Front. Comput. Sci. China 2010, 4(4): 522–528
DOI 10.1007/s11704-010-0570-9

RESEARCH ARTICLE

Network intrusion detection based on system calls and data mining

Xinguang TIAN (✉)[1], Xueqi CHENG[1], Miyi DUAN[1,2], Rui LIAO[2], Hong CHEN[3], Xiaojuan CHEN[4]

1 Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
2 Institute of Computing Technology, Beijing Jiaotong University, Beijing 100029, China
3 Zhengzhou Information Science and Technology Institute, Zhengzhou 450004, China
4 College of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100037, China

Abstract Anomaly intrusion detection is currently an active research topic in the field of network security. This paper proposes a novel method for detecting anomalous program behavior, which is applicable to host-based intrusion detection systems monitoring system call activities. The method employs data mining techniques to model the normal behavior of a privileged program, and extracts normal system call sequences according to their supports and confidences in the training data. At the detection stage, a fixed-length sequence pattern matching algorithm is utilized to perform the comparison of the current behavior and historic normal behavior, which is less computationally expensive than the variable-length pattern matching algorithm proposed by Hofmeyr et al. At the detection stage, the temporal correlation of the audit data is taken into account, and two alternative schemes could be used to distinguish between normalities and intrusions. The method gives attention to both computational efficiency and detection accuracy, and is especially suitable for online detection. It has been applied to practical hosted-based intrusion detection systems, and has achieved high detection performance.

Keywords intrusion detection, data mining, system call, anomaly detection

Received December 12, 2008; accepted March 15, 2010

E-mail: tianxinguang@163.com

1 Introduction

Intrusion detection is one of the main directions of research in network security. There are two main techniques for network intrusion detection: misuse detection and anomaly detection [1,2]. Misuse detection systems model attacks as specific patterns, and use the patterns of known attacks to identify a matched activity as an attack instance. Misuse detection is not effective against unknown attacks [3]. Anomaly detection systems use established normal behavior profile of a subject, e.g., a user, a program, or a host machine, to identify any unacceptable deviation as the result of an attack [4,5]. The main advantage of anomaly detection lies in its ability to detect novel and unknown attacks, and it acts as the major direction of research in intrusion detection.

Many anomaly detection methods establishing normal behavior profiles for programs or users have been developed in recent years [1–9]. Yan et al., [5] introduced a method for detecting anomalous program behavior based on hidden Markov models (HMMs). The empirical evaluation described in Ref. [5] demonstrates that the method can achieve high detection accuracy, but it is computationally expensive. Lane et al., [6] presented a method for anomaly detection of user behavior on the basis of instance-based learning techniques. The method was tested on user shell command data, and the test results show that the profiled user can be accurately differentiated from alternative users when sufficient training data is available. Tian et al., [2] introduced an anomaly detection approach based on shell commands and homogeneous

Markov chains. The advantage of this approach is computational efficiency, but its detection accuracy is not very high [2]. Its true positive rate is about 80% when the false positive rate is 0.1%.

This paper presents an anomaly detection method to detect intrusions by monitoring privileged program behavior. Compared with the anomaly detection methods in Refs. [3,5,8], the method in this paper gives attention to both computational efficiency and detection accuracy, and is especially suitable for online detection. It has been applied to practical hosted-based intrusion detection systems, and achieved high detection performance.

We analyze privileged processes and system calls in Section 2 and give the training method to establish normal behavior profiles for programs in Section 3. We present the operation to perform detection in Section 4 and discuss our evaluation results in Section 5. Then we conclude the paper in Section 6.

2 Analysis of privileged processes and system calls

There are many different levels on which an anomaly intrusion detection system can monitor system behavior. Privileged processes are a good level to focus on because exploitation of vulnerabilities in privileged processes can give an intruder super-user status. Furthermore, privileged processes, especially those that listen to a particular port, constitute a natural boundary for a computer [1,5]. Compared to user behavior, process or program behavior is relatively stable over time. Users can carry out a wide variety of actions, whereas processes usually perform a specific, limited function [6,7]. Privileged processes are running programs that perform services, such as sending or receiving mail, which require access to system resources that are inaccessible to ordinary users. This is a simple definition, but the meaning of a privileged process varies from program to program. For some programs, a privileged process corresponds to a single task. For other programs, multiple privileged processes are required to complete a task [8].

In anomaly detection of program behavior, observable data (audit data) is required to distinguish between normal and intrusive behavior. In the work reported here, we choose system calls into the kernel of an operating system as the observable data. Each process is represented by its trace: the ordered list of system calls used by the process from the beginning to the end of its execution. This observable trace is much simpler than other proposals, especially those based on standard audit packages [10].

There are many ways in which system call data could be used to represent the normal behavior of a program and thus to detect anomalies, each of which involves building or training a model using traces of normal processes [11]. In this paper, a model based on data mining techniques is used to characterize the normal behavior of a program. In a traditional classification task, both positive and negative examples of the target concept are required for training [12]. We characterize program behavior purely from positive examples, and invoke the assumption that anything not seen in the historical data represents an anomaly; this is considered to be a reasonable assumption in many former studies on anomaly detection because divergence from past normal behavior is a practical and important indication of intrusion [11].

In the method presented in this paper, two different stages, the training stage and the detection stage, are needed to perform anomaly detection of program behavior. At the training stage, normal system call sequences are extracted in the training data according to their supports and confidences, and a dictionary of system call sequences is constructed to represent the normal behavior profile of the program. During the detection stage, a fixed-length sequence pattern matching algorithm is utilized to compare the current behavior with the historic normal behavior, and the two different schemes can be used to determine whether the monitored program's behavior is normal or anomalous while the temporal correlation of the audit data is taken into account.

3 Training

During training, we make the stationary assumption, that is, we assume that the action sequence of a privileged process is not related to the time of its execution. In fact, processes often have different distributions of system calls at the beginning of their execution than they do at the end. But the above assumption can greatly reduce the computational complexity. Compared to the computation of the HMM's parameters in [5], the training here is relatively simple. Furthermore, when new training data is acquired, we can firstly compute the supports and confidences of system call sequences in the new training data, and then calculate the general supports and confidences according to the proportion of the new data to all the training data, and thus update the system call sequences in the dictionary. The following steps are carried out to characterize the normal behavior profile of a process in the training stage.

3.1 Collecting training data

It is assumed here that the collected training data set consists of M traces. The M traces are denoted by R_1, R_2,..., R_M, where R_i is an ordered list of system calls issued by a normal process which was created by the program in history ($1 \leqslant i \leqslant M$). $R_i = \{s_1^i, s_2^i, ..., s_{r(i)}^i\}$, where $r(i)$ denotes the number of the system calls in R_i, and s_j^i denotes the jth system call in R_i. Let $r = r(1) + r(2) + \cdots + r(M)$. Here, r equals the total number of the system calls in the collected training data set.

3.2 Segmenting the training data into overlapping sequences

Then, segment each of the M traces R_1, R_2,..., R_M, into overlapping sequences of system calls, and thus generate a system call sequence list from each of the M traces.

Let S_1, S_2,..., S_M denote the M system call sequence lists generated from R_1, R_2,..., R_M respectively, where S_i is the system call sequence list generated from $R_i | 1 \leqslant i \leqslant M$. And $S_i = \{Seq_1^i, Seq_2^i, ..., Seq_{r(i)-1}^i\}$, where $Seq_j^i = (s_j^i, s_{j+1}^i)$. Seq_j^i denotes the jth system call sequence in S_i. Here $1 \leqslant j \leqslant r(i) - 1$. The length of each system call sequence in the sequence lists is 2. That is, each sequence contains two system calls.

3.3 Computing the supports and confidences of the system call sequences

Let s_1^*, s_2^*,..., s_D^* denote the unique system calls in the training data set R_1, R_2,..., R_M, where D equals the number of the unique system calls. Let $Seq = (s_i^*, s_j^*)$ denote the system call sequence in which the first system call is s_i^*, and the second system call is $s_j^* (1 \leqslant i,j \leqslant D)$. The support of $Seq = (s_i^*, s_j^*)$ in the training data set could be computed as follow:

$$\text{support}(Seq) = \text{number}(Seq)/(r - M), \tag{1}$$

where support(Seq) denotes the support of the sequence $Seq = (s_i^*, s_j^*)$ in the training data set; number(Seq) denotes the number of occurrences of $Seq = (s_i^*, s_j^*)$ in the M system call sequence lists generated from the training data set R_1, R_2,..., R_M; $r - M$ is the total number of the system call sequences in the M system call sequence lists. Support(Seq) is the occurrence probability of $Seq = (s_i^*, s_j^*)$.

Let $Seq^* = (s_i^*, s^*)$ denote system call sequences whose first system calls are s_i^*. In $Seq^* = (s_i^*, s^*)$, s^* denotes an unfixed (random) system call. The confidence of the sequence $Seq = (s_i^*, s_j^*)$ in the training data set can be computed as follow:

$$\text{confidence}(Seq) = \text{number}(Seq)/\text{number}(Seq^*), \tag{2}$$

where confidence(Seq) denotes the confidence of the sequence $Seq = (s_i^*, s_j^*)$ in the training data set; number(Seq^*) denotes the number of occurrences of $Seq^* = (s_i^*, s^*)$ in the M system call sequence lists generated from the training data set R_1, R_2,..., R_M. In many cases, number(Seq^*) equals the number of occurrences of the system call s_i^* in the training data set. confidence(Seq) is the probability of transition from s_i^* to s_j^*. According the above definitions, we can compute the supports and confidences of the system call sequences in the M system call sequence lists S_1, S_2,..., S_M.

3.4 Constructing a dictionary of system call sequences

Here, a dictionary of system call sequences needs to be constructed to characterize the normal behavior of the program. First, two parameters need to be set. They are the minimum support and the minimum confidence. Let *minsup* denote the minimum support, and Let *minconf* denote the minimum confidence.

Let K be the number of the system call sequences whose supports are above or equal to *minsup* in the M system call sequence lists S_1, S_2,..., S_M. The set of the K system call sequences, whose supports are above or equal to *minsup*, is denoted as $\Omega_s = \{Seq_1^*, Seq_2^*, ..., Seq_K^*\}$.

Let W be the number of the system call sequences whose confidences are above or equal to *minconf* in the M system call sequence lists. The set of the W system call sequences, whose confidences are above or equal to *minconf* is denoted as

$$\Omega_c = \{Seq_1^+, Seq_2^+, ..., Seq_W^+\}.$$

Let Ω_d denote the constructed dictionary of system call sequences which is used to characterize the normal behavior of the program. Ω_d could be Ω_s, Ω_c, $\Omega_s \cup \Omega_c$, or $\Omega_s \cap \Omega_c$. Here we choose the system call sequences in $\Omega_s \cap \Omega_c$ to construct the dictionary, i.e., $\Omega_\text{d} = \Omega_s \cap \Omega_c$.

4 Performing detection

During the detection stage, observable data needs to be acquired and processed. The dictionary of system call sequences is used to classify the behavior of the monitored processes created by the program as normal or anomalous. Detection is performed as follows.

4.1 Acquiring observable data

Let $\overline{R}=(\overline{s}_1,\overline{s}_2,...,\overline{s}_{\overline{r}})$ denote the observable data acquired during the detection stage. It is a trace generated by the monitored process, where $\overline{s}_i$ is the ith system call in $\overline{R}$, and $\overline{r}$ is the number of the system calls in $\overline{R}\,(1\leqslant i\leqslant\overline{r})$. In the case of online detection, the system calls in the trace are acquired in time order.

4.2 Segmenting the trace into overlapping sequences of system calls

The fundamental data units analyzed in the detection are system call sequences. To analyze the temporal behavior of the monitored process, the trace $\overline{R}=(\overline{s}_1,\overline{s}_2,...,\overline{s}_{\overline{r}})$ is segmented to form a stream of overlapping system call sequences which is denoted by $\overline{S}=(\overline{S}eq_1,\overline{S}eq_2,...,\overline{S}eq_{\overline{r}-1})$, where $\overline{S}eq_i$ denotes the ith system call sequence in the stream $(1\leqslant i\leqslant\overline{r}-1)$, and $\overline{S}eq_i=(\overline{s}_i,\overline{s}_{i+1})$. The length of each system call sequence in the sequence stream is 2, i.e., each sequence contains two system calls.

4.3 Sequence matching and normality analysis

For each system call sequence $\overline{S}eq_i$ in the sequence stream $\overline{S}=(\overline{S}eq_1,\overline{S}eq_2,...,\overline{S}eq_{\overline{r}-1})$, compare $\overline{S}eq_i$ with the sequences in the sequence dictionary Ω_d. If any sequence in Ω_d is identical to $\overline{S}eq_i$ (i.e., $\overline{S}eq_i\in\Omega_\mathrm{d}$), the system call sequence $\overline{S}eq_i$ will be considered normal, and the score of the normality measure corresponding to $\overline{S}eq_i$ will be assigned as $C_\mathrm{normal}(\overline{S}eq_i):=1$. Otherwise, $\overline{S}eq_i$ will be considered abnormal, and $C_\mathrm{normal}(\overline{S}eq_i):=0$. Thus, a normality measure stream $(C_\mathrm{normal}(\overline{S}eq_1),C_\mathrm{normal}(\overline{S}eq_2),...,C_\mathrm{normal}(\overline{S}eq_{\overline{r}-1}))$ can be obtained.

4.4 Smoothing the normality measure stream and classifying the process behavior

According to our study [1], intrusive traces generally resemble normal traces. All of the real intrusions we have studied produce abnormal system calls in temporally local clusters. On the basis of this fact, we apply windowed mean-value filters to smooth the normality measure stream $(C_\mathrm{normal}(\overline{S}eq_1),C_\mathrm{normal}(\overline{S}eq_2),...,C_\mathrm{normal}(\overline{S}eq_{\overline{r}-1}))$. Here, smoothing the normality measure stream and classifying the process behavior are performed jointly. Two alternative schemes are proposed for smoothing the stream and classifying the behavior.

In the first scheme, the normality measure stream is smoothed with a window of fixed length, e, and the process behavior is classified with a fixed threshold, λ. The normality measure stream $(C_\mathrm{normal}(\overline{S}eq_1),C_\mathrm{normal}(\overline{S}eq_2),...,C_\mathrm{normal}(\overline{S}eq_{\overline{r}-1}))$ is smoothed as follows:

$$D(k)=\frac{1}{e}\sum_{i=k-e+1}^{k}C_\mathrm{normal}(\overline{S}eq_i), \tag{3}$$

where $e\leqslant k\leqslant\overline{r}-1$. $D(k)$ is called a locality frame proportion, and it equals the proportion of normal sequences in the locality frame of the e system call sequences $\overline{S}eq_{k-e+1}$, $\overline{S}eq_{k-e+2}$, ..., $\overline{S}eq_k$. $D(k)$ gives us a signal for anomaly detection. The signal can indicate the number of abnormal sequences in the temporally local cluster. The mean-value filter can provide a locality frame proportion for each normality measure behind the eth measure in the input stream $(C_\mathrm{normal}(\overline{S}eq_1),C_\mathrm{normal}(\overline{S}eq_2),...,C_\mathrm{normal}(\overline{S}eq_{\overline{r}-1}))$. This is convenient for online detection.

Subsequently, the current behavior of the monitored process could be classified according to $D(k)$ and the threshold λ. If $D(k)\geqslant\lambda$, the current behavior of the monitored process will be considered normal. If $D(k)<\lambda$, the current behavior will be flagged as anomalous. Here the current behavior corresponds to the e system call sequences $\overline{S}eq_{k-e+1}$, $\overline{S}eq_{k-e+2}$, ..., $\overline{S}eq_k$, and also corresponds to the $e+1$ system calls $\overline{s}_{k-e+1}$, $\overline{s}_{k-e+2}$,..., $\overline{s}_{k+1}$ that are generated by the monitored process. The threshold λ on the locality frame proportions is a primary sensitivity parameter. A higher threshold tends to catch more intrusions but also gives more false positives. A lower threshold will result in fewer intrusions detected and fewer false positives.

In the second scheme, the normality measure stream is smoothed with windows of different lengths, and the process behavior is classified using more than one threshold. Here, we need to choose the number of the window lengths which is denoted by V, V window lengths which are denoted by $e(1), e(2), ..., e(V)$, V upper thresholds which are denoted by $u(1), u(2), ...,$ $u(V)$, and V lower thresholds which are denoted by $d(1),\ d(2), ..., d(V)$, where $e(1) < e(2) < ... < e(V)$, and $u(1) > u(2) > ... > u(V-1) > u(V) = d(V) > d(V-1) > ... > d(2) > d(1)$. $u(n)$ and $d(n)$ correspond to the nth window length $e(n)$. After computing the kth normality measure $C_{\text{normal}}(\overline{S}eq_k)$ in the stream $(C_{\text{normal}}(\overline{S}eq_1), C_{\text{normal}}(\overline{S}eq_2), ..., C_{\text{normal}}(\overline{S}eq_{\overline{r}-1}))$, the stream can be smoothed and the current behavior of the monitored process can be classified as follows.

Step 1 Define a variable n, and assign an initial value as $n := 1$.

Step 2 If $k \geqslant e(n)$, go to Step 3. If $k < e(n)$, do not execute the following steps, and do not classify the current behavior of the monitored process.

Step 3 Compute the mean value $D(k,n)$:

$$D(k,n) = \frac{1}{e(n)} \sum_{i=k-e(n)+1}^{k} C_{\text{normal}}(\overline{S}eq_i). \tag{4}$$

Step 4 If $D(k,n) > u(n)$, the current behavior of the monitored process is classified as normal (consistent).

Step 5 If $D(k,n) \leqslant d(n)$, the current behavior of the monitored process is classified as anomalous.

Step 6 If $d(n) < D(k,n) \leqslant u(n)$, then $n := n+1$, and go to Step 2.

Compared to the first scheme, the second scheme is computationally more expensive, but can also achieve higher detection accuracy.

It should be emphasized that the above operations including the acquirement of the observable data, trace segmentation, normality analysis, normality measure stream smoothing and process behavior classification are performed in parallel in the case of online detection. In the case of the first scheme, after the monitored process generates $e+1$ system calls, whenever it generates a new system call, a new system call sequence will be formed, and the normality measure of the sequence will be assigned. Subsequently a new locality frame proportion will be produced by the mean-value filter, and thus the current behavior of the monitored process can be classified according to the threshold.

Compared with the anomaly detection method based on hidden Markov models in Ref. [5], our method is less computationally expensive during the detection stage, and is more applicable to online detection. In practical detection, the computational efficiency can be improved by optimizing the matching of system calls. In addition, our method has high flexibility in the classification of process behavior; the threshold λ can be adjusted to control the true and false positive rates according to practical detection demands. In practical application, the parameters in the method including the minimum support, minimum confidence, window lengths and thresholds can be determined from training data by cross-validation which is popularly used in intrusion detection.

5 Experimental analysis

We tested the anomaly detection method described above using the experimental data set issued by the University of New Mexico [11] which is publicly available. The data set we chose includes 85 traces of the *sendmail* program, among which 79 traces are normal and 6 traces are intrusive that contains SCCP or DECODE attacks. These traces vary in their size and complexity. In the experiment, we used 13 normal traces as the training data to construct the dictionary of system call sequences. The 6 intrusive traces and 10 normal traces are used as the test data to evaluate the true and false positive rate respectively.

There are 289180 system call sequences in the 13 system call sequence lists generated from the 13 normal traces in the training. We set the minimum confidence as $minconf = 0$. During the detection stage, the first scheme is used to smooth the normality measure stream and classify the process behavior. When the minimum support *minsup* was set to 0, and the window length was set as $e = 7$, the curves of locality frame proportions output by the mean-value filter during the detection stage are shown in Fig. 1, where the continuous line represents the locality frame proportions corresponding to the normal data (10 normal traces), and the broken line represents the intrusive data (6 traces containing attacks). Figure 2 shows the curves of locality frame proportions when the minimum support *minsup* was set to 0, and the window length was set as $e = 20$.

Table 1 presents the experimental results when the minimum support *minsup* and the threshold λ change. The false positive rate is the percentage of decisions in which normal behavior was flagged as anomalous [1,11]. The false positive rate was computed differently from the true positive rate. To detect an intrusion (anomaly), we require only that the signal $D(k)$ falls below the preset threshold λ. However, making a single determination as to whether a

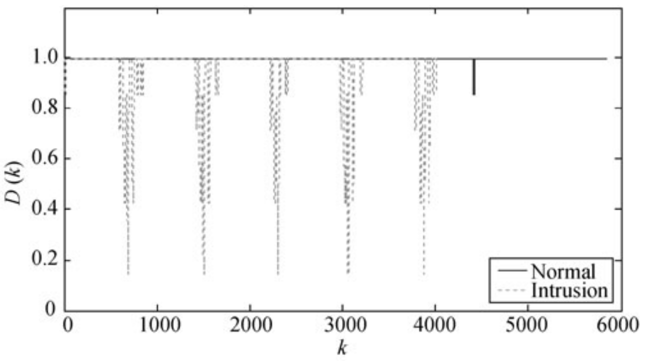

Fig. 1 Curves of locality frame proportions when $e = 7$

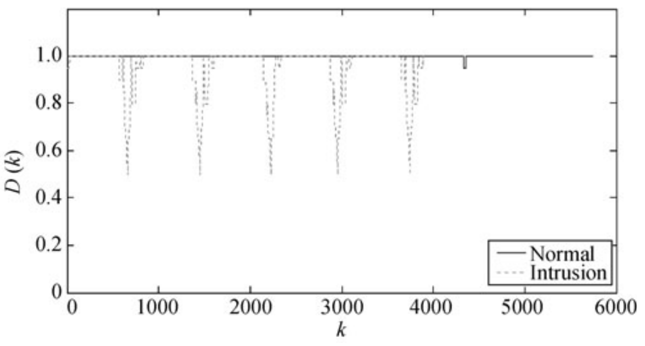

Fig. 2 Curves of locality frame proportions when $e = 20$

normal trace appears anomalous or not is insufficient, especially for very long traces. Each time that the behavior seen in the trace is flagged as anomalous should be counted separately [11]. Then, the false positive rate is the percentage of decisions in which normal behavior was flagged as anomalous.

According to the experimental results in Table 1, higher *minsup* will result in more true positives but also gives more false positives. In addition, we also tested the HMM-based method in Ref. [5] and the sequence-based method in Ref. [8]. using the same experimental data set. In Ref. [5], the number of the states of the hidden Markov model is set to 48, which is equal to the number of unique system calls in the training data. In Ref. [8], the length of system call sequences was set to 6, and the window length was set to 20. Table 2 presents the experimental results.

As is shown in Table 2, the true positive rate of the method in this paper is much higher than that of the method presented in Ref. [8], while the false positive rate of the our method is also lower. This means that our method can provide higher detection accuracy. The experimental duration in Table 2 is the time taken for training and testing, which reflects the computational costs of the different methods. We see from the results that the computational cost of our method is much smaller than that of the method in Ref. [5].

Furthermore, we changed the threshold λ gradually to get different true positive rates corresponding to different false positive rates while *minsup* is set to 0, and thus drew the ROC curve that reflects the relationship between the true and false positive rates. In the experiment of the methods in Refs. [5] and [8], we also drew the ROC curves by changing the corresponding thresholds. In the ROC curves, the x axis is false positive rate; the y axis is true positive rate; different points can be achieved by changing the thresholds λ. Figure 3 shows the ROC curves corresponding to the three methods. According to Fig. 3, the general detection accuracy of our method is much higher than that of Refs. [5] and [8].

6 Conclusion

In this paper we propose a novel intrusion detection method by establishing normal behavior profiles for privileged programs on the basis of the data mining technique, which has been applied to practical host-based intrusion detection systems. The method gives attention to both computational efficiency and detection accuracy, and is especially suitable for online detection. The results of our experience show our method can achieve higher detection accuracy than the existing typical methods such as HMM-based method in Ref. [5] and the sequence-based method in Ref. [8], while computational cost of our method is much smaller. The proposed method is also applicable to discrete behavior data from other subjects such as shell commands from a user and audit events from a host machine.

Table 1 Experimental results when *minsup* and λ change

minsup	Threshold	False positive rate/%	True positive rate/%	Number of sequences in Ω_d
2/289180	0.5	0	83	120
6/289180	0.2	0	83	99
10/289180	0.9	0.89	100	57
10/289180	0.2	0.02	100	57

Table 2 Experimental results of three methods

Performance index	False positive rate/%	True positive rate/%	Experimental duration/s
Method in Ref. [5]	0.02	83.33	49696
Method in Ref. [8]	0.15	66.67	8235
Our method	0	83.33	8072

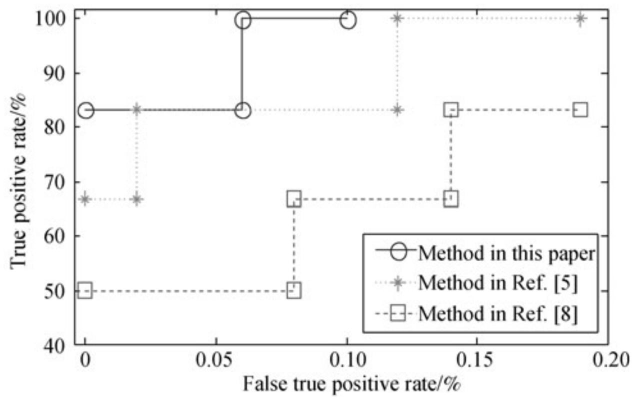

Fig. 3 ROC curves corresponding to the three methods

Acknowledgements This work was supported by the National High-Technology Research and Development Program of China (2006AA01Z452), and National Information Security 242 Program of China (2005C39).

References

1. Tian X G, Duan M Y, Sun C L, Li W F. Intrusion detection based on system calls and homogeneous Markov chains. Journal of Systems Engineering and Electronics, 2008, 19(3): 598–605
2. Tian X G, Duan M Y, Li W F, Sun C L. Anomaly detection of user behavior based on shell commands and homogeneous Markov chains. Chinese Journal of Electronics, 2008, 17(2): 231–236
3. Mukkamala S, Sung A H, Abraham A. Intrusion detection using an ensemble of intelligent paradigms. Journal of Network and Computer Applications, 2005, 28(2): 167–182
4. Oh S H, Lee W S. A clustering-based anomaly intrusion detector for a host computer. IEICE Transactions on Information and Systems. E (Norwalk, Conn.), 2004, 87-D(8): 2086–2094
5. Yan Q, Xie W X, Yang B, Song G. An anomaly intrusion detection method based on HMM. Electronics Letters, 2002, 38(13): 663–664
6. Lane T, Brodley C E. An empirical study of two approaches to sequence learning for anomaly detection. Machine Learning, 2003, 51(1): 73–107
7. Lee W, Dong X. Information-theoretic measures for anomaly detection. In: Proceedings of the 2001 IEEE Symposium on Security and Privacys, May 2001, Oakland, USA, IEEE Computer Society, 2001: 130–134
8. Hofmeyr S A, Forrest S, Somayaji A. Intrusion detection using sequences of system calls. Journal of Computer Security, 1999, 6 (3): 151–180
9. Ye N, Emran S M, Chen Q, Vilbert S. Multivariate statistical analysis of audit trails for host-based intrusion detection. IEEE Transactions on Computers, 2002, 51(7): 810–820
10. Verwoerd T, Hunt R. Intrusion detection techniques and approaches. Computer Communications, 2002, 25(15): 1356–1365
11. Warrender C, Forrest S, Pearlmutter B. Detecting intrusions using system calls: alternative data models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, May 1999, Berkely, USA, IEEE Computer Society, 1999: 133–145
12. Tian X G, Gao L Z, Sun C L, Duan M Y, Zhang E Y. A method for anomaly detection of user behaviors based on machine learning. The Journal of China Universities of Post and Telecommunications, 2006, 13(2): 61–65, 78