International Journal of Computational Intelligence Systems, Vol. 5, No. 1 (February, 2012), 30-38

Intrusion Detection Models Based on Data Mining

Guojun Mao *
School of Information, Central University of Finance & Economics, Xueyuannan 39, Beijing 100086, China
E-mail:maximmao@hotmail.com

Xindong Wu
Department of Computer Science, University of Vermont, Burlington, VT 05405, USA
E-mail:xwu@cems.uvm.edu

Xuxian Jiang
Department of Computer Science, North Carolina State University, NC 27695-8206, USA
E-mail:jiang@cs.ncsu.edu

Received 12 October 2011
Accepted 1 December 2011

Abstract

Computer intrusions are taking place everywhere, and have become a major concern for information security. Most intrusions to a computer system may result from illegitimate or irregular calls to the operating system, so analyzing the system-call sequences becomes an important and fundamental technique to detect potential intrusions. This paper proposes two models based on data mining technology, respectively called frequency patterns (*FP*) and tree patterns (*TP*) for intrusion detection. *FP* employs a typical method of sequential mining based on frequency analysis, and uses a short sequence model to find out quickly frequent sequential patterns in the training system-call sequences. *TP* makes use of the technique of tree pattern mining, and can get a quality profile from the training system-call sequences of a given system. Experimental results show that *FP* has good performances in training and detecting intrusions from short system-call sequences, and *TP* can achieve a high detection precision in handling long sequences.

Keywords: Intrusion detection, data mining, frequency pattern, tree pattern.

1. Introduction

Computer systems often suffer from a certain level of security flaws, which provide chances for intruders to attack computers for various purposes. Existing research studies have demonstrated that it is not sufficient to merely use some prevention measures such as user passwords and encryptions to solve this problem.[1-3]

Intrusion detection based on the observation that a system cannot be absolutely secure, therefore, becomes an active exploration topic in information security.

Two general intrusion detection techniques are *anomaly detection* and *misuse detection*. As far as anomaly detection is concerned, sufficient knowledge of the expected normal behaviors in a computer system must be provided in one way or another and such knowledge is used to detect possible attacks. Meanwhile, misuse detection, also known as signature detection, requires prior knowledge of possible attack patterns (or signatures) to match future intrusions. Both the technologies have their own advantages and disadvantages. Anomaly detection does not require prior knowledge of intrusions and can thus detect unknown intrusions that never successfully attacked the system before testing, but they may generate a large of alarms or false positives. The main advantage of misuse detection is that it can quickly determine known intrusive activities, but its biggest problem is that it can not discover any unknown attacks beyond defined signatures and so they may generate some false negatives which are riskier than false positives to harm the system.

DARPA[11,12] has conducted several evaluations on the state-of-the-art in intrusion detection systems. These results showed that the best intrusion detection systems had only detection rates below 70%, that is, the best intrusion detection systems can at most correctly identify 70% attack incidents in their computer systems. Most of the missed attacks were new types of intrusions. In fact, most of the research systems in the DARPA evaluations were leading commercial products that mainly employ misuse detection techniques. These systems were often not very effective for detecting new attacks, and the improvement is often too slow to keep up with the changes of sophisticated attackers who always develop new attack types. As a key technique to the defense against novel attacks, anomaly detection has received more attention in the research and development of intrusion detection.

1.1. *Related work*

In 1996, Forrest et al[1] introduced an intrusion detection method based on monitoring the system calls used by active processes. Through extracting privileged Unix processes, the system call sequences can be created, and the normal patterns can be mined by analyzing the correlations of system calls in a sequence.

In 1997, Helman et al[4] presented the problem of prioritizing actions under uncertainty using statistics and data mining techniques. In 1998, Hofmeyr et al[2] argued that detecting irregularities in the behaviors of privileged programs should regard the processes as a black box. If we regard the processes as a black box, we do not need any specialized knowledge of the internal functioning or the intended role of each process, and so we can infer the functioning and role information indirectly by observing the normal behavior of the process.

Over the past several years, many intrusion detection methods and models based on data mining techniques have been developed. In fact, if we consider intrusion detection as a data analysis process, anomaly detection is about finding the normal usage patterns, whereas misuse detection is about modeling the intrusion patterns from the audit data. With the rapid development in data mining, a wide variety of algorithms in data mining will be available to intrusion detection research.

In 1998, Lee et al[5] pointed out the significance of applying data mining techniques to intrusion detection. They think that many data mining methods, including of frequent pattern mining, classification technology, link analysis and sequence analysis, can help to create effective and efficient intrusion detection models. In 2000, Lee et al[9] gave a data mining framework for building intrusion detection models. To facilitate adaptability and extensibility, they propose the use of meta-learning as a means to construct a combined model that includes multiple data mining methods.

In 1999, Warrender et al[7] summarized and compared several methods and models to train system profiles, including simple enumeration, comparison of frequencies, rule induction and Hidden Markov Models (HMMs). In 2001, an intrusion detection model based on decision tree mining was proposed by Li and Ye[13]. As a result, the effectiveness of some data mining methods such as the association rule mining and decision tree classifying was proven. In 2001, Lee et al[10] discussed anomaly detection techniques based on information-theoretic measures. In 2001, Ye and Li[14] developed an anomaly detection method that builds the

normal profiles by averaging long-term normal activities in computer systems.

In 2002, Mohammed[8] established two intrusion detection models based on tree pattern mining called TreeMinerH and TreeMinerV. In 2004, Brugger[6] provided an overview of the various data mining methods to be employed for intrusion detection.

In recent years, there has been more focus on data mining based anomaly detection. In 2007, Patcha et al[16] gave an overview of anomaly detection techniques and thought that data mining would be one of most anticipated techniques for intrusion detection. In 1999, Goverdhan et al[15] proposed an unsupervised clustering scheme for isolating atypical behaviors that can reduce alarm rates. They tried to deduce intrusions from related atypical records beyond simple outlier detection. In 2010, Chandola et al[17] presented a called RBA (Reference based analysis) method, a novel data mining tool. Because RBA can transform any complex data type into a multivariate continuous representation, they demonstrate that applying the RBA framework in analyzing system call traces can help discover possible attacks.

1.2. *Our contributions*

In this paper, we suppose that our observation data can be collected from privileged programs of an operating system, and the behaviors of an application program can be modeled into a series of system-call sequences related to its dynamic processes.

In general, a system-call sequence is too long to be directly mined, and so partitioning a long sequence into shorter ones is a popular technique in handling very long sequences. There has been some evidence[5] that most very long or length-varied sequences of system calls can be characterized as a set of short sequences, and mining these short sequences can get close enough results to the knowledge hidden in the original long or length-varied sequences, and it is able to get a higher mining efficiency.

Based on the short sequence model, there are two methods called *FP* and *TP* to be designed for mining system call patterns in this paper.

The main contributions of this paper are as follows.

- It gives a formal definition of the short sequence model, which can be used as an observation data format to anomaly detection research as well as the in-memory data structure for training and detecting data objects;
- Two pattern structures for normal-behavior profiles are provided which respectively make use of the techniques of frequency mining and tree mining, and their training and detecting algorithms are also designed;
- Experiments on public datasets indicate that these methods for anomaly detection are promising and complementary to each other.

The rest of this paper is organized as follows. Section 2 describes the short sequence model and its related concepts. In Section 3, we present the *FP* model based on frequency analysis in sequences of system calls. In Section 4, the *TP* method based on tree pattern mining is introduced. Section 5 provides experimental results and their analyses, and a brief conclusions is drawn in Section 6.

2. The Short Sequence Model

Because any system damage is finally caused by running programs that execute system calls, any unusual behavior in computer systems can be detected by monitoring the system calls being executed by programs. Meanwhile, any attack can leave some traces that differ from normal system call sequences, and so analyzing the temporal ordering of these system calls can find out some potential anomalies.

In fact, with time increase, a nontrivial program may activate a large size of system call sequences, and it is impossible to directly analyze them in a high efficiency, but the locally short range ordering of system calls can reflect the program executing normality or abnormity as most attacks often take place in limited time interval. Therefore, if we transform a long system call trace into a series of local short sequences, we can efficiently build up a pattern database including of locally normal system calls according to normal running process traces of tested programs, and quickly discover abnormal call orders that an intrusion program is bringing into effect in the observed time interval.

Before we introduce our algorithm details, let us provide some concepts and terms used at first.

Definition 1 (A Process Trace). Given a process p, a p's trace t is a sequence of system calls conducted by p

from the beginning to the end of the process, denoted by $t = <c_1, c_2, ..., c_L>$.

The process traces of a system have three categories:
(1) *Normal* process traces, which have been proven that they do not cause any attacks to the system;
(2) *Anomaly* process traces, which can result in intrusions to the system;
(3) *Unproven* process traces, which are not sure yet to be normal or anomaly ones.

From this point of view, an anomaly detection method aims at training a normal pattern base from known normal process traces and detecting whether unproven process traces are normal or anomaly. If we have known some anomaly process traces, they can be used to evaluate the false negative rate of the anomaly detection method. Of course, the false positive rate of the anomaly detection method can be tested by some known normal process traces in the system.

However, most process traces in a system are very long and length-varied, and therefore we cannot directly use them to build the pattern base. A possible method is transforming an original long process trace into some short sequences of system calls with a fixed length. This transformation can be done by using sliding window techniques.

Definition 2 (A Short Sequence). Given the size of a sliding window K and a process trace $t = <c_1, c_2, ..., c_L>$, if $L > K$, the set of short sequences of t is created through sliding windows, which means t is transformed into a set of short sequences $(s_j)_{L-K+1}$, where $s_j = <c_j, c_{j+1}, ..., c_{j+K-1}>$ is a short sequence with length K (j=1, 2, ..., L-K+1).

Definition 3 (Short Sequence Model). Given a system and the size of a sliding window K, all short sequences related to all key processes in this system are denoted by $(s_{ij})_{M*N}$: each short sequence s_{ij} with size K is related to the ith process p_i(i =1, 2, …, M) and is the jth short sequence of p_i ($j \le N$).

Assuming there are M distinct processes and at most N distinct short sequences for every process in a system, $(s_{ij})_{M*N}$ can be used as a model to express system calls for training data. Also, $(s_{ij})_{M*N}$ can be a new in-memory data structure with a regular format instead of the original length-varied sequences of process traces collected from a system.

Thus, based on the above short sequence model, the problem of anomaly detection can be solved as follows:

(1) After collecting some original process traces that could be very long and length-varied, we can transform them into relevant short sequences (as shown in Definition 3). If these original process traces have been proven normal, their short sequences can used as training data to learn the normal pattern base.

(2) Designed and using some mining methods for anomaly detection, the normal pattern base can be learned from available training short sequences by these methods.

(3) For a new unproven process trace, it can be detected through the built pattern base.

3. Frequency Patterns for Anomaly Detection

In this section, we introduce a method for intrusion detection based on frequency mining technology. There might be many appropriate data mining methods to help build up normal pattern bases, but frequency patterns can get higher mining efficiency. Also, by defining a criterion based on expected frequencies, this method accords with the views that people distinguish normal or abnormal behaviors in life, that is, people always think what happens infrequently just means high possibility of anomaly events in life.

Based on frequency mining, we need to solve the following problems:
(1) What does the frequency of a sequence mean and how is a short sequence determined to be frequent or infrequent;
(2) How is a normal pattern base established based on frequency statistics;
(3) How is a new process trace detected to decide whether it includes anomaly behaviors or not.

Firstly, whether or not a system call is frequent in a short sequence must be associated with both the position of c in this short sequence and the process that this short sequence is collected from.

Definition 4 (A Frequent System Call). Let the number of the key processes be M, the size of sliding windows be K, and the user-specified threshold called *min-support* (percentage) be δ. A system call c is called a frequent system call in the lth position (l=1, 2, ..., K)

related to the ith process (i=1, 2, ..., M) if the ratio of the times that c occurs in the lth position of short sequences in the ith process to the number of all short sequences in the ith process is not less than δ.

Secondly, whether or not a short system call sequence is frequent is related to all system calls in this sequence with position sensitivity.

Definition 5 (A Frequent Short Sequence). Given the number of the key processes M, the size of sliding windows K, and the min-support δ, a short sequence s is frequent related to the ith (i=1, 2, ..., M) process if any call c_l in s (l=1, 2, ..., K) is frequent in the lth position related to the ith process.

Finally, a pattern base can be built through frequent short sequences from testing records. Theoretically, using frequency pattern mining, normal patterns of a process should be a set of all frequent short sequences related to this process. Therefore, the frequent pattern base of a system is just a collection of the frequent short sequences related to all key processes in this system.

Definition 6 (A Frequent Pattern). Given the size of sliding windows K and a set of short sequences $(s_{ij})_{M*N}$, using frequency analysis, we can get frequent patterns, denoted by $(f_{il})_{M*K}$, such that each frequent pattern f_{il} is the collection of frequent system calls in the lth position related to the ith process.

If $(s_{ij})_{M*N}$ is a training dataset of normal short sequences, we can build a normal frequent pattern base $(f_{il})_{M*K}$ that includes the frequent patterns trained from $(s_{ij})_{M*N}$. In addition, let i be fixed, $(s_{ij})_{M*N}$ is degenerated into a one-dimensional structure, denoted as $(s_{ij})_K$, which can be used to express the set of short sequences related to the fixed process p_i. Similarly, we can use $(f_{il})_K$ to express the frequent pattern of the fixed process p_i. Fox example, $(s_{1j})_K$ is the set of short sequences related to process p_1, and $(f_{1l})_K$ is the frequent pattern of process p_1.

After introducing the above concepts, we are now ready to present our algorithms *FP_Trainer* and *FP_Detector*, which respectively establish a frequent pattern base and detect anomaly behaviors using this pattern base.

If the normal process traces in a computer system has been obtained and their related short call sequences has been extracted, then the pseudocode of *FP_Trainer* can be described as follows.

Algorithm *FP_Trainer*$((s_{ij})_{M*N},(f_{il})_{M*K})$.
/* Mining pattern base $(f_{il})_{M*K}$ from training set $(s_{ij})_{M*N}$ */
begin
for i = 1 to M
  Make_pattern$((s_{ij})_N, (f_{il})_K)$;
  Merge$((f_{il})_K, (f_{il})_{M*N})$;
endfor;
end.

In *FP_Trainer*, there are two procedures to be called:

(1) Procedure *Make_pattern(),* which generates frequent patterns $(f_{il})_K$ from the short sequence set $(s_{ij})_N$ related to the fixed process p_i. According to Definition 6, this procedure can be implemented through computing the frequency of every call at each point.

(2) Procedure *Merge(),* which puts the newly generated patterns $(f_{il})_K$ into the frequent pattern base $(f_{il})_{M*K}$. They are relatively straightforward and therefore are omitted due to length constraints.

Once the normal database is defined, the next decision is how to measure a program behaviors are normal or abnormal. The easiest and most natural measure is to match a tested trace with defined normal patterns in the database.

Using the frequent pattern base, an unproven trace can be detected. Of course, when the detected trace is a long sequence of system calls, it has to be first transformed into a set of short sequences, which can use the above Procedure *Make_pattern()* to generate short sequences of the same size with ones in the defined pattern base. The following provides the description of algorithm *FP_Detector*.

Algorithm *FP_Detector*$((f_{il})_{M*K}, t)$.
/* Detecting sequence t using pattern base $(f_{il})_{M*K}$*/
begin
flag = *false*;
Transform t into a set of short sequences $(s_{ij})_{L-K-1}$;
Make_pattern$((s_{ij})_{L-K-1}, (f^*_{il})_K)$;
for l=1 to K
  if (f^*_{il} is not contained by f_{il}) then flag = *true*;
Return flag;
endfor;

end.

Frequent pattern mining for intrusion detection is relatively simple. The advantage of such a simple approach is computational efficiency, but it is not a very sophisticated method and its detection precision is not very high.

4. Tree Patterns for Anomaly Detection

In this section, we present another type of patterns for intrusion detection that is based on a tree structure. For every key process, its normal system call sequences can be organized into a tree pattern, such tree patterns for all key processes in a computer system just become a tree pattern base. Tree patterns can be generated by using tree-like mining techniques[5] such as Decision Tree, FP-Tree. Thus, how to define and build up such tree patterns is topic of this section.

Definition 7 (A Tree Pattern). Given the size of sliding windows K and a training dataset of short sequences $(s_{ij})_{M*N}$, the tree patterns for anomaly detection are trained from $(s_{ij})_{M*N}$, denoted as $(t_{il})_{M*V}$, are a set of trees, each of which t_{il} presents the lth tree in a forest (or a set of trees) related to process p_i. Furthermore, every path of t_{il} from the root to a leaf constitutes a short sequence in $(s_{ij})_{M*N}$. When i is fixed, we also use $(t_{il})_V$ to express the forest comprised of all pattern trees related to the fixed process p_i.

According to Definition 7, a tree pattern for anomaly detection is a set of forests each of which represents the mined patterns from sequences of a process. Figure 1 gives an example of a tree pattern base for anomaly detection, which has two processes and the size of short sequences is 4. Algorithm *TP_Trainer* to build a tree pattern base is designed as follows.

Algorithm *TP_Trainer*$((s_{ij})_{M*N}, (t_{il})_{M*V})$.
/* Mining pattern base $(t_{il})_{M*V}$ from training set $(s_{ij})_{M*N}$ */
begin
for i = 1 TO M
  for j = 1 TO N
    c = the first call in s_{ij};
    if (c is the root of a tree in $(t_{il})_V$)
      then Update this tree using s_{ij};
      else Create a tree in $(t_{il})_V$ using s_{ij};
  endfor;
end.

After a tree pattern base is built, we can make use of it to test an unproven sequence of system calls. A key problem is how to evaluate the matching degree between the tested short sequences and the tree pattern base.

Definition 8 (A Matching Value). Given a tree pattern base $(t_{il})_{M*V}$, and an user-specified matching parameter w (w<1), for a testing short sequence $s = < c_1, c_2, \ldots, c_K >$ related to process p_i (where K is the size of sliding windows), we only need to evaluate the matching degree between s and $(t_{il})_V$ that is the forest related to p_i in $(t_{il})_{M*V}$. The computing formula is as follows: $MV(s, (t_{il})_V) = w + w^2 + \ldots + w^L$, where L is the length of the longest sub-sequence of s that matches tree patterns $(t_{il})_V$.

According to Definition 8, computing MV of a short sequence in the known pattern base requires first searching for the longest path in the pattern tree, the root of which just matches the first call of the testing sequence. In order to show more details about it, before formally describing the algorithm *TP_Detector*, we give an example to illustrate how to get the MV between a short sequence and the pattern base.

Example 1. Considering the tree patterns in Figure 1. Assume K = 4, w = 1/2, and related to Process p_1. If a tested short sequence of system calls $s1$ = <3, 26, 65, 4>, then $MV(s1, (t_{1l})_V)$ = 1/2+1/4+1/8+1/16=15/16; If the tested short sequence of system calls $s2$ = <3, 26, 65, 75>, then $MV(s2, (t_{1l})_V)$ = 1/2+1/4+1/8=7/8; When the tested system call sequence $s3$ = <75, 3, 65, 4>, then $MV(s3, (t_{1l})_V) = 0$.

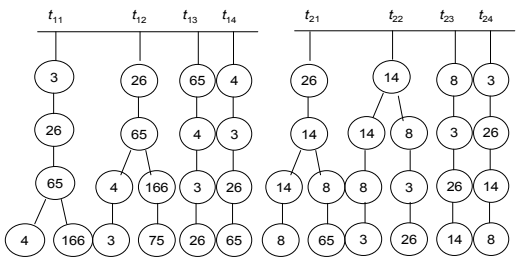

Fig. 1 An example of tree patterns

Using the above Function MV, the pseudocode of *TP_Detector*, the procedure to detect if an unproven call sequence is abnormal based on tree patterns, can be described as follows.

Algorithm *TP_Detector*($(t_{il})_{M*V}$, t).
/* *Testing trace t using tree pattern base*$(t_{il})_{M*V}$ */
begin
Let the process related to t be i;
flag = *false*;
Transform t into short sequences $(s_{ij})_{L\text{-}K\text{-}1}$
for j =1 TO L-K-1
Compute $MV(s_{ij}, (t_{il})_V)$;
if ($MV < \varepsilon$) then flag = *true*;
endfor;
Return flag;
end.

5. Experimental Evaluations

To evaluate the performances of the two models proposed in this paper, we conducted a series of experiments. All experiments were implemented using VC on a computer with 1.3G-CPU and 384M-RAM.

In addition, this paper is based on the idea of collecting traces of normal behavior in a system, and has an allied goal with some typical researches in this field[1, 2]. Such, the datasets used in our experiments can be selected from some public datasets. Therefore, all experiments in this paper are all related to the process *sendmail* in UNIX, which can be obtained from the public website (http://www.cs.unm.edu/~immsec/systemcalls.htm).

There is a good technical introduction to the process *sendmail*[1, 2]. Based on the standard *sendmail*, we design some datasets for training and testing our methods. Table 1 gives the names and features of the datasets used in this paper.

Table 1. Datasets and their features.

Name	Sequence #	Object	Feature
T_small	19526	Training	Normal
T_mid	96329	Training	Normal
T_big	1556560	Training	Normal
D_ano	128	Testing for detection precision to intrusions	Anomaly
D_nor	1556	Testing for false positives	Randomly extracted from *T_big*

Experiment 1. To assess the running time of *FP_Trainer* and *TP_Trainer*, where the size of the sliding window K = 11, the minimum support δ=20% on *FP_Trainaer*, and w = 1/e on *TP_Trainer*, Table 2 shows the results.

Table 2. Training Time of *FP_Tainer* and *TP_Tainer* (Sec).

Data	FP_Trainer	TP_Trainer
T_small	23	234
T_mid	102	598
T_big	1555	17331

Analysis of Results about Experiment 1: *FP_Tainer* has a much shorter training time than *TP_Tainer* on all datasets. The shorter training time on *FP_Tainer* comes from the simplicity of its model which makes computing and generating frequent patterns more efficient. However, both algorithms' scale-ups in training times are acceptable because they are both less than linear increases with the number of training sequences. As Table 2 shows, *T_big* has about 80 times more sequences than *T-small*, but the training time of *FP_Tainer* on *T_big* is 68 times less than on *T-small*, and the training time of *TP_Tainer* on *T_big* is 75 times less than on *T-small*.

Experiment 2. To assess memory usage of *FP_Trainer* and *TP_Trainer*, where K = 11, δ=20% on *FP_Trainer*, and w = 1/e on *TP_Trainer*, Table 3 shows the results.

Table 3. Memory Usage on *FP_Tainer* and *TP_Tainer* (*KB*).

Data	FP_Trainer	TP_Trainer
T_small	716	2520
T_mid	3744	10872
T_big	6140	25021

Analysis of Results about Experiment 2: *FP_Tainer* has much less main memory usage than *TP_Tainer* on all datasets. Also, both scale-ups in memory usage for training with increasing sizes of sequences are stable. In fact, due to the limited number of system calls in a certain system, both the frequency patterns and the tree patterns will show an obvious astringency in size with increasing the training data. This astringency results in

their good properties in memory utilization to a large-scale training dataset.

Experiment 3. This experiment assesses the detection precision of *FP_Detector* and *TP_Detector* to detect anomaly sequences, where K = 11, δ=20% on *FP_Detector*, and w = 1/e on *TP_Detector*. We use dataset *D_Ano* which includes 128 anomaly sequences to test whether *FP_Detector* and *TP_Detector* can successfully detect them. *FP_Detector* and *TP_Detector* make use of the patterns learn from *FP_Trainor* and *TP_Trainor* on dataset *T_big*. For comparative studies, we programmed the *HMMs* algorithm[7], which is randomly initialized with 60 states and then learns its parameters by dataset *T_big*. Figure 2 shows the detection accuracies (that reflects their false negative rates) of the three methods when the size of *D_Ano* is increased one by one.

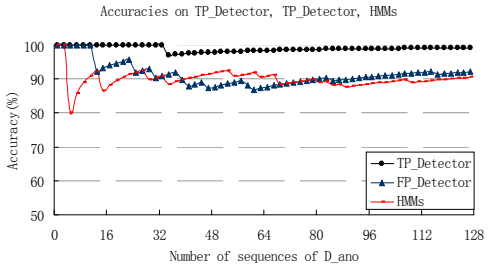

Fig. 2 Accuracy changes of *TP-Detector*, *FP-Detector* and *HMMs* with increasing sequences on *D_Ano*. The same test data were used for three algorithms, but their accuracy changes were obvious different. Along 128 tested sequences, *TP-Detector* failed only once located on the 35th; *FP-Detector* failed ten times located on the 13th, 25th, 30th, 38th, 39th, 46th, 58th, 81th, 85th and 116th; *HMMs* failed twelve times located on the 5th, 14th, 29th, 35th, 55^{h}, 63th, 68th, 69th, 80th, 84th, 89th and 109th.

Analysis of Results about Experiment 3: As Fig. 2 shows, *TP_ Detector* has a much higher detection precision than *FP_Detector* and *HMMs*. Through deeply analyzing the lengths of anomaly sequences in *D_Ano*, *FP_Detector* and *HMMs* did not successfully find out such sequences that are longer than the average length of all sequence of *D_Ano*. Such, it may be guessed that *FP_Detector* and *HMMs* could not be the good methods for detecting the long testing sequences, but *FP_Detector* can solve this problem well.

Experiment 4. This experiment assesses the false positive rates of *FP_Detector* and *TP_Detector*. For this goal, we randomly extract a testing dataset called *D_nor* from *T_big*, where its size is about 0.1% of the size of *T_big*. We set sliding window K = 11. For model *FP*, *FP_Trainor* learn the patterns on *T_big* with several different min-support values: δ = 10~25%, then *FP_Detector* is used to *D_nor* to test whether or not its sequences can be successfully flagged as normal sequences. In model *TP*, *TP_Trainor* uses w = 1/e to run on *T_big* to get the pattern base, then we use *TP_Detector* to test *D_nor* with min-match values: ε = 45~60%. Such, this experiment can also help evaluate the effects of detection accuracy on different min-support or min-match values. Table 4 gives the related false positive rates of *FP_Detector* and *TP_Detector* in dataset *D_nor*.

Table 4. False positive rates (FPS) of *TP-Detector* and *FP-Detector* on *D_Ano* . The size of their sliding windows is 11. In *FP-Detector*, min-supports are respectively set into 10%, 15%, 20%, 25%. In *FP-Detector*, matching parameter w = 1/e. min-matchs are 45%, 50%, 55%, 55%.

	FP			
	δ=10%	δ=15%	δ=20%	δ=25%
FSR (%)	**0.193**	**0.321**	**0.514**	**0.707**
	TP			
	ε = 45%	ε = 50%	ε = 55%	ε = 60%
FSR (%)	**0**	**0**	**0.063**	**0.193**

Analysis of Results about Experiment 4: As Table 4 shows, *FP_Detector* can result in higher false positive rates when min-supports are bigger. This is because that a smaller min-support means more normal patterns are mined and saved and so less normal data to test can be mistaken as being anomaly, but it also increases the risk that an anomaly data is mistaken as a normal one. On the other hand, when min-matchs get bigger, *TP_ Detector* can result in higher false positive rates. In fact, mini-match parameter is set in order to detect intrusions, and a smaller mini-match means a more cautious strategy to affirm an intruder, so it can decrease false positive rates. Of course, *TP_ Detector* has lower false positive rates than ones of *FP_ Detector* on all investigated datasets, but both can get rather good detect results.

6. Conclusion

This paper presents two novel approaches to anomaly detection. These approaches make use of data mining techniques to learn normal pattern bases. Model *FP* uses frequency pattern mining technology to quickly find out frequent system call sequences, so it has efficient advantages in computing and memory usage. On the other hand, Model *TP* makes use of tree-like pattern mining methods, that could get a good detection accuracy. We also thoroughly evaluated our models in false positive rates and false negative rates by different control parameters. The experiments showed and validated their advantages disadvantages.

Future work is going to apply more data mining methods to anomaly detection, and to compare their performances each other including of the models in this paper.

Acknowledgments

This work is supported by grants from the National Science Foundation in China (60873145) and the Discipline Construction Foundation of CUFE.

References

1. S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, A sense of self for UNIX processes, in *Proc. 1996 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press (Los Alamitos, CA, 1996), pp. 120–128
2. S. A. Hofmeyr, S. Forrest and A. Somayaji, Intrusion detection using sequences of system calls, *J. Computer Security*, **6**(3) (1998) 151-180.
3. W. Lee, S. J. Stolfo and M. Chan, Learning patterns from Unix process execution traces for intrusion detection, in *Proc. AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, AAAI Press (Menlo Park, 1997), pp.50–56.
4. P. Helman and J.Bhangoo, A statistically based system for prioritizing information exploration under uncertainty, *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, **27**(4)(1997) 449–466.
5. W. Lee and S. J. Stolfo, Data mining approaches for intrusion detection, in *Proc. 7th USENIX Security Symposium*, Usenix Association (San Antonio, TX, 1998), pp. 79-94.
6. S. T. Brugger, Data mining methods for network intrusion detection, *Ph. Dissertation, University of California* (Davis, CA, USA, 2004).
7. C. Warrender, S. Forrest and B.Pearlmutter, Detecting intrusions using system calls: alternative data models, in *Proc. 1999 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press (Oakland, CA, USA, 1999), pp.133-145.
8. J. Z.Mohammed, Efficiently mining frequent trees in a forest, in *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press (Alberta, Canada, 2002), pp. 71-80.
9. W. Lee and S. Stolfo, A data mining framework for Building intrusion detection models, *ACM Transactions on Information and System Security*, **3**(4) (2000) 227-261.
10. W. Lee and X. Dong, Information-theoretic measures for anomaly detection, in *Proc. 2001 IEEE Symp. on Security and Privacy*, IEEE Computer Society Press (2001), pp.130−143.
11. R. Lippmann, D. Fried and I. Graf, Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation, in *Proc. 2000 DARPA Information Survivability Conference and Exposition* (2000), pp. 12-26.
12. J. W. Haines, D. J. Fried and J. Korba, Analysis and results of the 1999 DARPA off-line intrusion detection evaluation, in *Proc. Intl. Symposium on Recent Advances in Intrusion Detection*, Springer-Verlag (2000), pp. 162-182.
13. X. Li, and N. Ye, Decision tree classifiers for computer intrusion detection. *Journal of Parallel and Distributed Computing Practices*, **4**(2) (2001) 179−190.
14. N. Ye, X. Li, Q. Chen, Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Trans. Syst. Man Cybern*., **31**(4) (2001) 266–274.
15. G. Singh, F. Masseglia1, C. Fiot and A. Marascu1, Data mining for intrusion detection: from outliers to true intrusions, in *Proc. the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer (Bangkok, Thailand, 2009), pp. 891-898.
16. A .Patcha and J. M. Park, An overview of anomaly detection techniques: existing solutions and latest technological trends. *Computer Networks*, **51**(7) (2007) 3448-3470.
17. V. Chandola, S. Boriah and V. Kumar, A reference based analysis framework for analyzing system call traces, in *Proc. the ACM Sixth Annual Workshop on Cyber Security and Information Intelligence Research* (2010). Retrieved from http://dx.doi.org/10.1145/1852666.1852703.