Hybrid Intrusion Detection Model Based on Ordered Sequences

Abdulrahman Alharby and Hideki Imai

Institute of industrial Science, The university of Tokyo,
4-6-1, Komaba, Meguro-ku, Tokyo, 153-8505 Japan
alharby@imailab.iis.u-tokyo.ac.jp
imai@iis.u-tokyo.ac.jp

Abstract. An algorithm for designing hybrid intrusion detection system based on behavior analysis technique is proposed. This system can be used to generate attack signatures and to detect anomalous behavior. The approach can distinguish the order of attack behavior, and overcome the limitation of the methods based on mismatch or frequencies, which performs statistical analysis against attack behavior with association rules or frequent episode algorithms. The preprocessed data of the algorithm are the connection records extracted from DARPA's tcpdump data. The algorithm complexity is analyzed against a very known algorithm, and its complexity is decreased greatly. Using the proposed algorithm with transactions of known attacks, we found out that our algorithm describes attacks more accurately, and it can detect those attacks of limited number of transactions. Thus, any important sequence is considered and discovered, even if it's a single sequence because the extraction will cover all possible sequences combinations within the attack transactions. Four types of attacks are examined to cover all DARPA attack categories.

Keywords: intrusion detection, continuous pattern, discontinuous pattern, data mining.

1 Introduction

Over the past decade, the number as well as the severity of computer attacks has significantly increased. CSO magazine conducted a survey on the 2004 cyber crimes, the survey shows a significant increase in reported electronic crimes. Compared to the previous year, more than 40% of intrusions and electronic crimes are reported. Also, 70% of the respondents reported at least one electronic crime or intrusion was committed against their organization [1]. According to collected statistics, electronic crimes have an incredible impact on economy. Reports say that electronic crimes have cost more that $600 million in 2003.

IDSs are considered as powerful security tools in computer systems environments. These systems collect activities within the protected network and analyze them in order to detect intrusions. System activities are usually collected from

V. Gorodetsky, I. Kotenko, and V. Skormin (Eds.): MMM-ACNS 2005, LNCS 3685, pp. 352–365, 2005.

two main sources, network packet streams and host log files. Once the information is collected, the detection algorithm starts looking for any evidence for intrusions existence.

There are two general methodologies of detection used by IDSs: *misuse* and *anomaly* detection [2,3]. *Misuse* detection systems such as STAT [4] look for a known malicious behavior or signature, once it is detected an alarm is raised for further actions. While this type is useful for detecting known attacks, it can't detect novel attacks, and its signatures database needs to be upgraded frequently. The main feature of this model is its low false alarm rate. *Anomaly* detection models (e.g. IDES [5]) compare reference model of normal behavior with the suspicious activities and flag deviations as anomalous and potentially intrusive. Unlike *misuse* detection, *anomaly* detection systems identify unknown intrusions. The most apparent drawback of these systems is the high rate of false alarms. The two detection approaches can be combined to detect attacks more efficiently. There are various types of detection models (e.g. [6], [7], and [8]). Among these techniques, ADAM: Audit data analysis and mining, association rules data mining [9,10], and classification data mining [11,12,13] are the main used algorithms.

Following this introduction, we provide a background on the related work, and a briefing of our contribution. Section 2 then presents the proposed algorithm. In Section 3, the experiments are explained, including: data set model, details of learning and detection phases. Finally, Section 4 summarizes this paper's main conclusions.

1.1 Related Work

There has been extensive considerable work in representing and recognizing normal or malicious activities. Henry et al. in [14] proposed an approach that uses a time-based inductive machine (TIM) to generate rule-based sequential patterns that characterize the behavior of a user. This approach, to some extent, is similar to our approach in that both can be used to offer a simplified view of a set of complex data. There are, however, some fundamental differences between the two approaches: first, Henry's approach conducts a heuristic search to find the rules that satisfy certain given criteria, while our approach is mainly used for the evaluation of generated patterns. Second, Henry's model uses only continuous patterns, while our model combines both continuous and discontinuous patterns. Third, in the case of using our model as *Anomaly* detection, deviation from the norm in TIM is detected by matching the two sides of the rule, while in our model deviation is conducted by the summation weights of the matched patterns.

The most efforts that contribute to the current proposal are proposed by Kim and Wenke lee in [15] and in [16] respectively. While Kim proposed a new intrusion detection classification using data mining based on CTAR which considers temporal attribute of audit data. Wenke applied data mining with frequent episode algorithm, and structure statistic features. Wenke built his detection model based on RIPPER classifier. In the following, we summarize some drawbacks that have been noticed in these two approaches: First, although some

intrusion behaviors depend on frequent episode or temporal attribute, analysis based on statistical features may not reflect the different features relationship in the context of time order. e.g., attacks with features appearing only once in the records, and attacks based on features that don't have frequent connection records or features that occur only once in an attack. Second, both detection methods of Wenke and Kim were designed to detect mainly Probe and DoS attacks. Current efforts of intrusion detection focus on detecting attacks with no clear evident features, such as application layer attacks or what are called in DARPA dataset remote to local and user to root attacks. Third. the most important, using statistical analysis would lead to lose order actions. Because attack evident features spread over many records, we need a technique to search the records vertically, and dig out the records for each single itemset sequences that may reflect attack features, that is continuous and discontinuous based data mining.

1.2 Our Contribution

The objective of this paper is to treat the systems ordered actions differently. Our approach uses the continuous and discontinues patterns to characterize the system behaviour. We used the proposed technique to extract some attacks signatures, and also to build an anomaly detection classifier. To classify a new sequence into either normal or intrusive, our proposed classifier converts the new sequence into a number of patterns and then calculates the similarity between these patterns and those of the training sequences. There are some advantages to applying this method to intrusion detection: First, without affecting the detection rate, limited and reasonable deviations from the norm are allowed, thus, false positive rate is significantly reduced. Second, foremost advantage is that this technique aims to discover all important possible patterns within the sequence. Third, in case of using this technique for building attack signature, it can deal with any kind of attack attributes such as time, numerical, categorical, and free-text.

2 Proposed Algorithm

2.1 Notations and Definitions

This section defines concepts that are central to this article, including the fundamental notions and definitions.

Definition 1 (Notions).

- $C(k,l)$: used to represent the set of candidate sequences of k elements and l stars.
- $L(k,l)$: The sequences set that have a support value bigger than the given minimal support where the sequence length is k and it has l stars.

- *SupL(k,l)*: The super large set, *SupL(k,l)*, used to store the list of all supported sequences of both types continuous and discontinuous.
- Pattern: also called sequence, it is a number of ordered actions. the pattern X can be shown as $(x_1, x_2, .., x_n)$, each x_j means an item or element.
- record: single instance of an attack. If an attack is involved in multi-instances, then we say attack records for all involved instances.

Definition 2 (continuous patterns). Suppose a pattern S_i extracted from the sequence $X_i = \{x_1, x_2, ..., x_m\}$ and contains some actions, that is, $S_i = \{s_1, s_2, ..., s_l\}$ which may reflect ordered commands executed by a program run on a computer machine. The pattern S_i can be classified as continuous pattern if all contained elements appear in consecutive positions of the sequence X_i, such that, there is an integer r such that; $s_1 = x_r,\ s_2 = x_{r+1}, ...,\ s_d = x_{r+l-1}$. *For example*, the continuous pattern (s_3, s_4) occurs in sequence: $X_1 = (s_1, s_2, s_3, s_4, s_5, s_6)$.

Definition 3 (discontinuous patterns). We say that S_i is a discontinuous pattern if the elements of that pattern don't appear in consecutive positions of the sequence X_i, that is, if there are existing integers $r_1 < r_2 < ... < r_l$ such that $s_1 = x_{r_1},\ s_2 = x_{r_2}, ...,\ s_l = x_{r_l}$. *For example*, the pattern $(s_1, *, s_4)$ in sequence: $X_1 = (s_1, s_2, s_3, s_4, s_5, s_6)$ is a discontinuous pattern.

Definition 4 (star patterns). Star pattern is a pattern that contains one star or more as part of its elements. In a discontinuous pattern, hidden elements represented by star " $*$ " which is defined as a variable number of intermediate elements. The star pattern never starts or ends by " $*$ ". For example, if we have a sequence $X_i = \{x_1, x_2, x_3, x_4\}$, we may have these continuous patterns $(x_1, x_2), (x_2, x_3, x_4)$, and (x_1, x_2, x_3), or this discontinuous pattern $(x_1 * x_3, x_4)$. Because of the definition of the " $*$ ", the pattern $(x_1 * x_3, x_4)$ implicitly has two other patterns: (x_1, x_3, x_4), and (x_1, x_2, x_3, x_4).

2.2 Data Analysis and Patterns Generation

DARPA 1998 off-line data sets [17] developed to evaluate any proposed techniques for intrusion detection. These data prepared and managed by MIT Lincoln labs, sponsored by DARPA, and contain contents of every packet transmitted between hosts inside and outside a simulated military base. There were a collection of data including TCPDUMP and Basic security module (BSM) audit data of a victim Solaris machine. Both types are used in this work. While we used BSM data to model users normal behavior, we preprocessed and used tcpdump data set to model attack behavior. tcpdump records consist of a number of attributes as items of sequences, and these items include class attribute and other attributes, which are shown in Figure 1.

The aim of the proposed algorithm is to find out all frequent patterns from an attack records. Compared with CTAR or even with traditional Apriori algorithm, the proposed algorithm mines two types of sequences, one is continuous, and the

Service	*Src Port*	*Dest Port*	*Src IP address*	*Dest IP address*	*Class Attack/Normal*
http	*1106*	*80*	*192.168.001.005*	*192.168.001.001*	*Normal*
telnet	*20504*	*23*	*172.218.117.069*	*172.016.113.050*	*loadmodule*
...	...	...	...	...	...

Fig. 1. Dataset records, each one has a number of attributes. Class attribute has two categories, normal or attack. The rest of the attributes have many values.

other is discontinuous. The algorithm includes two steps, the first step is to search large-sequences of the first type of patterns, and the second step is to search the second type of patterns. In the following, the steps are summarized as follows:

- All attribute values in records database are considered as candidates to 1-element-zero-star-sequence-itemset, $C(1,0)$. After generating $C(1,0)$, the records database is scanned vertically. If the elements of $C(1,0)$ are contained in any instance, then the support of that element adds 1. Insert any element with support value greater than the given minimal support in 1-element-zero-star-sequence-large-itemset, $L(1,0)$, and store the results in a temporary database.
- Each two elements from two different attributes in $L(1,0)$ are combined to form 2-element-sequence-itemset-zero-star, $C(2,0)$. The records database is scanned for all patterns existing in $C(2,0)$. When the support value of a pattern exceeds the given minimal support it inserts in 2-element-sequence-large-itemset-zero-star, $L(2,0)$. We find out all k-element-large-zero-star $L(k,0)$ and store in a temporary database in turn. And then, we list all large-zero-star-sequence, $L(1,0)$, $L(2,0)$,..., $L(m,0)$, and store them in a common database called super large sequences set, *SupL*.
- After generating all possible $L(k,0)$, we extract all discontinuous patterns. First, from the temporary database of $L(3,0)$ we found out 2-element-1-star-sequence $C(2,1)$ by replacing the second item of the pattern by star. And then the records are scanned vertically for each pattern existing in $C(2,1)$, the patterns that have a support value exceeding the given minimal support are inserted in 2-element-zero-star-sequence-large-itemset, $L(2,1)$. We then found out all 2-element-l-star-large-itemset $L(2,l)$, and list all large-l-star-sequence, $L(2,1)$, $L(2,2)$, ..., $L(2,l)$. We do the same thing for all k-element-large-zero-star $L(k,0)$ in turn. The resulting sets add to *SupL* database. These steps are shown in Figure 2.

In order to describe the algorithm clearly, we will take the example of an attack that includes 5 items and generate all possible sequences, which are shown in figure 3.

2.3 Complexity Analysis

The proposed algorithm is very different from Apriori algorithm [18]. First, discontinuous sequences are not considered in Apriori algorithm. Second, item-

```
Input: Extracted transactions from Original records.
Output: SupL; L(k,l) for all ks and ls

//Generate all possible candidate patterns of 1-element-sequence
C(1,0) = gen (Original records)
//Extract 1-element-sequences that have support value bigger than the min_support
L(1,0) = subset (C(1,0))
   For (2 ≤ k ≤ m)
      C(k,0) = gen (L(k-1,0))    //Generate all combinations of L(k-1,0)
      L(k,0) = subset (C(k,0))    //Extract all supported continuous patterns
         For (1 ≤ l ≤ m-2)
            C(k,l) = gen (L(k,0))    //Generate all combinations L(k,0) with star
             L(k,l) = subset (C(k,l))    //Extract all supported discontinuous patterns
```

Fig. 2. Proposed algorithm

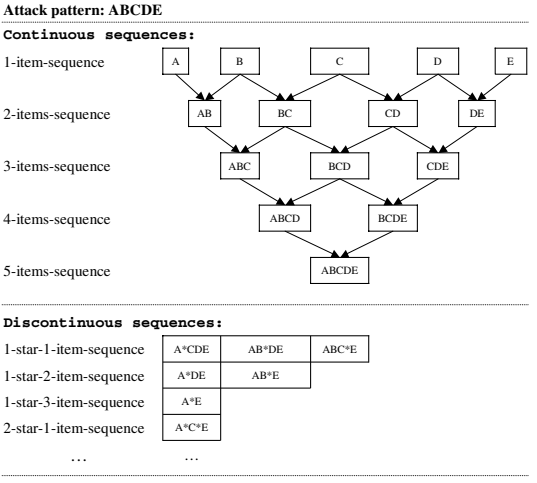

Fig. 3. Pattern extraction trees

record data is scanned vertically instead of horizontally. Among other steps, we found calculating the support value is the most time-consuming step, algorithm of support calculating is shown in Figure 4. Thus, the proposed algorithm reduces the complexity of continuous and discontinuous patterns mining greatly.

The Apriori algorithm built based on an iterative technique, where k-itemsets are used to generate $(k+1)$-itemsets. First, supported 1-itemset is generated, i.e. $L(1,0)$. Then, $L(1,0)$ is employed to generate the set of frequent 2-itemsets, i.e. $L(2,0)$, which is used to find $L(3,0)$, and so on until all supported k-itemsets are extracted. The next process consists of two actions; joining and pruning. First, the join step: To generate $L(k,0)$, a candidate set k-itemsets is extracted by joining $L(k\text{-}1,0)$ with itself, where items of $L(k\text{-}1,0)$ can be joined if their

```
Input: i=1, j=1, sequence x, pattern t
Output: sum, number of t included in x
 number(String[] x, int i, String[] t, int j){
  if (x[i]=(" * ")) i++; // If we have a star, skip it, it was already used
  // If the star was the last character, found another match.
  if (i = m AND x[i] = (" * ")) return ++sum;
  if (j = n) {return sum;}
  if (i = 0 AND j = 0) sum = 0;
  // The " i > 0 " test simulates a starting star.
  if (i > 0 AND x[i - 1] ≠ (" * ")) {
     if (x[i] = (t[j]) AND i = (m - 1)) { sum++;}
     else if (x[i] = (t[j])) { number(x, i + 1, t, j + 1); } }
  else {
     for (int p = j; p < n; p++) {
      if (x[i] = (t[p]) AND i = (m - 1)) {sum++;}
      else if (x[i] = (t[p])) { number(x, i + 1, t, p + 1); }
      }
 } return sum; }
```

Fig. 4. An algorithm to find out how frequent is each pattern within a certain number of records

first (k-2) items are similar. This set of candidate is denoted $C(k,0)$. Second, the prune step: $C(k,0)$ is a superset of $L(k,0)$, that is, its elements may or may not be frequent, but all of the frequent k-itemsets are included in $C(k,0)$, even if $C(k,0)$ is very large. In fact any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k-1)-subset of a candidate k-itemset is not in $L(k\text{-}1,0)$, then the candidate cannot be frequent either, and so, can be removed from $C(k,0)$. Suppose there are n records in the original data set, to find all n large sequences, the number of connection will be 2^n. To build the signature of an attack with around 100 records, this structure is not suitable.

In contrast, when we search for 1-itemsets candidate, $C(1,0)$, with our proposed algorithm, we need to scan the original records once and count all items, the same as the Apriori algorithm. When searching for frequent 1-itemsets, $L(1,0)$, instead of scanning original records, we only need to scan $C(1,0)$ which is composed of original records and much less than the original data. After generating all $L(k,0)$, we scan the original records once, and every $C(k,0)$ is also scanned once. In total, k times of scanning are performed. Since any $L(k,l)$ is extracted from the corresponding $L(k,0)$, we only need to scan the data stored in the temporary database instead of the corresponding $C(k,0)$ or original records. The data quantity is reduced evidently. And the most important, by taking out $C(2,0)$, and only scan the corresponding $L(1,0)$ which may compose the $C(2,0)$ in the temporary database. Then, the other $C(2,0)$ is taken out in turn.

Thus, for a limited number of attributes and more records, the proposed algorithm has proved more efficient compared to Apriori.

3 Experiments

3.1 Misuse Detection

For the sake of clarity, the algorithm is described through the example of number of attacks. Each attack includes a number of records, in some attacks tens of instances are collected, each record includes five attributes shown in table 1. We dig out continuous and discontinuous patterns of these attacks with the misuse intrusion detection algorithms. Results are shown in Figure 5.

The first examined attack is *Back* attack, which belongs to denial of service attack against the Apache web server. *Back* attack is fabricated by submitting frontslashes contained in URL's requests. The *Back* attack causes instances of the http process on the victim machine. As the server tries to process these requests it becomes unable to process other requests, consequently, the attack slows down the server. Attack signatures in Figure 5 show that the attacker https to the victim machine "172.016.114.050" from a certain machine. This flow of request consumes excessive processor time, when the original data was checked back, we found the attribute *Src port* has many values, none of them support the minimal given support value. Consequently, it is replaced by star in the patterns, and didn't appear in large-sequences $L(1,0)$ or in super-large-sequences, $SupL$.

The second simulated attack is the *ftp-write* attack, which belongs to R2U attack. It takes advantage of misconfiguration of an anonymous ftp, which allows the intruder to add files such as an rhosts file, and gain local access to the system. This is exactly what the patterns show in Figure 5. Regardless of the values of attributes: *Src port* and *dest port*, which are represented by star, the attacker anonymously ftps the victim machine, performs some tasks such as creating ".rhosts" file, and disconnects from the server. Then, as the second pattern shows, login to the victim machine by using rlogin to connect back to the server as ftp user, and finally performs some illegal actions on the victim machine.

An *eject* attack, the third simulated attack, belongs to U2R category. It exploits buffer overflow vulnerability of the distributed "eject" binary with Solaris 2.5. This vulnerability, if exploited, is used to gain root access on the attacked machine. As shown from the attack signature in figure 5, the attacker telnets the workstation "172.016.112.050", regardless of what source port is used, or from where the attack is launched, which explains the stars in the last three patterns. Then, telnet victim machine is exploited to distribute the malicious code. The implanted code, if compiled, can be run on the victim machine, as a command line session where the attacker can gain root access.

The last simulated attack is *ipsweep* which belongs to the probing attacks family. Attackers use this attack to search for vulnerable machines to determine which hosts are listening on a network. It can be performed by sending an ICMP Ping packets to every possible address within a subnet, listening machines will respond to the sender. The generated attack pattern shows that a Ping packet

Attack type	Generated patterns for chosen attacks
back (DoS) Week-2 Friday	Pattern 1: http (*service*) Pattern 2: 80 (*Des. port*) Pattern 3: 135.008.060.182 (*Src. IP*) Pattern 4: 172.016.114.050 (*Des. IP*) Pattern 5: http,*,80 Pattern 6: 135.008.060.182,172.016.114.050 Pattern 7: http,*,80,135.008.060.182 Pattern 8: http,*,80,135.008.060.182,172.016.114.050
ftp-write (R2U) Week-2,Friday	Pattern 1: ftp,*,195.073.151.050,172.016.112.050 Pattern 2: Login,*,195.073.151.050,172.016.112.050
eject (U2R) Week-6 Thursday	Pattern 1: telnet (*service*) Pattern 2: 23 (*Des. port*) Pattern 3: 172.016.112.050 (*Des. IP*) Pattern 4: telnet,*,23 Pattern 5: 23,*,172.016.112.050 Pattern 6: telnet,*,23,*,172.016.112.050
ipsweep (Probing) Week-3,Wednesday	Pattern 1: eco/i,7,7,202.077.162.213,*

Fig. 5. Number of chosen attacks, and their behavior as continuous and discontinuous sequences

"eco/i" is always sent from the same source "202.077.162.213", and the attribute *Dest IP address* is replaced by star " $*$ " which explains that the Ping message is sent to a variety of destinations. That is exactly how the attack is performed.

The experiment indicates that the pattern we obtained is different from the command pattern, it is a new pattern. It can describe attacks more accurately, detect the attacks whose features appear only once, improve detection rate, and offer a new idea for the research of intrusion detection.

3.2 Anomaly Detection

Data Model and Preprocessing. In our experiments, and to evaluate the algorithm as an anomaly detector, we used the Basic Security Module (BSM) audit data collected by DARPA. Besides many attributes of BSM events, each session contains one or more system calls information that are generated by the programs running on the Solaris system. Also, each session is labelled with a related unique process number.

Programmatically, for each single process all related individual sessions are extracted, and then the complete set of ordered system calls spreading over the sessions are recorded. For our data model, we only recorded the names of the executed system calls ignoring other session attributes. And then, the algorithm is used to transform each process into its related continuous and discontinuous patterns. A sample of System calls generated by one user during two processes; 118 and 102 is shown in table 1.

Anomaly Model. Our implementation is based on normal programs behavior. Two stages have to be defined, the learning and detection stages. In the following, the two stages are presented in more details.

Table 1. Sample of ordered normal system calls included in two processes 118, and 102, Executed by the user named by: *franko* within the first day of the first week of the training 1998 DARPA data set

Process	System calls
118	stat stat stat stat chdir chdir lstat stat stat open chdir chdir lstat stat stat open pathdonf stat stat open chdir pathdonf stat open chdir pathdonf stat stat open chdir
102	stat stat stat stat access stat open open access stat open open

Learning Process. DARPA simulated BSM audit data set featured 6 users whose activity can be used to test anomaly detection systems. The users are named as: *franko*, *georgeb*, *janes*, *fredd*, *williamf*, and *donaldh*. The activity of those users remains consistent from day to day, but on some days, those users exhibit anomalous behavior in ways that should be detectable to an anomaly detection system. The anomalies that are introduced into the users' sessions include logging in from a different source, logging in at an unusual time, executing new commands, and changing identity. In the training data, all anomalies were introduced during the 6^{th} week.

Among the seven weeks training period of DARPA data set, there are 6 weeks free of anomalous behaviour. Arbitrarily, 2 weeks (the first and the second) picked as a training data set, and left the sixth week for testing.We recorded only the names of the ordered system calls executed by those 6 users. Users names are usually found in two attributes: *path* or *mail*. Any process not related to any one of those users are ignored in either data sets, training or testing. The 2 weeks training data set consists of 17 intrusive instances and 17 clear or stealthy attacks. There are 7798 sessions within these 2 weeks. These normal training processes run only on Solaris machine. Once we have the training data set for the normal behavior, each single process is transformed to its related continuous and discontinuous patterns.

The proposed algorithm is used to generate all large-sequences $L(k,l)$ patterns that could be contained within one normal process. All system calls within one process are considered as a candidate to 1-element-sequence-itemset and stored in $C(1,0)$. This collection of patterns are used as a normal profile.

At a certain detector window size k, Large-sequences $L(1,0)$ patterns of only one process were generated in each run. A single process may contain a number of elements more than the detector window size, in this case, we applied the algorithm for the first k elements, and then moved to the next k elements until we covered all the elements included in the process.

We look for all normal processes separately and generate super-large- sequences, *SupL*. The resulting normal patterns are stored in a temporary data-

```
//build training normal patterns data set
Extract large-sequence L(k,l) of training dataset, and store in SupL_N;
for each process X in the testing data set Do
   extract all large-sequence L(k,l) patterns, and store in SupL_S;
   get value of n; // extracted programmatically
   compare SupL_N and SupL_S and get k_n;
   calculate k_n/n;
        if k_n/n ≥ threshold then
           The process X is normal;
        else then
           The process X is abnormal;
```

Fig. 6. An algorithm code for anomaly classifier

base called "normal pattern database" and denoted by $SupL_N$, and used later as a normal profile during monitoring and classifying testing processes.

Detection Process. This phase is intended to classify the testing processes to intrusive or normal. Once we have the training patterns data set for normal behavior, testing audit data is scanned for each new process associated with the same chosen 6 users. The new processes are also transformed to their related large-sequences patterns, $L(k,l)$. All possible patterns were generated for each testing process, and stored in a temporary database called "suspicious patterns database" and denoted by $SupL_S$. Then the similarity between patterns of the new process and the patterns of normal processes is calculated using similarity algorithm.

The similarity algorithm is described as follows: for any testing process that is needed to be classified, first, all corresponding large-sequence patterns $L(k,l)$ are extracted, and then each single generated pattern that is represented in $SupL_N$ database is given a weight $w = 1/n$, where n is the total number of all extracted patterns of that specific testing process. The value of n can be extracted programmatically. The value of w falls in the range $(0 \leq w \leq 1)$. By calculating the total summation weights (k_n) of all matches, strength of the normality signal can be determined. If the total weights summation exceeds a certain threshold, the testing process is classified as normal. Otherwise, it is an anomalous process. In Figure 6 an abstract of the pseudo code of the similarity algorithm is given.

Performance Measurements. Based on similarity function return value, the classifier makes the decision whether the process under investigation is intrusive or not. The first error that may occur is the false positive error which occurs when normal processes are classified as intrusions. The second error type is the false negative error which occurs when the real intrusive process is classified as normal, which is more serious.

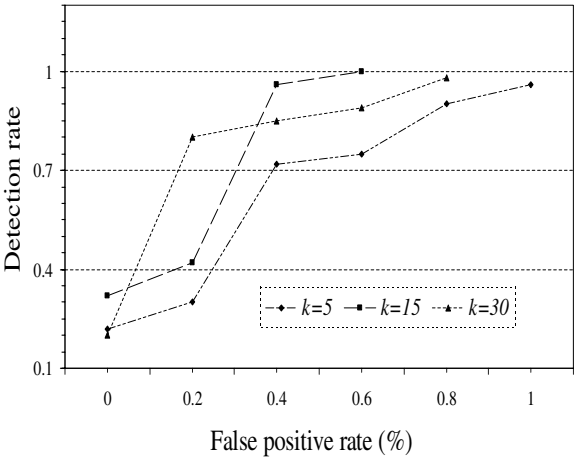

Fig. 7. Performance of the algorithm expressed in ROC curves. False positive rate vs attack detection rate for k =5, 15, and 30

Receiver Operating Characteristic (ROC) is a performance evaluation technique used to evaluate the intrusion detection algorithm [19]. It is related to the false error, and it is a trade off between detection rate and false alarms generated by the intrusion detection system. It can be obtained by varying the detection threshold and measuring the corresponding number of false alarms. This technique indicates how detection rate changes affect the raised false alarms. In our work, we used ROC metric to measure the performance of the proposed algorithm.

To evaluate the proposed algorithm as an anomaly detector, we formed a test data set from the DARPA BSM data of the 5 days of the sixth training week (none of the training data was chosen from this week). There are 53 intrusive sessions included in this testing data, and 14 distinct attacks included in these intrusive sessions. Also, 10 anomalous behaviors are included, such as unusual time logging in or from different source logging in, and new commands execution. Many of the attacks sessions were duplicated and appeared many times, like: *eject*, *neptune*, and *pod*. Duplicated sessions were not considered. Each process was classified to normal or intrusive, sessions associated with a single intrusive process was considered as an attack or anomalous sessions. The performance of the algorithm is evaluated as the detection rate versus false positive alarms. Detection rate and false positive alarm were built based on intrusive sessions detection and normal sessions misclassification. If one session is included in at least one intrusive process, it is counted as one attack. In our experiments, the presence of more than one intrusive process in one session does not affect the number of detection.

The proposed classifier can generate the related large-sequence patterns $L(k,l)$ of any length of sequences, this length may cover all the elements of the process, or just part of the process, and it is called the detector window

size and denoted as k. A detector window size that is smaller than the length of the process would cause the detector to parse one process into many sequences resulting in a low anomaly signal. At the same time, a detector window size that is larger than the process would cause the detector to see only the one process sequences in the given instance resulting in a fair anomaly detection.

In the experiments, we varied $k's$ value from 5 to 30, most of the processes contained a number of system calls less than 30. Compared to the processes sequences, these values cover the possibilities of being equal, less, or greater than processes length. Precisely, this choice describes how does the value of k affect the performance of the classifier. Figure 7 shows the ROC curves for three different k values. For this particular training and testing data set, $k = 15$ is the best choice, with this value, the detection rate reaches 100% faster and at low false positive rate compared with the other two $k's$ values. For $k = 15$, the classifier algorithm can detect out of 10 anomalous sessions only 3 sessions with zero false positive rate. Reducing the similarity threshold leads to higher detection rate, but, this reduction has some cost in that the false positive rate becomes higher. For $k = 15$, and at threshold 0.81, the detection rate reaches 100% with false positive rate 0.6% (only 48 false positive detection out of 7798 normal sessions included in the training data set).

4 Conclusion

A new classifier has been proposed, it's built based on different treatments of patterns extraction. This type of classification is used for forming attacks signatures and to detect anomalous behavior. The experiments with DARPA data set have shown that the proposed algorithm can detect the intrusive behaviour effectively. The experiments indicate that the patterns that we obtained are different from the command patterns. They are new patterns, can describe attacks more accurately, detect the attacks whose features appear only once, and offer a new idea for the research of intrusion detection. Also, we found that continuous sequences reflect a clean occurred sequences, while discontinuous patterns represent the sequences mixed with undesirable noisy data.

References

1. The survey is available at: www.csoonline.com/releases/ecrimewatch04.pdf
2. Kumar, S., Spafford, E.H.: A Software Architecture to Support Misuse Intrusion Detection. Proceedings of the 18th National Information Security Conference (1995) 194–204
3. Forrest, S., Hofmeyr, S.A., Somayaji, A., Logstaff, T.A.: A Sense of Self for Unix process, Proceedings of 1996 IEEE Symposium on Computer Security and Privacy (1996) 120–128
4. Ilgun, K., Kemmerer, R.A., Porras, P.A.: State Transition Analysis: A Rule-Based Intrusion Detection System. IEEE Transactions on Software Engineering, 21(3) (1995) 181–199

5. Javitz, H.S., Valdes, A.: The SRI IDES Statistical Anomaly Detector. In IEEE Symposium on Security and Privacy, Oakland, CA. SRI International (1991)
6. Axelsson, S.: Research in intrusion-detection systems: A survey. Technical report TR 98-17, Gteborg, Sweden: Department of Computer Engineering, Chalmers University of Technology (1999)
7. Hofmeyr, S.A., Somayaji, A., Forrest, S.: Intrusion Detection using Sequences of System Calls. Journal of Computer Security Vol. 6 (1998)
8. Fox, K.L., Henning, R.R., Reed, J.H., Simonian, R.P.: A neural network approach towards intrusion detection. Proceedings of 13th National Computer Security Conference, NIST, Baltimore, MD (1999) 125–134
9. Barbara, D., Couto, J., Jajodia, S., Wu, N.: ADAM: A testbed for exploring the use of data mining in intrusion detection, ACM SIGMOD Record, 30 (4) (2001)
10. Barbara, D., Couto, J., Jajodia, S., Wu, N.: An architecture for anomaly detection. D. Barbara and S. Jajodia (Eds.), Applications of Data Mining in Computer Security, Boston: Kluwer Academic (2002) 63–76
11. Lee, W., Stolfo, S.: Data Mining Approaches for Intrusion Detection. Proc. of the 7th USENIX Secunity Symposium (1998)
12. Lee, W., Stolfo, S.: A Data Mining Framework for Building Intrusion Detection Models, IEEE Symposium on Security and Privacy (1999)
13. Barbara, D., Couto, J., Wu, N.: ADAM: Detecting Intrusion by Data Mining. Proc. of the 2th IEEE Information Assurance Workshop (2001)
14. Teng, H., Chen, K., Lu, S.: Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns, Proceedings, IEEE Symposium on Research in Computer Security and Privacy (1990)
15. Kim, J.S., Lee, H.G., Seo, S., Ryu, K.H.: CTAR: Classification Based on Temporal Class-Association Rules for Intrusion Detection, WISA 2003, Lecture Notes in Computer Science Publisher: Springer-Verlag, Vol 2908/2003 (2003) 84–96
16. Lee, W.: A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems, Ph.D. Thesis, Computer Science Department, Columbia University, New York, NY. June (1999)
17. DARPA data set: www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.html
18. Agrawal, R., Imielinski, T., Swami, V.: Mining association rules between sets of items in large databases. P. Buneman and S. Jajodia, editors, Proceedings of the ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C. (1993) 207–216
19. Lippmann, R.P.: Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation. Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, Vol. 2