program behavior based on system call data invoked during the execution of processes in a UNIX or a UNIX compatible operating system. Compared to user behavior, program behavior is more stable over time because the range of program behavior is more limited (Liao and Vemuri, 2002) based on limited types of system calls. Furthermore, it would be more difficult for attackers to break into a computer system without revealing their tracks in the execution logs (Liao and Vemuri, 2002). Therefore, profiling program behavior is more effective for intrusion detection and this property attracts many researchers.

Anomaly detection is a one-class pattern recognition problem in nature. It is always a challenge for an effective detection method to select features that best characterize program behavior patterns so that an abnormality can be clearly distinguished from normal activities. In general, there are two categories of attributes of activities in computer systems: transition property and frequency property of events. Both properties were widely used for intrusion detection.

The methods considering the transition property of events often use sliding windows to divide system call data into short sequences for data preparation. In 1996, Forrest et al. (1996) introduced a simple anomaly detection method based on monitoring the system calls issued by active, privileged processes. Each process is represented by the ordered list of system calls it issued. In sequence time-delay embedding (stide), the profile of normal behavior is built by enumerating all fixed lengths of unique, contiguous system calls that occur

in the training data, and any violation of the defined profile is considered as an anomaly (Forrest et al., 1996). This work was extended by various methods. Lee et al. (1997) established a more concise and effective anomaly detection model by using Ripper to mine normal and abnormal patterns from the system call sequences. Wespi et al. (2000) further developed Forrest's idea and proposed a variable-length approach. Warrender et al. (1999) proposed a Hidden Markov Models (HMM) based method for modeling and evaluating invisible events. This method was further studied by many other researchers (Yan et al., 2002; Yeung and Ding, 2003; Cho and Park, 2003; Wang et al., 2004a). Our research group employed rough set theory (Cai et al., 2003) and plan recognition method (Feng et al., 2004) based on system call data for intrusion detection. Other methods or techniques, such as first-order Markov chain models (Ye et al., 2004), high-order Markov models (Ju and Vardi, 2001), also used the transition property of system call data for anomaly detection.

There are also many methods that consider the frequency property of system call data for anomaly detection. Yeung and Ding (2003) used information-theoretic measures for anomaly detection based on the frequency property of system call data. Liao and Vemuri (2002) used  $k$ -Nearest Neighbor ( $k$ -NN) classifier and Hu et al. (2003) applied robust Support Vector Machines (SVM) for anomaly detection based on system call data to model program behavior and classified each process as normal or abnormal when it terminated. Zhang and Shen (2005) modified the conventional SVM, robust SVM and one-class SVM, respectively, based on the idea from online SVM for intrusion detection also based on the frequency property of the system call data. Rawat et al. (2006) used Singular Value Decomposition (SVD) for intrusion detection based on the frequencies of the system call data. In our previous work, we also employed Non-negative Matrix Factorization (NMF) (Wang et al., 2004b) and Principal Component Analysis (PCA) (Wang et al., 2004c) to profile program behavior based on the frequency property of system call data for anomaly detection.

In general, intrusion detection methods based on the transition property of events model temporal variations that may be essential for classifying abnormal behavior from normal activities (Yeung and Ding, 2003). The methods based on the frequency property of events, on the other hand, do not model temporal variations. They focus on the distribution of the data. As Hidden Markov Models (HMM) is effective for modeling temporal variation, we use this method to profile program behavior based on the transition property of the system call data. Self Organizing Maps (SOM) is a widely used dimension reduction and classification method. We then use this method to process massive audit data streams considering the frequency property of the data. Extensive experiments are conducted to evaluate the relationships between intrusion detection performance and the properties of events utilized. Comparative experimental results can be used as references to select the best features for effective intrusion detection.

The remainder of this paper is organized as follows. Section 2 describes two data sets used in the experiments. Two intrusion detection methods, HMM method and SOM method are described in Sections 3 and 4 for profiling program behavior based on the data sets. Concluding remarks are then summarized in Section 5.

## 2. Data sets

System call is the interface for processes in user space to access the low-level functions or resources of a computer system. In general, executing a single program would produce several processes and each process would produce a number of system calls from the beginning of its execution to the end. In the experiments presented in this paper, we group the system calls which are invoked by the same process into one trace and classify whether the process behavior is normal or not.

To assess the proposed methods for anomaly detection, we used two sets of system call data in the experiments for profiling normal program behavior. One set is small and the other is large for representing different detection environments. The small set is from CERT synthetic Sendmail data collected at the University of New Mexico (UNM) by Forrest et al. (1996). CERT synthetic Sendmail data include two sets. A small data set was used in the experiments. There are 19,526 system calls in the normal data including 147 processes. Abnormal data were collected by conducting four syslog intrusions that include syslog-local-1, syslog-local-2, syslog-remote-1, and syslog-remote-2. The four syslog intrusions produced 23 processes. The data sets are available for downloading at <http://www.cs.unm.edu/~immsec> and the procedures for generating the data are also described on the website.

The large set was collected in the actual system in the Center for Networked Systems and Information Security (CNSIS) of Xi'an Jiaotong University. We collected live FTP system call sequences on a Red Hat Linux system with kernel 2.4.7-10, spanning a time period of about 2 weeks. The normal data are traces of system calls generated by authentic users in various conditions. Intrusion traces are associated with exploitations against a widely known Wu-Ftpd vulnerability, which allows remote attackers to execute arbitrary commands on the victim host (CERT Advisory, 2001). The Live FTP system call data include 95 normal processes and three intrusion processes. The statistics of the data sets used in the experiments are shown in Table 1.

## 3. Profiling program behavior based on Hidden Markov Models (HMM) using the transition property of system call data

### 3.1. Profiling program behavior based on HMM

HMMs are dynamic models that consider the transition property of events. An HMM describes a doubly stochastic

process. Each HMM contains a finite number of unobservable (or hidden) states. Transitions among the states are governed by a set of probabilities called transition probabilities. An HMM defines two concurrent stochastic processes: the sequence of the HMM states and a set of state output processes. There are three central issues in HMMs including the evaluation problem, the decoding problem, and the learning problem. Given an input sequence of system calls, an HMM can model this sequence by three parameters—state transition probability distribution  $A$ , observation symbol probability distribution  $B$  and initial state distribution  $\pi$  (Duda et al., 2004; Rabiner, 1989; Rabiner and Juang, 1986). Therefore, a sequence can be modeled as  $\lambda = (A, B, \pi)$  using its characteristic parameters. Other parameters except  $A, B, \pi$  used in HMMs are defined as follows:

- $T$  = length of the sequence of observations (training set);
- $N$  = number of states in the model (we either know or guess this number);
- $M$  = number of possible observations (from the training set);
- $S = \{s_1, s_2, \dots, s_N\}$  finite set of possible states;
- $V = \{v_1, v_2, \dots, v_M\}$  finite set of possible observations.

HMMs learning can be conducted by the Baum–Welch (BW) or forward-backward algorithm—an example of a generalized Expectation-Maximization (EM) algorithm (Duda et al., 2004).

Standard HMMs have a fixed number of states, so we must decide the size of the model  $N$  before training. Previous work (Warrender et al., 1999) indicated that a good choice for the application was to choose a number of states roughly corresponding to the number of distinct system calls used by the program. Therefore, choosing the number of the states depends on the data set used in the experiments.

$\xi_t(i, j)$  is defined as the probability being in state  $s_i$  at time  $t$  and the state  $s_j$  at time  $t + 1$  and it can be written as following equations (Duda et al., 2004; Rabiner, 1989; Rabiner and Juang, 1986):

$$\begin{aligned} \xi_t(i, j) &= P(i_t = s_i, i_{t+1} = s_j | O, \lambda) \\ &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \end{aligned} \quad (1)$$

Forward variable  $\alpha_t(i)$  is defined as the probability that the model is in state  $s_i$  at time  $t$  and has generated the target sequence up to step  $t$ . It can be written as  $\alpha_t(i) = P(O_1 \dots O_t, i_t = s_i | \lambda)$ . Backward variable  $\beta_t(j)$  is analogously defined to be the probability that the model is in state  $s_j$  at time  $t$  and will generate the remainder of the given target sequence.

**Table 1 – The statistics of the data sets used in the experiments**

Data set	Normal data			Abnormal data	
	Total number of system calls	Number of distinct system calls	Number of normal traces (processes)	Total number of system calls	Number of intrusion traces (processes)
Sendmail data	19,526	53	147	6504	23
Live FTP data	1,167,878	45	95	1682	3

$\gamma_t(i)$  is the probability with which it stays at state  $s_i$  at time  $t$ .

$$\begin{aligned} \gamma_t(i) &= P(i_t = s_i | O, \lambda) \\ &= \sum_{j=1}^N \xi_t(i, j) \end{aligned} \quad (2)$$

From the definition of  $\xi_t(i, j)$  and  $\gamma_t(i)$ , if we define  $E_{ij}$  as the expected number of transitions from state  $s_i$  to state  $s_j$ ,  $E_{if}$  as the expected number of transitions from state  $s_i$  and  $E_i$  as the expected number of transitions from state  $s_i$ , the following equations can then be satisfied:

$$E_{ij} = \sum_{t=1}^{T-1} \xi_t(i, j) \quad (3)$$

$$E_{if} = \sum_{t=1}^{T-1} \gamma_t(i) \quad (4)$$

$$E_i = \sum_{t=1}^T \gamma_t(i) \quad (5)$$

We also define  $E_{jv_k}$  as the expected number of times in state  $s_j$  and observing symbol  $v_k$ . Given the above variables calculated, a new model  $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$  can be re-estimated by the following equations (Rabiner, 1989):

$$\bar{\pi}_i = \gamma_1(i) \quad (6)$$

$$\bar{a}_{ij} = \frac{E_{ij}}{E_{if}} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \quad (7)$$

$$\bar{b}_j(k) = \frac{E_{jv_k}}{E_j} = \frac{\sum_{t=1, \text{ s.t. } O_t=v_k}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad (8)$$

During the process of training the normal model, we start with initial model  $\lambda = (A, B, \pi)$  by setting rough and arbitrary  $\pi$ ,  $A$  and  $B$  and then calculate the re-estimated model  $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$  by Eqs.(6)–(8) based on the training set of system call sequences  $O$ . Clearly  $\bar{\lambda}$  is closer to actual model compared to  $\lambda$ , that is,  $P(O|\bar{\lambda}) > P(O|\lambda)$ . Replace  $\lambda$  by  $\bar{\lambda}$  and repeat until some convergence criterion is met (e.g., sufficiently small change in the estimated values of the parameters on subsequent iterations).

After the HMMs were learned on training set of system calls, normal program behavior is modeled by the parameters of the HMMs  $\lambda = (A, B, \pi)$ .

### 3.2. Anomaly detection scheme

Instead of considering both the unobservable state transition and output probability in the Warrender's methods (Warrender et al., 1999), our method only takes into account the output observable probability. Our method is also different from the HMM based intrusion detection method proposed by Yeung and Ding (2003). Our method defines the "mismatches" and considers all the short sequences of system calls in each trace for intrusion detection while their method only calculates the output possibility of the former sequences of each trace until an abnormality is found. Our method sounds more reasonable because if a single system call does not appear in the training set, the output probability of one sequence embedding the system call is zero. The trace embedding the sequence may be normal but in this

case their method would classify it as an intrusion immediately according to their detection scheme, possibly resulting in a false alarm.

Given a test trace of system calls (length  $S$ ) that were generated during the execution of a process from the beginning to the end, we use a sliding window of length  $L$  to move along the trace and get  $(S - L + 1)$  short sequences of system call  $X_i (1 \leq i \leq S - L + 1)$ .

Using the normal model  $\lambda = (A, B, \pi)$  built by the learning method described in above section, the probability that a given observation sequence  $X$  is generated from the model can be evaluated using the *forward algorithm* by forward variable  $\alpha_t(i)$  defined in the above section (Duda et al., 2004).

Summing up  $\alpha_t(i)$  for all  $i$  yields the value  $P(O|\lambda)$  that can be calculated by the following procedures (Duda et al., 2004; Rabiner, 1989):

Step 1. Initialization

$$\alpha_1(i) = \pi_i b_i(O_1) \quad (9)$$

Step 2. Induction

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}) \quad (10)$$

Step 3. Termination

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (11)$$

In the procedure of actual calculation, we use log-probabilities and compute  $\log P(O|\lambda)$  instead of  $P(O|\lambda)$  for the purpose of increasing the scale of the probability.

In general, a well-trained HMM gives sufficiently high likelihood only for sequences that correspond to normal behavior. Sequences correspond to abnormal behavior, on the other hand, should give a significantly lower likelihood values (Yeung and Ding, 2003). The HMM based anomaly detection method in this paper is heavily based on this property.

Given a predetermined threshold  $\varepsilon_1$  with which we compare the probability of a sequence  $X$  in a test trace, if the probability is below the threshold, the sequence  $X$  is flagged as a "mismatch". We sum up the mismatches and define the *anomaly index* as the ratio between the numbers of the mismatches and of all the short sequences in the test trace. The classification rule is thus assigned as following equation:

$$\begin{aligned} \text{anomaly index} &= \frac{\text{numbers of the mismatches}}{\text{numbers of all the sequences in the test trace}} \geq \varepsilon_2 \quad (12) \end{aligned}$$

If Eq. (12) is met, the trace (process) embedding the test sequences is considered as a possible intrusion. Otherwise it is considered as normal.

### 3.3. Testing results

For Sendmail data, 105 traces of the normal data were randomly selected for training and all the other data, 42 traces

of the normal data and 23 traces of intrusion data were used for detection. For live FTP data, five traces of the normal data were randomly selected for training and all the other data, 90 traces of normal data and three traces of intrusion data were used for detection. In the training set of the normal data, there are 53 distinct system calls in Sendmail data and 45 distinct system calls in live FTP data, respectively. We therefore used 53 states and 45 states of HMMs in the experiments for profiling the program behavior of Sendmail and FTP accordingly.

In our experiments, each trace of the system call data was first converted into short sequences of fixed lengths. Since there are only seven system calls in several traces of Sendmail data, we used window sizes 3 and 6, respectively, in the experiments for comparison of the testing results. The average anomaly indexes of the normal and intrusion traces of two data sets were calculated, respectively, and summarized in Table 2. For clear observation, the testing results of Sendmail data and Live FTP data with window size 6 are graphically shown in Figs. 1 and 2. The missing alarm rates and false alarm rates are also summarized in Tables 3 and 4.

In the experiments, we also measured the CPU time of training and detection based on the HMM method. The experiments were carried out on a computer with Pentium IV CPU 2.4 GHz and 512 MB memory and the testing results are shown in Table 5.

Forrest et al. (1996) used stide method and Lee et al. (1997) used Ripper for intrusion detection based on the same Sendmail data. They used a complete sequence of numerous system calls issued by programs for detection. In order to compare the detection results, we also performed additional experiments classifying the whole sequences of data issued by a program instead of detecting each trace of data issued by a process as either normal or abnormal. In the experiments, we used window sizes 3, 6, 7, 10, and 11, respectively, for comparison of the testing results. The anomaly indexes are shown in Table 6 together with the results obtained by Forest et al. and Lee et al. for comparison.

From Tables 2-6 and Figs. 1 and 2, it is observed that:

- (1) Good testing results are obtained by profiling program behavior based on the HMM method for anomaly detection. The anomaly indexes of abnormal sequences are significantly higher than those of the normal data for both of the two data sets and the HMM based method can detect

Table 2 – The average anomaly indexes of normal and abnormal traces of two sets of system call data			
System call sequences		Anomaly indexes (%)	
		Window size = 3	Window size = 6
Sendmail data	Syslog-remote-1	27.4143	40.0643
	Syslog-remote-2	32.9825	48.1475
	Syslog-local-1	28.2067	46.79
	Syslog-local-2	28.7650	47.445
	Normal	0.015	0
Live FTP data	Intrusion	11.42	16.25
	Normal	0.1021	0.1978

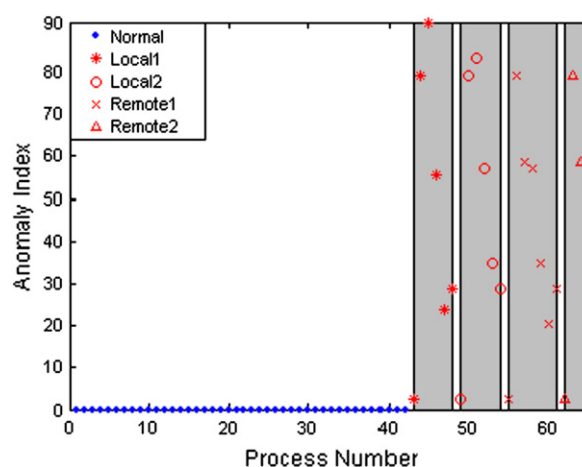


Fig. 1 – Testing results on Sendmail system call data with window size 6. The y-axis represents the anomaly index and x-axis represents the system call traces (processes). The gray shading indicates abnormal data.

all the intrusions without any false alarms. It is therefore an effective intrusion detection method with high detection accuracy.

- (2) Based on the HMM method, the anomaly indexes of abnormal sequences are much higher while the anomaly indexes of normal sequences are much lower than those results obtained by Forrest et al. (1996) and Lee et al. (1997) for all the window sizes. It shows that our method is better than the methods proposed by Forrest et al. and Lee et al. in terms of detection accuracy.
- (3) Different window sizes result in different anomaly indexes of each trace of data. For example, while the HMM method can detect all the intrusions with window size 6, one false alarm is raised with window size 3. The computational expense increases as well with large window sizes used. The performance of the intrusion detection is related to the window sizes and thus the choice of window size should be addressed in practical IDS.

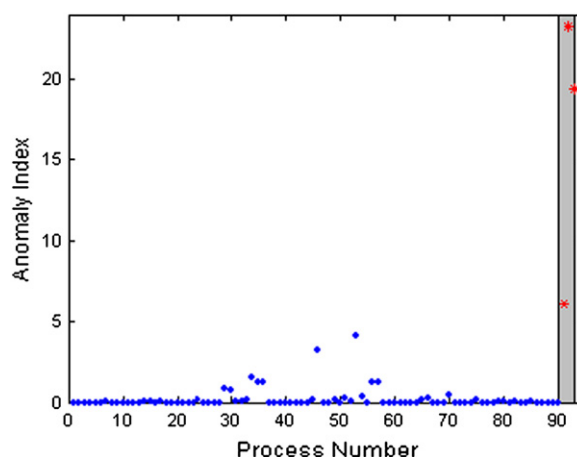


Fig. 2 – Testing results on Live FTP system call data with window size 6. The y-axis represents the anomaly index and x-axis represents the system call traces (processes). The gray shading indicates abnormal data.

**Table 3 – The detection rates and false alarm rates of Sendmail data**

Window size	Missing alarms (23 abnormal processes in total)	False alarms (42 normal processes in total)	Missing alarm rate (%)	False alarm rate (%)
3	0	1	0	2.4
6	0	0	0	0

(4) While the proposed method is effective for intrusion detection, training a normal behavioral model is computationally expensive. For example, training about 50,000 system calls requires more than 50 min. Detection is efficient once normal program behavior is profiled. However, it also requires much time for detection of large amounts of data. This is not adequate for real-time intrusion detection in training stage.

#### 4. Profiling program behavior based on self organizing maps using the frequency property of system call data

##### 4.1. Self Organizing Maps (SOM)

The Self Organizing Maps (SOM) was proposed by Kohonen (1990, 2001) as a projection method and a clustering algorithm. High dimensional data are mapped to a low dimensional space, typically to one- or two-dimensional lattice (Kohonen, 2001). In clustering, similar data vectors are mapped to nearby neurons and therefore the topology of the original data is preserved. These properties are very useful for data analysis and pattern classification.

The learning phase of SOM, which is trained to model the input space, has the following two characteristics:

- Competition: the learning is enforced by competition among the neurons: when an input  $x$  arrives, the neuron that is best able to represent it wins the competition and is allowed to learn it better.
- Cooperation: not only the winning neuron but also its neighbors on the lattice are allowed to learn. Neighboring neurons will gradually specialize to represent similar inputs, and the representations will become ordered on the map lattice. The cooperative learning is based on neighborhood functions.

The neurons represent the inputs with weight vectors  $w_i$ . One weight vector is associated with one neuron called unit in a more abstract setting. The unit, indexed with  $c$ , whose weight vector is nearest to the input  $x$  is the winner of the competition:

$$c(x) = \arg \min_i \|x(t) - w_i\| \quad (13)$$

Usually Euclidean metric is used for the measure of distance.

The winning unit and its neighbors adapt to represent the input even better by modifying their weight vectors towards the current input. The amount the units learn will be governed by a neighborhood kernel  $h$ , which is a decreasing function of the distance of the units from the winning unit on the map lattice. The neighborhood function  $h_{ci}(t)$  is specified as (Kohonen, 2001):

$$h_{ci}(t) = h(\|r_c - r_i\|; t) \quad (14)$$

where  $r_c$  and  $r_i$  represent the locations of winner  $c$  and neuron  $i$  on the map lattice.

Two neighborhood functions are usually used for data analysis: Bubble neighborhood and Gaussian neighborhood.

The Bubble neighborhood is defined as:

$$h_{ci}(t) = \begin{cases} \alpha(t) & \|r_c - r_i\| < R(t) \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

where  $R(t)$  is the neighborhood radius,  $\alpha(t)$  is the learning rate factor. The learning rate and neighborhood radius monotonically decrease with time  $t$ .

Gauss neighborhood function is defined as:

$$h_{ci}(t) = \begin{cases} \alpha(t) \exp\left(-\frac{\|r_c - r_i\|^2}{2R^2(t)}\right) & \|r_c - r_i\| < R(t) \\ 0 & \text{otherwise} \end{cases} \quad (16)$$

During the learning process at time  $t$ , the weight vectors are changed iteratively according to the following adaptation rule

$$w_i(t+1) = w_i(t) + h_{ci}(t)[x(t) - w_i(t)] \quad (17)$$

where  $x(t)$  is the input at time  $t$  and  $h_{ci}(t)$  is a neighborhood function centering at the output map coordinates of the winning neuron  $c$ .

The learning process, consisting of winner selection by Eq. (13) and adaptation of the weight vectors by Eq. (17), can be modeled with a neural network structure, in which the neurons are coupled by inhibitory connections.

##### 4.2. Profiling program behavior based on SOM

###### 4.2.1. Data preparation

During data preparation, we group the system calls that are invoked by the same process into one trace. The frequency of each individual system call in each trace is then calculated. Each trace of the data is then represented by a data vector whose entry is associated with the calculated frequency. For

**Table 4 – The detection rates and false alarm rates of Live FTP data**

Window size	Missing alarms (3 abnormal processes in total)	False alarms (93 normal processes in total)	Missing alarm rate (%)	False alarm rate (%)
3	0	0	0	0
6	0	0	0	0

**Table 5 – Training and detection time based on the HMM intrusion detection method**

Data set	Number of system calls for training	CPU time for training (s)	Number of system calls for detection	CPU Time for detection (s)
Sendmail data	13,885	1520.9	12,145	23.2 (window size = 3) 48 (window size = 6)
Live FTP data	57,814	3005.8	1,113,428	1420 (window size = 3) 2841 (window size = 6)

example, the system calls invoked by the process 3939 in CERT Sendmail data are shown below.

Process ID: 3939

```

105 104 104 106 105 104 104 106 105 104 104 106
5 4 5 5 0 40 41 105 104 104 106 61 5
85 50 27 18 50 27 2 3 2 3 3 2 3
3 2 3 3 2 3 3 2 3 3 2 3 2 3
12 112 19 128 9 9 5 9 9 5 112 4 5
5 5 5 3 4 3 6 9 4 6 9 10 94 95 4

```

In Sendmail data, each system call is represented by a number. The mapping between system call numbers and actual system call names is given in a separate file. For example, the number 105 represents system call “vtimes”; the number 5, represents system call “open”, and so on. Instead of dividing each trace of system call data into short sequences used by the HMM and most current intrusion detection methods (Forrest et al., 1996; Lee et al., 1997; Wespi et al., 2000; Warrender et al., 1999; Yan et al., 2002; Yeung and Ding, 2003; Cho and Park, 2003; Wang et al., 2004a; Cai et al., 2003; Feng et al., 2004), we use each trace of the data as observable as used in Liao and Vemuri (2002). In each trace of the data, the frequencies of individual system calls are calculated. For example, the frequency of number 105 in the process 3939 is 0.056; the frequency of number 5 is 0.153. Each trace of system call data is thus transformed into a data column vector as data input for anomaly detection.

Suppose there are  $m$  traces in the observation system call data, and there are a total of  $n$  distinct system calls in the data set, the observed data can be expressed by  $m$  vectors with each vector containing  $n$  distinct observations. An

$n \times m$  matrix  $X$ , where each element  $X_{ji}$  stands for the frequency of the  $j$ -th distinct system call occurring in the  $i$ -th trace, is then constructed. A matrix representing a data set is shown below as an example:

Distinct system call	Trace1	Trace2	Trace $m-1$	Trace $m$
1	0.051	0.055	...	0.049 0.051
2	0.122	0.125	...	0.119 0.115
	⋮	⋮	⋮	⋮
155	0.101	0.101	...	0.103 0.102
167	0.03	0.03	...	0.02 0.03

#### 4.2.2. Profiling program behavior based on SOM

The number of neurons and of the SOM dimensions is decided firstly for the learning phase of SOM. Usually, the number of neurons is chosen as the extraction of root of training vectors' number. The ratio between the SOM dimensions is chosen based on the ratio of the two largest eigenvalues that are calculated based on the correlation matrix of the training set.

SOM training is done in two phases. The first is the ordering phase during which the weight vectors of the map units are ordered. During the second phase, the values of the weight vectors are fine-tuned. In the first phase, the weight vectors of the map are initialized to tentative values. The lattice type of the map and the neighborhood function used in the training procedures are also defined in the initialization. In the experiments, the map is initialized with random numbers. The lattice type is selected to be hexagonal and the neighborhood function type is the Bubble function defined in Eq. (15).

**Table 6 – The anomaly indexes of the sequences of Sendmail system call data with different window sizes compared to the results obtained by Forrest et al. and Lee et al.**

System call sequences	Anomaly Index (%)						
	Forrest et al.		Lee et al.		The proposed method based on HMM		
	Window size	Window size	Window size	Window size	Window size	Window size	Window size
	11	7	11	10	7	6	3
Syslog-remote-1	5.1	11.5	50.89	49.95	46.9	45.2	30.34
Syslog-remote-2	1.7	8.4	50.81	49.94	47.25	45.41	29.72
Syslog-local-1	4.0	6.1	47.61	46.58	43.25	41.36	25.43
Syslog-local-2	5.3	8.0	49.62	48.56	45.09	43.21	26.78
Normal	0	0.6	0	0	0.55	0	0

**Table 7 – The value of the mean  $\mu$ , the standard deviation  $\sigma$ , and the threshold  $(\mu + \sigma)$  of the distances between each winning neuron and its associated training data vectors based on Sendmail data**

Neuron	1	2	3	4	5	6	7	8	9	10
$\mu$	$2.2888 \times 10^{-16}$	–	–	0.0195	0.0011	0.0211	–	0.0481	–	0.0045
$\sigma$	$1.5007 \times 10^{-31}$	–	–	0.0123	0.0004	0.0095	–	0.0287	–	0.0103
$(\mu + \sigma)$	$2.2888 \times 10^{-16}$	–	–	0.0318	0.0015	0.0306	–	0.0768	–	0.0148

Note that neurons 2, 3, 7, 9 have never won the competition during the learning phase.

During the second phase, the weight vectors in each unit converge to their ‘correct’ values. There are two methods to terminate the training process. One method is to check the convergence by a threshold and the other method is to define a number of iterations. Usually, the number of iterations is minimally set to be 500 times the number of neurons defined in the network (Kohonen, 2001). In the experiments, different numbers of iterations are tried and the testing results remain similar when the number exceeds a certain one. We then choose the optimal number for training process according to the actual data sets.

In the experiments, we use the SOM Toolbox from MATLAB package for training after defining the parameters. After the learning phase is completed, we validate the SOM. We group each winning neuron and its associated training data vectors. The distances between each winning neuron and all of its associated training data vectors are calculated and stored. We feed back the training data set to the SOM and validate if for each winner neuron at least 84.13% of the associated training data vectors is distributed around the winning neuron within one unit. If it does not meet, the training will be repeated. Here we assume that for each winning neuron the distances follow normal distribution and that the distances between each winning neuron and its associated training data vectors will be smaller than standard deviation plus the mean, that is, the distances will be smaller than one unit of standard deviation from the mean. For instance, if the mean and the standard deviation of the distances between a winning neuron and its associated training data vectors are denoted as  $\mu$  and  $\sigma$ , 84.13% of those distances of the training data vectors will be smaller than  $(\mu + \sigma)$ . Based on this property, normal program behavior is profiled (Ramadas et al., 2003). Here  $(\mu + \sigma)$  is a threshold determining missing alarm rates and false alarm rates.

The threshold can also be set as two units of standard deviation from the mean, denoted as  $(\mu + 2\sigma)$ . In this case, only 95.44% of those distances of the training data vectors are smaller than the threshold. The threshold is a trade-off among missing alarm rates and false alarm rates. It can be

made large enough to obtain low missing alarm rate but the false alarm rate could also be high. Although  $(\mu + \sigma)$  is shown to be effective for profiling program behavior, our experiments need to be conducted with various thresholds for our future work.

#### 4.3. Intrusion detection scheme

Given a test trace of system call data, we first transformed it into a data vector by the procedure of data preparation described above and then calculated the distances between the data vector and all the neurons. If the winning neuron of the test vector is not included in the winning neurons of the training samples, the test data are classified as abnormal immediately. If it is included, we calculate the distance between the test data vector and its winning neuron. If the distance follows the normal behavior profiled by SOM, that is, if the distance is smaller than  $(\mu + \sigma)$  as defined in above section, the data vector is then classified as normal. Otherwise it is considered as abnormal.

The Pseudo code of profiling model and detection scheme are described as follows:

##### Profiling normal program behavioral model:

Defined weight vectors of each neuron:

$$W = \{w_1, w_2, w_3, \dots, w_c, \dots, w_k, w_p\}$$

Winner weight vectors of the training data:

$$W_{normal} = \{w_k, w_l, w_m, \dots, w_q\}$$

Where  $q \leq p$

**For**  $w_i \in W_{normal}$ , **do**

    Calculate the distances:  $d(v_{ji}, w_i) = \|v_{ji} - w_i\|$

    Where  $v_{ji}$  is training data vector associated with the winner  $w_i$

    Calculate the mean distances  $\mu_i$  and the standard deviation of distances  $\sigma_i$

##### Intrusion Detection:

**For** each trace of test data that has been converted into vector  $x$ , **do**

**For**  $w_i \in W$  **do**

**Table 8 – The distances between several normal vectors of Sendmail data and their associated winning neurons**

Normal data of Sendmail	Trace 1	Trace 2	Trace 3	Trace 4	Trace 5	Trace 6	Trace 7	Trace 8
Distance	0.0198	$2.2888 \times 10^{-16}$	0.0355	0.0010	0.0247	0.0020	0.0294	0.0010
Winning neuron	4	1	8	5	4	10	6	5

**Table 9 – The distances between syslog-local-1 intrusion data vectors and their corresponding winning neurons**

Syslog-local-1 intrusion data	Trace 1	Trace 2	Trace 3	Trace 4	Trace 5	Trace 6
Distance	–	0.1875	–	0.2703	0.2399	0.2333
Winning neuron	2	8	7	8	8	8

Calculate the distances:  $d(x, w_i) = \|x - w_i\|$

**End for**

$d_x = \min d(x, w_i)$

$w_x = \arg \min d(x, w_i)$

Where  $w_x^i$  is the weight vector associated with the winner of the test vector  $x$

**If**  $w_x \notin W_{normal}$ , **then**

$x$  is abnormal

**Else if**

$d_x \leq (\mu_x + \sigma_x)$

$x$  is normal

**Else then**

$x$  is abnormal

**End For**

#### 4.4. Testing results

Two sets of system call data were used as well in the experiments. For Sendmail data, we used the same training sets and testing sets used by the HMM method for intrusion detection for the purpose of comparison. There are 105 traces in the training data and an SOM of dimensions  $1 \times 10$  is built accordingly. The number of iterations is set as 2000 during training of Sendmail data. For Live FTP data, 50 traces of the normal data were randomly selected for training and all the other data, 45 traces of normal data and three traces of intrusion data were used for detection. SOM dimensions are then built as  $1 \times 7$ . The number of iterations is set as 1000 during training of Live FTP data.

The value of the mean  $\mu$ , the standard deviation  $\sigma$ , and the threshold  $(\mu + \sigma)$  of the distances between each winning neuron and its associated training data vectors based on Sendmail data are summarized in Table 7. We then calculated the distances between the test data vectors and their associated winning neurons. The distances for several normal and abnormal traces of Sendmail data are shown as examples in Tables 8 and 9, respectively.

For Live FTP data, the value of the mean  $\mu$ , the standard deviation  $\sigma$ , and the threshold  $(\mu + \sigma)$  of the distances between each winning neuron and its associated training data vectors are summarized in Table 10. The distances of several normal and abnormal traces of Live FTP data are shown as examples in Table 11.

Based on the proposed intrusion detection scheme, we calculated the missing alarm rates and false alarm rates of the two data sets. The testing results of Sendmail data and Live FTP data are shown in Tables 12 and 13, respectively.

In the experiments, we also measured the CPU time for training and detection and the results are summarized in Table 14.

From Tables 7–14, it is observed that:

- (1) The distances for intrusion traces are larger than those for normal traces. This shows that the proposed method is effective for intrusion detection. For Sendmail data, the method can detect all the intrusions while giving three false alarms. For Live FTP data, the method can detect all the intrusions without any false alarms.
- (2) The SOM method is efficient for real-time intrusion detection. It only required about 7 s for detecting more than 1 million system calls. Training is also efficient relatively. Training about 60,000 system calls needed about 33 s.
- (3) The time for training and detection is related to the number of traces and iterations during learning phase. For example, training 105 traces of 13, 885 system calls with 2000 iterations required more time than training about 6000 system calls with 50 traces and 1000 iterations.

## 5. Concluding remarks

Intrusion detection is a one-class pattern recognition problem in nature. In general, there are two main issues for pattern recognition problems including feature selection and extraction and classification. It is a challenge to select features that are crucial for effective intrusion detection. There are two main properties of events that are commonly used for intrusion detection: the transition property and the frequency property. In this paper, we compared the HMM method using the transition properties of events and the SOM method using the frequency properties of events to profile program behavior for intrusion detection. Two data sets were used to assess the two methods: CERT synthetic Sendmail data collected in UNM by Forrest et al. and Live FTP data collected in the CNSIS lab of Xi'an Jiaotong University. The comparison of these two methods and concluding remarks are summarized as follows.

The testing results show that HMM is a better method than SOM in terms of detection accuracy on Sendmail data.

**Table 10 – The value of the mean  $\mu$ , the standard deviation  $\sigma$ , and the threshold  $(\mu + \sigma)$  of the distances between each winning neuron and its associated training data vectors based on Live FTP data**

Neuron	1	2	3	4	5	6	7
$\mu$	0.0205	0.0202	0.0363	0.0341	0.0532	0.0381	0.0319
$\sigma$	0.0140	0.0107	0.0177	0.0072	0.0332	0.0054	0.0066
$(\mu + \sigma)$	0.0345	0.0309	0.0540	0.0413	0.0864	0.0435	0.0385

**Table 11 – The distances between several normal and intrusion Live FTP data vectors and their associated winning neurons**

Live FTP data	Normal traces				Intrusion traces			
	Distance	0.0199	0.0282	0.0113	0.0397	0.0139	0.0874	0.0964
Winning neuron	1	3	2	5	3	4	7	7

However, the detection accuracy of HMM on Live FTP data is the same as that of SOM. The HMM method detected all intrusions without any false alarms on both the Sendmail data and the Live FTP data. On the other hand, while the SOM method successfully detected all the intrusions in the Live FTP data, it failed to detect 3 out of 42 intrusions in the Sendmail data. These results show that focusing on the transition property of events can achieve better detection accuracy than focusing on the frequency property. However, using the frequency property of events can also yield satisfactory results in intrusion detection.

The HMM method requires a considerable amount of time, not only to profile normal behavior but also for detection. It required more than 50 min for training about 50,000 system calls and required about 47 min for detecting about 1 million system calls. In contrast, the SOM method, operating on the same sets of data, required only about half of 1 min for training and only 7 s for detection. These results show that the SOM method, which relies on the frequency property of events for detection, has very low computational overhead compared to the HMM method utilizing the transition property of events. In real environments, a computer system in daily operation can produce massive amounts of audit data from various data sources. For example, during the process of collecting system calls of Sendmail on a host machine, only 112 messages produced a combined trace with the length of over 1.5 million system calls (Forrest et al., 1996). And in experiments carried out in the CNSIS lab of Xi'an Jiaotong University for collecting Live FTP data, more than 1 million system calls were generated in less than half an hour. Given these figures, the ability to processing large amounts of audit data at high speeds is essential for a practical IDS, one which can detect intrusions in real-time, before substantial damage has been done to the system. The testing results here show that the frequency based approach is better suited for real-time intrusion detection because of its low computational expense, not only for training, but also for detection.

Most current intrusion detection methods considering the transition property of events firstly divide sequences

of system calls by a fixed length of sliding window for data preparation. Detection performance is shown to be sensitive to window size. As window size increases, detection performance improves, but only at considerable computational expense. Moreover, the question of how to choose the window size has not been sufficiently addressed. The SOM method takes into account the frequency property of system calls. Processes are considered as observable so that the system call information contained in a process will not be separated factitiously by sliding windows. Moreover, it avoids deciding the size of sliding windows which are often chosen by experience. In the SOM method, each process is represented by a data vector, where each entry is the frequency of a distinct system call during the execution of a process. In this way, the anomaly intrusion detection problem is transformed into the simpler problem of classifying these vectors as normal or abnormal. The SOM method is thus an efficient intrusion detection method with good real-time performance.

Based on the comparative studies of two intrusion detection methods discussed above, it shows that utilizing the transition property of events can produce a good detection performance only at high computational expense. Relying on the frequency property of events, on the other hand, is suitable for real-time intrusion detection, providing adequate detection performance at low computational overhead. In practical use in anomaly detection, it is suggested to utilize the transition property of events only for a small data set or only if significant resources are available for a large data set to achieve high detection accuracy. Otherwise, considering the frequency property of events is recommended for processing large amount of audit data to achieve real-time intrusion detection.

There is still much that remains to explore, and our future work will focus on two aspects of the intrusion detection problem: (1) the integration of the frequency property with the transition information on sequences of system calls in order to achieve a compromise between detection performance and computational expense; (2) the further evaluation of the

**Table 12 – The missing alarm rates and false alarm rates for Sendmail data**

Missing alarms (23 abnormal processes in total)	False alarms (42 normal processes in total)	Missing alarm rate (%)	False alarm rate (%)
0	3	0	7.1

**Table 13 – The missing alarm rates and false alarm rates for Live FTP data**

Missing alarms (3 abnormal processes in total)	False alarms (45 normal processes in total)	Missing alarm rate (%)	False alarm rate (%)
0	0	0	0

**Table 14 – The training and detection time for the SOM based intrusion detection method**

Data set	Number of system calls for training	Number of iterations	CPU time for training (s)	Number of system calls for detection	CUP time for detection (s)
Sendmail data	13,885 (105 traces)	2000	144	12,145 (65 traces)	0.24
Live FTP data	57,814 (50 traces)	1000	33	1,113,428 (48 traces)	7.06

relative merits of the two properties of events with more extensive data.

## REFERENCES

- Anderson D, Frivold T, Valdes A. Next-generation intrusion detection expert system (NIDES): a summary. Technical Report SRI-CSL-95-07. Menlo Park, California: Computer Science Laboratory, SRI International; 1995.
- Cai Z, Guan X, Shao P, Peng Q, Sun G. A rough set theory based method for anomaly intrusion detection in computer networks. *Expert Systems* 2003;18(5):251–9.
- CERT Advisory CA-2001-07 file globbing vulnerabilities in various FTP servers. <<http://www.cert.org/advisories/CA-2001-07.html>>; 2001 [retrieved May 2001].
- Cho SB, Park HJ. Efficient anomaly detection by modeling privilege flows using hidden Markov model. *Computers & Security* 2003;22(1):45–55.
- Denning DE. An intrusion-detection model. *IEEE Transactions on Software Engineering* 1987;13(2):222–32.
- Duda RO, Hart PE, Stork DG. *Pattern classification*. 2nd ed. Beijing: China Machine Press; 2004.
- Feng L, Guan X, Guo S, Gao Y, Liu P. Predicting the intrusion intentions by observing system call sequences. *Computers & Security* 2004;23(5):241–52.
- Forrest S, Hofmeyr SA, Somayaji A, Longstaff TA. A sense of self for Unix processes. In: *Proceedings of the 1996 IEEE symposium on research in security and privacy*. Los Alamos, CA; 1996. p. 120–8.
- Hu W, Liao Y, Vemuri VR. Robust support vector machines for anomaly detection in computer security. In: *Proceeding of the 2003 international conference on machine learning and applications (ICMLA'03)*. Los Angeles, California; 2003.
- Ju WH, Vardi Y. A hybrid high-order Markov chain model for computer intrusion detection. *Journal of Computational and Graphical Statistics* 2001;10(2):277–95.
- Kohonen T. The self-organizing map. *Proceedings of the IEEE* 1990;78:1464–80.
- Kohonen T. *Self-organizing maps*. 3rd ed. Berlin: Springer; 2001.
- Lee W, Xiang D. Information-theoretic measures for anomaly detection. In: *Proceedings of the 2001 IEEE symposium on security and privacy*. Oakland, CA, USA; 2001. p. 130–43.
- Lee W, Stolfo S, Chan P. Learning patterns from Unix process execution traces for intrusion detection. *AAAI Workshop: AI approaches to fraud detection and risk management*; 1997.
- Liao YH, Vemuri VR. Use of *k*-nearest neighbor classifier for intrusion detection. *Computer & Security* 2002;21(5):439–48.
- Lunt T, Tamaru A, Gilham F, Jagannathan R, Jalali C, Neumann PG, et al. A real-time intrusion detection expert system (IDES)—final technical report. Technical report. Menlo Park, California: Computer Science Laboratory, SRI International; 1992.
- Rabiner LR. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceeding of the IEEE* 1989;77(2).
- Rabiner LR, Juang BH. An introduction to hidden Markov models. *IEEE ASSP Magazine*; 1986.
- Ramadas M, Ostermann S, Tjaden B. Detecting anomalous network traffic with self-organizing maps. In: *Proceedings of sixth international symposium on recent advances in intrusion detection (RAID 2003)*. Pittsburgh, Pennsylvania; 2003. p. 36–54.
- Rawat S, Gulati VP, Pujari AK. On the use of singular value decomposition for a fast intrusion detection system. *Electronic Notes in Theoretical Computer Science* 2006;142:215–28.
- Smaha SE. Haystack: an intrusion detection system. In: *Proceedings of the IEEE fourth aerospace computer security applications conference*; 1988.
- Wang W, Guan X, Zhang X. Modeling program behaviors by hidden Markov models for intrusion detection. In: *Proceedings of the third international conference on machine learning and cybernetics (ICMLC 2004)*; 2004a. p. 2830–5.
- Wang W, Guan X, Zhang X. Profiling program and user behaviors for anomaly intrusion detection based on non-negative matrix factorization. In: *Proceedings of 43rd IEEE conference on control and decision*, Atlantis, Paradise Island, Bahamas; 2004b. p. 99–104.
- Wang W, Guan X, Zhang X. A novel intrusion detection method based on principal component analysis in computer security. In: *Advances in neural networks-ISNN2004. International IEEE symposium on neural networks*, Dalian, China. Lecture notes in computer science (LNCS), No. 3174; 2004c. p. 657–62.
- Warrender C, Forrest S, Pearlmutter B. Detecting intrusions using system calls: alternative data models. In: *Proceedings of 1999 IEEE symposium on security and privacy*; 1999. p.133–45.
- Wespi A, Dacier M, Debar H. Intrusion detection using variable-length audit trail patterns. In: *Proceedings of the third international workshop on the recent advances in intrusion detection (RAID'2000)*, No. 1907 in LNCS; 2000.
- Yan Q, Xie W, Yan B, Song G. An anomaly intrusion detection method based on HMM. *Electronics Letters* 2002;38(13):663–4.
- Ye N, Zhang Y, Borror CM. Robustness of the Markov chain model for cyber attack detection. *IEEE Transactions on Reliability* 2004;53(1):116–21.
- Yeung DY, Ding Y. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition* 2003;36(1): 229–43.
- Zhang Z, Shen H. Application of online-training SVMs for real-time intrusion detection with different considerations. *Computer Communications* 2005;28:1428–42.
- Wei Wang** ([wei.wang.email@gmail.com](mailto:wei.wang.email@gmail.com)) received his B.S. degree in process equipment and control engineering and M.S. degree in mechanical and electronic engineering from Xi'an Shiyou University, Xi'an, China, in 1997 and 2000, respectively, and his Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, China in 2005. He is currently a postdoctoral fellow in Department of Information and Communication technology, University of Trento, Italy. His research interests currently focus on computer networked systems and computer network security.
- Xiaohong Guan** ([xhguan@tsinghua.edu.cn](mailto:xhguan@tsinghua.edu.cn)) received his B.S. and M.S. degrees in control engineering from Tsinghua University, Beijing, China, in 1982 and 1985, respectively, and

his Ph.D. degree in electrical engineering from the University of Connecticut in 1993. He was a senior consulting engineer with PG&E from 1993 to 1995. From 1985 to 1988 and since 1995 he has been with the Systems Engineering Institute, Xi'an Jiaotong University, Xi'an, China, and currently he is the Cheung Kong Professor of Systems Engineering and Director of the National Lab for Manufacturing Systems. He is also the Chair of Department of Automation and Director of the Center for Intelligent and Networked Systems, Tsinghua University, China. He visited the Division of Engineering and Applied Science, Harvard University from Jan. 1999 to Feb. 2000. His research interests include computer network security, wireless sensor networks and economics and security of complex networked systems.

**Xiangliang Zhang** ([xiangliangzhang@gmail.com](mailto:xiangliangzhang@gmail.com)) received her B.S. degree in information and communication engineering and her M.S. degree in Electronic Science and Technique from Xi'an Jiaotong University, Xi'an, China, in 2003 and in 2006, respectively. She was a internship student in department of Information and Communication Technology, University of Trento, Italy, from February 2006 to May 2005. Her research interests include machine learning, evolutionary computation and their applications, e.g., grid management and computer security.

**Liwei Yang** ([lwyang@sei.xjtu.edu.cn](mailto:lwyang@sei.xjtu.edu.cn)) received his B.S. degree in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 2002 and his research interests include intrusion detection and network security.