Host-Based Intrusion Detection Using Dynamic and Static Behavioral Models

Dit-Yan Yeung Yuxin Ding

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
dyyeung@cs.ust.hk

November 22, 2001

Abstract

Intrusion detection has emerged as an important approach to network security. In this paper, we adopt an anomaly detection approach by detecting possible intrusions based on program or user profiles built from normal usage data. In particular, program profiles based on Unix system calls and user profiles based on Unix shell commands are modeled using two different types of behavioral models for data mining. The dynamic modeling approach is based on *hidden Markov models* (HMM) and the principle of *maximum likelihood*, while the static modeling approach is based on event occurrence frequency distributions and the principle of *minimum cross entropy*. The *novelty detection* approach is adopted to estimate the model parameters using normal training data only, as opposed to the classification approach which has to use both normal and intrusion data for training. To determine whether or not a certain behavior is similar enough to the normal model and hence should be classified as normal, we use a scheme that can be justified from the perspective of hypothesis testing. Our experimental results show that the dynamic modeling approach is better than the static modeling approach for the system call datasets, while the dynamic modeling approach is worse for the shell command datasets. Moreover, the static modeling approach is similar in performance to instance-based learning reported previously by others for the same shell command database but with much higher computational and storage requirements than our method.

Keywords: anomaly detection, computer security, data mining, hidden Markov model, intrusion detection, maximum likelihood, minimum cross entropy, profiling, shell command, system call

1 Intrusion Detection Problems

Intrusion detection, which refers to a certain class of system attack detection problems, is a relatively new area in computer and information security. Many intrusion detection systems built thus far are based on the general model proposed by Denning in a seminal paper [1]. From a high-level view, the goal is to find out whether or not a system is operating normally. Abnormality or anomaly in the system behavior may indicate the occurrence of system intrusions that are the consequences of successful exploitation of system vulnerabilities.

One aspect for the categorization of intrusion detection systems is the target environment for detection, which is related to where such systems are used. *Host-based* intrusion detection systems detect possible attacks into individual computers on which the intrusion detection systems run. Such systems typically make use of information specific to the operating systems of the target computers. On the other hand, *network-based* intrusion detection systems monitor network behavior by examining the content as well as the format of network data packets, which typically are not specific to the exact operating systems used by individual computers as long as these computers can communicate among themselves using the same network protocol. For both types of intrusion detection systems, one may use a *data mining* approach by "mining" through the host-based or network-based data collected to detect possible attacks from internal or external intruders.

Another aspect for categorization is the modeled behavior for detection, which is related to the methods used for implementing such systems. *Misuse detection* systems detect evidence of attacks based on knowledge about abnormal behavior acquired from known attacks. *Anomaly detection* systems, on the other hand, model normal system behavior to provide a reference against which deviations are detected and alerted as possible intrusions. In other words, the major difference between the two approaches is on whether normal or abnormal (i.e., intrusive) behavior is modeled explicitly. Misuse detection and anomaly detection systems are sometimes also referred to as knowledge-based and behavior-based systems, respectively [2].

Since misuse detection systems typically require known intrusive scenarios to be hand-coded *a priori* and such scenarios are usually very specific to the operating systems if this approach is used for host-based systems, the anomaly detection approach is often the preferred choice for host-based intrusion detection systems because many pattern recognition and machine learning methods, such as density estimation methods [3], may be used for modeling normal system behavior.

In this paper, data mining methods based on the anomaly detection approach are proposed for host-based intrusion detection. We consider both normal program profiling based on *system calls* [4, 5, 6, 7, 8] and normal user profiling based on *shell commands* [9, 10, 11, 12, 13, 14]. In operating systems such as Unix, the privileged processes that only the superuser is authorized to execute make use of system calls to access privileged system resources or services. Thus such privileged processes are often a major target for intruders. To monitor user behavior on systems that accept shell commands from users, shell command sequences from audit logs can be used for user profiling.

The remainder of this paper is organized as follows. We will first discuss in Section 2 two different paradigms for building behavioral models, followed by discussions on two general categories

of behavioral models in Section 3. We will then present two specific methods, one based on the dynamic modeling approach (Section 4) and the other based on the static modeling approach (Section 5). In Section 6, the system call and shell command datasets [8, 12] used in our experiments will be described. Methods for preprocessing the data for use by the dynamic and static models will also be explained. In Section 7, we will present details of the model construction and performance evaluation aspects. Justification of the procedure from a hypothesis testing perspective will be given. We will then present some experimental results in Section 8. In Section 9, our contributions will be summarized and some possible future research issues will be outlined.

2 Classification versus Novelty Detection

Typical *classification* problems studied in pattern recognition can be formulated as building a classifier that classifies each pattern into one of $c \geq 2$ classes with as low classification error as possible. To build such a discriminative classifier, training examples from all c classes are needed. If the classifier is built using machine learning techniques, this approach is often referred to as *supervised learning*.

While this formulation is commonly used in pattern recognition, there also exists another formulation, called *novelty detection* [15, 16, 17], which is much less explored than classification. Simply put, novelty detection refers to the detection of novel or abnormal events or patterns. In a probabilistic sense, it is equivalent to deciding whether an unknown test pattern is produced by the underlying data distribution that corresponds to the training set of normal patterns. While novelty detection problems appear to be similar to 2-class classification problems, with the two classes corresponding to normal and abnormal patterns respectively, a major difference is that novelty detection methods typically use only training examples from the class corresponding to normal patterns to build a generative model of normal behavior. Table 1 summarizes this major difference. The novelty detection approach is particularly attractive under situations where novel or abnormal patterns are expensive or difficult to obtain for model construction. If the model is built using machine learning techniques, we refer to this as an *unsupervised learning* approach. In this paper, the novelty detection approach is adopted because it is superior to the classification approach when intrusion data are scarce.

3 Dynamic versus Static Behavioral Models

Normal program or user behaviors are profiled by building behavioral models using data collected from normal operations. There are generally two categories of behavioral models. *Dynamic models* explicitly model temporal variations that are essential for discriminating abnormal system behavior from normal behavior. *Static models*, on the other hand, do not explicitly model temporal variations. They could be used for anomaly detection problems if the normal system behavior does not exhibit significant temporal variations, or if the temporal sequences are first converted into some non-temporal representation typically in the form of multidimensional feature vectors with no time dimension. In general, dynamic models are more powerful in representing subtle temporal regularities and hence should be used if possible.

Different anomaly detection methods have been applied. They include instance-based learning [18, 12], multi-layer perceptrons [5, 9], decision trees [10], hidden Markov models (HMM) [11, 8], frequent episodes [13], correlation analysis [4], statistical likelihood analysis [5], rule learning [6, 7, 8], and uniqueness method [14]. Of these methods, only HMMs are intrinsically dynamic in nature. Also, not all of them are based on the preferred novelty detection approach and hence they have to make use of both normal and intrusion data for model construction.

In this paper, intrusion detection systems based on profiling system call sequences and shell command sequences are first studied with the dynamic modeling approach using HMMs. These systems can be seen as extensions and variants of previous HMM-based intrusion detection systems. Afterwards, we will propose an information-theoretic static modeling approach based on the usage frequencies of system calls or shell commands. Comparative studies of the two modeling approaches on different intrusion detection problems under different operating conditions are then performed.

4 Dynamic Modeling Approach Based on Hidden Markov Models and Maximum Likelihood

4.1 Hidden Markov Models

HMMs are stochastic models of sequential data that have been used successfully for many applications in knowledge discovery, pattern recognition, and speech recognition. Each HMM contains a finite number of unobservable (or hidden) states. State transitions are governed by a stochastic process to form a Markov chain, i.e., a stochastic finite state machine. At each state, some state-dependent events can be observed. The emission probabilities of these observable events are determined by a probability distribution, one for each state. Of interest here are discrete HMMs in which the observed events (system calls or shell commands) are discrete symbols, as opposed to other models not studied here, called continuous-density HMMs.

Based on the connectivity topology between states, two types of HMMs can be distinguished [19]. Fully-connected or ergodic HMMs allow state transitions between all state pairs. On the other hand, left-to-right HMMs do not allow state transition back to any state to the left of the current state. In fact, most left-to-right HMMs used in practice only allow state transition from a state to itself (called self-transition), to the immediate neighbor to the right, and to the neighbor two steps to the right. In this paper, our left-to-right HMMs are further restricted to only the first two types of state transition, as shown in Figure 1 below.

To estimate the parameters of an HMM for modeling normal system behavior, sequences of normal events (system calls or shell commands in our case) collected from normal system usage are used as training examples. An *expectation-maximization* (EM) algorithm [20] known as the *Baum-Welch re-estimation algorithm* [21] for mixture density estimation is used to find the maximum-likelihood (ML) parameter estimate. More details of the algorithm can be found in [19].

4.2 Sample Likelihood With Respect to Model

Given a trained HMM M, the sample likelihood of an observation sequence S with respect to M can be computed using either the *forward algorithm* or the *backward algorithm* [19]. From a generative point of view, this can be seen as computing the probability that a given observation sequence is generated by the model. Alternatively, we can also consider it as providing a quantitative measure for assessing how well the model matches the sequence.

Ideally, a well-trained HMM can give sufficiently high likelihood values only for sequences that correspond to normal behavior. Sequences that correspond to intrusive behavior should give significantly lower likelihood values. By comparing the sample likelihood of S with respect to M against a certain threshold, one can decide whether S deviates significantly from M and hence should be considered a possible intrusion. We will describe how to determine the threshold in Section 7.3.

5 Static Modeling Approach Based on Occurrence Frequency Distributions and Minimum Cross Entropy

5.1 Occurrence Frequency Distributions

Suppose the occurrence frequencies of different events (system calls or shell commands) are measured during a certain period of time. A probability distribution (i.e., a probability mass function defined over the space of all possible events under consideration) can be used to represent the overall occurrence pattern during that period. Since the order in which different events occur is not taken into account in the distribution, we refer to this as a static modeling method. Using this representation scheme, a normal program or user behavioral model is simply represented as occurrence frequency distribution, which is the basis on which possible system intrusions can be detected.

Let $P(M)$ denote the probability distribution characterizing the behavior of a normal model M and let $P_i(M), i = 1, 2, \ldots, N$ denote the occurrence probability of event i among a total of N possible events. Similarly, $Q(S)$ and $Q_i(S), i = 1, 2, \ldots, N$ denote the probability distribution and individual event probabilities, respectively, for some behavior S being monitored. In what follows, the dependencies on M and S are not explicitly shown for the sake of notational simplicity.

5.2 Cross Entropy Between Distributions

To characterize how different two distributions P and Q are, we need a measure for quantifying the dissimilarity between them. An information-theoretic measure that can serve this purpose is known as *cross entropy* [22, 23], which is also related to *Kullback-Leibler information measure* [24].

For our purpose, we use the following definition of cross entropy:

$$C(P,Q) = \sum_{i=1}^{N} (Q_i - P_i) \log \frac{Q_i}{P_i}$$

Note that changing the roles of P and Q does not affect this measure, i.e., the measure is symmetric with respect to the two distributions involved:

$$C(P,Q) = C(Q,P)$$

Moreover, it can be shown that the following two properties always hold:

$$\begin{aligned} &C(P,Q) \geq 0 \\ &C(P,Q) = 0 \quad \Leftrightarrow \quad P = Q \end{aligned}$$

Thus, by checking whether the cross entropy between P and Q is larger than a certain threshold, one can decide whether S should be considered a possible intrusion with respect to the model M. We will describe how to determine the threshold later in Section 7.3.

6 Data Preprocessing and Partitioning

6.1 Preprocessing of System Call Data

The system call datasets are available at the public-domain archive in the Department of Computer Science of the University of New Mexico.[1] In this paper, we report results for four Unix programs: `ps`, `login`, `named`, and `sendmail`. In general, executing a single program may lead to multiple processes. For system calls that are issued by the same process, we group them together to form a *trace*. For example, Table 2 shows several system calls together with the corresponding process IDs. The system calls can be grouped into two traces: 13 15 13 16 ... and 16 14 .... In general, different traces are of different lengths.

The intrusion data were generated by running programs intruded according to public advisories posted on the Internet. Both `ps` and `login` used Trojan intrusions, which allow unauthorized access to the system through a built-in "back-door". For the `named` data, the intrusion used was buffer overflow [8]. For `sendmail`, the intrusions were `sunsendmailcp` and a decode alias attack [4].

Note that the ground truth of intrusion data is more difficult to vet than one might expect. Although an intruded program contains anomalous code, a particular execution of the program may not involve the anomalous code at all and hence it can still generate a trace of system calls that is entirely normal. In fact, as will be discussed later in Section 8.1.1, we have found evidence in the datasets (with the exception of `ps`) that could possibly be attributed to this. Ideally one would like to have intrusion datasets that contain truly intrusive data only. However, this would require the implementation of some filtering mechanism for the data generated from an intruded program, which in general is not easy to materialize.

[1] The URL is `http://www.cs.unm.edu/~immsec/data/`.

6.2 Preprocessing of Shell Command Data

The shell command datasets are available at the public-domain KDD archive maintained by the Department of Information and Computer Science of the University of California at Irvine.[2] They were contributed by some researchers from Purdue University [12]. Since it is difficult to obtain real intrusion data, only normal data were collected via the history file mechanism from eight different Unix users over a period of more than two years. For each user login session (i.e., from login to logout), each word typed by the user was recorded as a token. Since many Unix commands are followed by parameters (e.g., `ls -laF Paper Notes letterhead.tex`), the set of all distinct tokens would become too large to be manageable. To reduce the size of the token set, only a count of the files or directories is represented as a token instead of the actual file or directory names (e.g., `ls -laF <3>`). Similar to the case of system calls, all the tokens in a login session also form a *trace*.

Note that the datasets contain no real intrusion data because it is difficult to collect such data in real applications. This holds in general for this kind of intrusion detection problems. In our experiments, (normal) data from other users were used as if they were "intrusive" data for a given user. Thus, by its very nature, this problem is more like a classification problem than a novelty detection problem, although we still use a novelty detection approach as it is more desirable in practice.

6.3 Partitioning of Datasets

In general, each set of data is partitioned into three subsets that are used for different purposes:

1. Training set (normal data only)
2. Threshold determination set (normal data only)
3. Test set (both normal and intrusion data)

The *training set* of data is for estimating the parameters of a behavioral model. Only normal data are needed when the novelty detection approach is used. As the model is built using normal data only, we need a criterion to decide when a new behavior observed should be considered normal or intrusive. In particular, it corresponds to finding a threshold for some similarity (e.g., likelihood) or dissimilarity (e.g., cross entropy) measure. The *threshold determination set* (cf., *validation set* for cross validation in statistics) of normal data is used for determining this threshold.[3]

After the model parameters and the threshold have been estimated using the training and threshold determination sets, respectively, the test set can be used for evaluating the performance of the model. More details about the performance measures used will be discussed in Section 7.1.

Table 3 shows the dataset sizes of the system call data used in our experiments. Since the available data for programs `ps` and `login` are quite limited and the similarity between normal

[2] The URL is `http://kdd.ics.uci.edu/databases/UNIX_user_data/UNIX_user_data.html`.

[3] Determining the threshold using a set of data different from the training and test sets is just a special case of model selection. In general, the threshold determination set is called validation set or parameter selection set.

traces is usually quite high for system call data as confirmed by our preliminary investigations, we used all the normal data for training with no separate threshold determination set and test set. We tried to use a separate threshold determination set for `named` but we found that the result was better by using the training set to determine the threshold. However, separate test sets were available for programs `named` and `sendmail`.

Table 4 summarizes the dataset sizes of the shell command data used in our experiments. As discussed above, for each user, the (normal) data of all other users were treated as if they were "intrusive" data for that user. Whereas the test data of a user can be used for measuring the false detection rate (FDR), the test data of all other users can be used for measuring the true detection rate (TDR). (These two performance measures will be explained in detail in Section 7.1.) Since the available datasets are quite large, we used disjoint sets of data for training, threshold determination, and testing. Partitioning of the datasets is as follows. For each user, the shell command traces recorded in chronological order are partitioned into two groups such that the tokens in the first group are roughly twice as many as those in the second group. The second group forms the test set. To minimize the differences between the training and threshold determination sets, the traces in the first group are assigned to two sets in an interleaved manner, i.e., the odd-numbered traces are assigned to one set and the even-numbered traces are assigned to another set. Thus the two sets have roughly the same number of traces. Table 5 shows the number of distinct tokens found in the data for each user. When the datasets for all eight users are combined together, the total number of distinct tokens is equal to 2356.

7 Model Construction and Performance Evaluation

7.1 Performance Criteria

For performance evaluation, we define two measures, namely, *true detection rate* (TDR) and *false detection rate* (FDR):

$$\begin{aligned} \text{TDR} &= \Pr(\text{intrusive} \mid \text{intrusive}) \\ &= \frac{\text{number of intrusive testing traces detected as intrusive}}{\text{number of intrusive traces in test set}} \end{aligned}$$

$$\begin{aligned} \text{FDR} &= \Pr(\text{intrusive} \mid \text{normal}) \\ &= \frac{\text{number of normal testing traces detected as intrusive}}{\text{number of normal traces in test set}} \end{aligned}$$

In other words, TDR is the probability that an intrusive trace is correctly detected, and FDR is the probability that a normal trace is incorrectly reported as intrusive. We prefer these two measures because both relate reporting the occurrence of an intrusive event to the ground truth (i.e., normal or intrusive nature) of that event. This is in line with the convention used in [8] although they refer to the two measures as *true positives* and *false positives*, respectively. We use the term 'detection' to make the meaning of detecting intrusions more explicit. *Hit rate* and *false alarm rate* can also be used in place of TDR and FDR, respectively. Note that the commonly used term, called *false acceptance rate* or *false negatives*, for expressing $\Pr(\text{normal} \mid \text{intrusive})$ can be computed simply by subtracting TDR from 1.

7.2 Model Training

To train an HMM, fixed-length sequences of events are extracted from each trace of the training set by moving a window of the specified width (i.e., sequence length) through the entire trace with a step size of 1. Identical sequences extracted are represented by only a single copy in the training set. Both fully-connected and left-to-right HMMs were used in our experiments. In what follows, we will refer to these two types of HMMs as FC-HMM and LR-HMM, respectively.

For the static modeling approach, all traces from the training set are used to create a distribution-based behavioral model.

7.3 Threshold Determination

After the parameters of a model have been estimated from the training data, the threshold determination set is used to determine an appropriate threshold which will subsequently be used as a criterion for detecting possible intrusions.

For HMM-based dynamic modeling, fixed-length sequences are extracted from each trace of the threshold determination set in the same way as before for the training data. The sample likelihood of each sequence with respect to the model can then be computed. For the static modeling approach, each trace of the threshold determination set is used to compute a distribution as well as the cross entropy between this distribution and the reference distribution computed based on the training data.

For each chosen FDR for the threshold determination set, a corresponding threshold value can be obtained. Note that in the case of HMMs, a trace is said to be intrusive if it contains at least one intrusive sequence. In our experiments, different threshold values were tried by choosing different FDR values.

7.4 Model Testing

To test whether a trace in the test set is intrusive, fixed-length sequences extracted from the trace are presented to a trained HMM to compute the likelihood values. If at least one sequence has a likelihood value that is lower than the threshold, the trace is said to be intrusive. In other words, we can conclude that a trace under investigation is intrusive as soon as the first intrusive sequence is found inside the trace, even though the end of the trace has not been reached.

In the case of the static modeling approach, in order to perform timely detection of possible intrusions, it would be desirable if a decision could be made as soon as sufficient data have been collected to compute a reasonably reliable distribution. Since a trace may take a very long time to complete (if a program that generates system calls runs for a long time or if a user login session is long), we do not want to wait until the end of the trace to make a decision. Instead, a distribution is computed for each sub-trace sequence. The cross entropy between this distribution and the reference distribution of the model computed based on the training data will be compared with the threshold to determine whether it is an intrusive sequence. The extraction of variable-length sequences from a trace is illustrated in Figure 2 below.

The detection of possible intrusions in a trace can be performed immediately after the first K events (i.e., system calls or shell command tokens) have arrived. We refer to K as the *minimum sequence length*. If this value is small, it implies that possible intrusions can be detected with small time delay and hence is favorable. However, the value cannot be set too small or else there is insufficient information for making reliable decisions. Thus the choice of an appropriate value for K has to be a tradeoff between these two considerations.

7.5 Hypothesis Testing Perspective

In this section, we will justify the scheme above from a hypothesis testing perspective. Although our explanation is based on HMMs, it also holds for the information-theoretic static modeling method based on cross entropy.

Let M denote an HMM learned from the training data. Given a sequence S from the test set, we want to decide whether it is likely to be generated by M. In other words, we want to determine whether S is a normal sequence. This problem can be formulated as applying a statistical test [25]. Let us generate a sufficiently large sample $\mathcal{R}$ of (normal) sequences from M. For an arbitrary sequence R in $\mathcal{R}$, the log-likelihood of R with respect to M is denoted as L_R, which is equal to $\log \Pr(R \mid M)$. Similarly, the log-likelihood of S is denoted as L_S, which is equal to $\log \Pr(S \mid M)$. Based on the empirical probability distribution of $\log \Pr(R \mid M)$ over the sample $\mathcal{R}$, we then test the hypothesis that L_S is drawn from the probability distribution of the log-likelihood of the sequences in $\mathcal{R}$, i.e.

$$\Pr(L_R \leq L_S) > \psi$$

for some threshold $0 < \psi < 1$. We reject the null hypothesis if the probability is not greater than ψ, implying that S is not a normal sequence with respect to model M.

In our case, the threshold determination set of normal data plays the role of $\mathcal{R}$ although $\mathcal{R}$ is not actually generated by M. If M is a well-trained model representing the training set, the underlying distributions of the training and threshold determination sets are close enough to each other, and the threshold determination set is sufficiently large, then it is not unreasonable to use the threshold determination set as $\mathcal{R}$. Apparently, we can see that the threshold ψ is just the FDR chosen for the threshold determination set.

8 Experimental Results and Discussions

8.1 Experiments for System Call Data

8.1.1 Results

Experiments were conducted on the system call data using both the dynamic modeling approach (FC-HMM and LR-HMM) and the static modeling approach. Tables 6 and 7 below show some results for the programs `ps`, `login`, `named`, and `sendmail`.

Different choices of sequence length and number of states were tried for HMMs. For each sequence length chosen, the HMM with the smallest number of states that maximizes the TDR is shown. For FC-HMM, the maximum number of states tested is approximately equal to the total number of different system call categories in the corresponding program (30 for `ps`, 48 for `login`, 50 for `named`, and 60 for `sendmail`). We found that FC-HMM almost had no discrimination power in detecting intrusions when the number of states was set too small (smaller than 6). Generally speaking, the performance could be improved by increasing the sequence length and the number of states, although this was not always the case. For LR-HMM, the maximum number of states tested is equal to the sequence length. The performance was found to be very sensitive to the sequence length. Increasing the sequence length always improved the discrimination power of the model. Similarly, for the static modeling method, different values of minimum sequence length were tried.

Recall that all normal data for `ps` and `login` were used for training, leaving no separate data for threshold determination. For HMMs, the threshold was chosen to be the minimum likelihood among all training sequences. For the case of distribution-based static modeling, a cross entropy value was computed between the entire training set and each trace in the training set. The threshold was chosen to be the maximum cross entropy among all traces in the training set. Thus the FDR of the training set is always equal to 0. Since no separate normal data were available for testing, the FDR entries for these two programs are marked as '-' in Table 6.

For `named`, since the traces within a set are all very similar, we found that a slight change of the threshold value could lead to a great change in TDR. As a result, we determined the threshold using the training set instead of a separate threshold determination set of normal data.

For `sendmail`, we determined the threshold using a separate set of normal data (i.e., threshold determination set). Four different threshold values were chosen for each model configuration by making the FDR of the threshold determination set equal to 5%, 10%, 15%, and 20%. We found that the FDR of the test set was always very close to that of the threshold determination set. As discussed before, this practice of using a separate set of normal data to determine the threshold can be justified from a hypothesis testing perspective and should be used if at all possible. Hence `sendmail` is a good example of the general case for the system call data.

We found that the TDR for `login` and `named` could never go beyond 77.8% and 60.0%, respectively. After examining the training set (with normal traces only) and the instruive traces in the test set of each program, we discovered that the intrusive traces that failed to be detected as intrusive are in fact identical to some normal traces in the training set. As mentioned in Section 6 above, a possible reason for this is that sometimes the execution of an intruded program can still generate a normal trace if the execution does not involve any anomalous part of the program. Thus the above TDR values are the best that one could obtain. For the `sendmail` data, we also found one intrusive trace to be identical to one normal training trace.

8.1.2 Discussions

From the experimental results shown above, we can see that the information-theoretic static modeling method was consistently inferior to HMM-based dynamic modeling for the intrusion detection problem involving system calls. In particular, the model constructed for the `ps` pro-

gram was unable to detect any intrusive trace. We speculate two possible reasons for this. First, different traces from the same dataset are usually quite similar in their distributions, making it difficult to distinguish normal traces from intrusive ones simply by basing on the cross entropy values induced by different distributions. Second, as system calls are generated through the execution of a program which can be modeled by a finite state machine, the temporal dependencies between system calls are salient features for intrusion detection and hence should be captured using some dynamic modeling techniques.

8.2 Experiments for Shell Command Data

8.2.1 Results

Tables 8–10 show the results for the shell command data using three different methods.

For each method, only two choices of the sequence length or minimum sequence length are included in the table to illustrate the effect of varying the parameter, although more choices were actually tried in our experiments. As before for the `sendmail` example, the threshold was chosen in such a way that the FDR of the threshold determination set was equal to some prespecified value (5%, 10%, 15% or 20%). For each chosen FDR value, the TDR shown is the average taken over the individual TDR values by treating the data from other users as intrusion data. The number of states shown is the minimum value that maximizes the TDR.

Increasing the sequence length always increased the discrimination power of both FC-HMM and LR-HMM in detecting intrusions. Since traces shorter than the sequence length chosen were eliminated and there exist many short shell command traces in the datasets corresponding to short login sessions (unlike system call traces which are rather long typically with more than 300 system calls per trace), increasing the sequence length had the consequence of eliminating the shorter traces which could be partially responsible for the performance improvement because these traces could not model the behavior well. This is also a possible reason for the observed performance improvement of the static modeling method as the minimum sequence length increases.

8.2.2 Discussions

In our experiments, the information-theoretic static modeling method performed significantly better than both FC-HMM and LR-HMM, typically 10%-20% higher in the TDR. A possible reason is that the temporal dependencies between shell commands are much weaker and hence are not particularly useful for intrusion detection. Instead, the static shell command distribution seems to be sufficient for many users.

8.3 General Discussions

From our experiments, we conclude that the HMM-based dynamic modeling approach is better suited for the intrusion detection problem based on system calls, but the information-theoretic

static modeling approach is a better choice for that based on shell commands.

Although FC-HMM usually gives slightly better performance than LR-HMM, increasing the number of states in an LR-HMM can approach the performance of an FC-HMM with fewer states. For example, the FC-HMM with sequence length 30 in Table 8 is similar in performance to the LR-HMM with sequence length 50 in Table 9. Note that the time complexity of each training iteration of an FC-HMM is $O(W^2T)$, where W denotes the number of states and T denotes the sequence length. As a comparison, the time complexity of each training iteration of an LR-HMM is only $O(WT)$.

We also measured the CPU execution time for different methods. All the tasks were run on an UltraSPARC 30 workstation with 256MB memory. Tables 11–13 show the CPU time required for the training and testing stages measured for each user in the experiments as reported before (Tables 8–10). It can be seen that LR-HMM is faster than FC-HMM for both the training and testing stages. Our information-theoretic static modeling method based on cross entropy is impressive in that its training time is always negligible because it simply requires the computation of a distribution based on the training data. The testing time is also comparable to that for HMMs. Our method would be particularly attractive if new models have to be built regularly due to frequent changes in the system behavior.

8.4 Comparison with Previous Work

We also compared our results with those from previous work. To facilitate comparison, we performed another experiment using the same experimental setup as that used by [18, 12]. The datasets were partitioned into training, parameter selection (or threshold determination in our case), and test sets as shown in Table 14 below. Moreover, in their work, the TDR and FDR were computed based on sequences. We think it makes more sense to measure TDR and FDR according to traces as in our experimental results reported earlier. However, to facilitate comparison here, we used the same scheme as theirs for this experiment.

Table 15 shows the classification results obtained by [18] using *instance-based learning* (IBL), giving an average TDR of 34.2% with an average FDR of 5.3%. Table 16 shows the classification results obtained by us using the static modeling method based on cross entropy. The average TDR is 35.6% at an average FDR of 5.5%. Thus, it can be concluded that the two methods can achieve very similar performance in terms of the TDR and FDR measures.

It should be noted, however, that there are major differences between the two methods in terms of computational and storage requirements. Apparently IBL has much higher storage requirement because all training examples have to be explicitly stored. Also, each unknown test case has to be matched against all the stored examples and hence the computational overhead is also very high. Although data reduction techniques can alleviate the problems to a certain extent, the high computational and storage requirements are still the major limitations of IBL methods. Our method is clearly superior in this aspect because the training examples are summarized as a distribution, the storage requirement of which does not depend on the size of the training set. Similarly, during testing, the computational requirement is very low as discussed above.

9 Conclusion

9.1 Concluding Remarks

In this paper, we have presented two different anomaly detection approaches for two different host-based intrusion detection problems. For the intrusion detection problem involving system call sequences, the use of dynamic behavioral models is superior. A possible reason is that temporal dependencies are salient features for this problem. This is in line with other sequence learning problems with subtle temporal relationships, which have found HMMs to be among the best methods. On the other hand, the use of static behavioral models can give better results for the intrusion detection problem involving shell command sequences. It can be speculated that temporal dependencies are not very useful or may even be harmful for this problem. Our information-theoretic static modeling approach based on cross entropy is simple and computationally cheap, yet it can outperform the more sophisticated dynamic modeling approach based on HMMs. A lesson to learn is that one should be careful in finding the best match between problems and methods. Apparently the static modeling approach based on cross entropy is a better match to the problem than the dynamic modeling approach based on HMMs.

9.2 Contributions

The contributions of this paper are two-fold. First, although HMMs and cross entropy are not new, using them for solving novelty detection problems (as opposed to classification problems) is still an area that is far from being sufficiently explored. By formulating the detection problem under a hypothesis testing framework, this paper presents and demonstrates the use of theoretically justified methods for solving novelty detection problems. It is our hope that this effort could lead to more research along the same line.

Second, intrusion detection is an important topic with significant practical implications. Our contribution to this application area is that we have proposed some practically feasible methods for solving two types of host-based intrusion detection problems with extensive experiments performed on real-world data. This serves as an effort to broaden the applications of pattern recognition techniques.

9.3 Future Research

A closer look at Tables 15 and 16 above reveals the fact that IBL is better for some users (0, 3, 4, 6) while the cross-entropy method is better for other users (1, 2, 5, 7). This shows that the two methods are complementary to each other. A potential future research direction is to combine these two methods and possibly also some other methods to further improve the discrimination power. In addition to host-based intrusion detection problems, we are also conducting research on network-based intrusion detection. Some of the ideas learned from the current research may also be relevant to network-based intrusion detection.

Acknowledgments

The research reported in this paper has been supported in part by the Hong Kong Innovation and Technology Commission (ITC) under project AF/223/98 and the Hong Kong University Grants Committee (UGC) under Areas of Excellence research grant AoE98/99.EG01.

References

[1] D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[2] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–922, 1999.

[3] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, New York, NY, USA, 2nd edition, 2001.

[4] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for Unix processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA, 6–8 May 1996.

[5] D. Endler. Intrusion detection: applying machine learning to Solaris audit data. In *Proceedings of the Fourteenth Annual Computer Security Applications Conference*, pages 268–279, Phoenix, AZ, USA, 7–11 December 1998.

[6] G.G. Helmer, J.S.K. Wong, V. Honavar, and L. Miller. Intelligent agents for intrusion detection. In *Proceedings of the 1998 IEEE Information Technology Conference - Information Environment for the Future*, pages 121–124, Syracuse, NY, USA, 1-3 September 1998.

[7] W. Lee and S.J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the Seventh USENIX Security Symposium*, pages 79–93, San Antonio, TX, USA, 26-29 January 1998.

[8] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 133–145, Oakland, CA, USA, 9-12 May 1999.

[9] J. Ryan, M.J. Lin, and R. Miikkulainen. Intrusion detection with neural networks. In M.I. Jordan, M.J. Kearns, and S.A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 943–949. MIT Press, 1998.

[10] D. Gunetti and G. Ruffo. Intrusion detection through behavioral data. In *Proceedings of the Third International Symposium on Intelligent Data Analysis*, pages 383–394, Amsterdam, Netherlands, 9-11 August 1999.

[11] T. Lane. Hidden Markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden, 31 July 1999.

[12] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[13] W. Lee, S.J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–132, Oakland, CA, USA, 9-12 May 1999.

[14] M. Schonlau and M. Theus. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1/2):33–38, 2000.

[15] W.J. Daunicht. Autoassociation and novelty detection by neuromechanics. *Science*, 253(5025):1289–1291, 1991.

[16] C.M. Bishop. Novelty detection and neural network validation. *IEE Proceedings: Vision, Image and Signal Processing*, 141(4):217–222, 1994.

[17] N. Japkowicz, C. Myers, and M. Gluck. A novelty detection approach to classification. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 518–523, Montréal, Quebec, Canada, 20–25 August 1995.

[18] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158, San Francisco, CA, USA, 2–5 November 1998.

[19] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[20] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[21] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[22] J.E. Shore and R.W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26(1):26–37, 1980.

[23] R.W. Johnson and J.E. Shore. Comments on and correction to 'axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy' (Jan 80 26–37). *IEEE Transactions on Information Theory*, 29(6):942–943, 1983.

[24] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[25] P.R. Cohen. *Empirical Methods for Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1995.

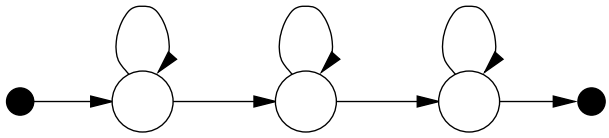

Figure 1: Left-to-right HMM with two state transition types

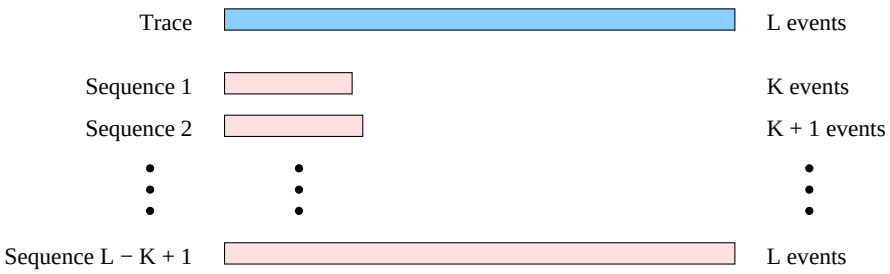

Figure 2: Extraction of variable-length sequences from a trace

Table 1: Classification vs. novelty detection in terms of data usage

Classification Paradigm			
	Training set	Validation set	Test set
Class 1 (normal data)	√	√	√
Class 2 (abnormal data)	√	√	√

Novelty Detection Paradigm			
	Training set	Validation set	Test set
Class 1 (normal data)	√	√	√
Class 2 (abnormal data)			√

Table 2: An example with system calls forming two traces

Process ID	System call
1	13
1	15
2	16
1	13
2	14
1	16

Table 3: Data partitioning for system call datasets

		Program			
		`ps`	`login`	`named`	`sendmail`
No. of system call categories		22	46	46	53
Training set (normal)	No. of traces	24	12	8	135
	No. of system calls	6144	8894	677340	154159
Threshold determination set (normal)	No. of traces	(training set)			91
	No. of system calls	(training set)			832364
Test set (normal)	No. of traces	(training set)		12	120
	No. of system calls	(training set)		7690572	813241
Test set (intrusive)	No. of traces	26	9	5	13
	No. of system calls	6968	4853	1800	3688

Table 4: Data partitioning for shell command datasets

User	Training set		Threshold determination set		Test set	
	No. of traces	No. of tokens	No. of traces	No. of tokens	No. of traces	No. of tokens
0	171	5733	170	6802	147	6316
1	196	5702	194	5327	365	5571
2	134	6776	134	3844	216	5120
3	313	13626	312	10309	286	11970
4	213	10826	212	11850	121	10981
5	832	20285	831	18009	762	19020
6	419	4485	418	5062	502	4738
7	562	16784	562	16586	466	16706

Table 5: Number of distinct tokens for each user

User	No. of distinct tokens in training set	Total no. of distinct tokens
0	151	286
1	152	308
2	174	484
3	291	476
4	375	561
5	360	607
6	228	447
7	406	704

Table 6: Results for system call data (`ps`, `login`, `named`)

Program	Model	Sequence length	No. of states	Minimum sequence length	TDR (%)	FDR (%)
`ps`	FC-HMM	6	30	-	100	-
		10	20	-	96.2	-
		20	20	-	61.5	-
		28	8	-	92.3	-
	LR-HMM	20	12	-	65.4	-
		28	16	-	100	-
	Cross entropy	-	-	30	0	-
		-	-	50	0	-
`login`	FC-HMM	6	48	-	77.8	-
		10	48	-	77.8	-
		20	38	-	66.7	-
	LR-HMM	30	8	-	55.6	-
		40	20	-	77.8	-
		50	40	-	77.8	-
	Cross entropy	-	-	30	55.6	-
		-	-	50	55.6	-
`named`	FC-HMM	6	50	-	60.0	0
		12	40	-	60.0	0
		20	50	-	60.0	0
	LR-HMM	20	8	-	60.0	0
		30	12	-	60.0	0
	Cross entropy	-	-	30	60.0	0
		-	-	50	60.0	0

Table 7: Results for system call data (`sendmail`)

Program	Model	Sequence length	No. of states	Minimum sequence length	TDR (%) FDR =5%	FDR =10%	FDR =15%	FDR =20%
`sendmail`	FC-HMM	6	10	-	61.5	84.6	84.6	92.3
		12	40	-	0	84.6	84.6	84.6
		20	20	-	0	91.7	91.7	91.7
	LR-HMM	20	17	-	0	66.7	66.7	84.6
		40	15	-	0	81.8	81.8	90.9
	Cross entropy	-	-	20	66.7	66.7	66.7	66.7
		-	-	40	72.7	72.7	72.7	72.7

Table 8: Results for shell command data (FC-HMM)

User	TDR (%) of FC-HMM (sequence length = 10)					TDR (%) of FC-HMM (sequence length = 30)				
	No. of states	FDR =5%	FDR =10%	FDR =15%	FDR =20%	No. of states	FDR =5%	FDR =10%	FDR =15%	FDR =20%
0	10	31.9	50.3	62.3	67.2	10	45.0	52.8	60.9	66.6
1	50	57.9	80.9	84.0	90.1	5	73.1	79.4	89.5	93.7
2	30	46.1	62.6	70.1	79.1	20	49.8	63.1	83.3	89.4
3	10	34.2	45.7	54.8	64.4	10	40.7	64.4	73.9	75.8
4	20	11.1	18.9	36.8	44.5	20	24.8	27.5	45.3	52.0
5	20	49.2	71.4	73.8	78.0	20	57.1	70.8	79.1	82.3
6	20	13.0	26.2	42.4	53.6	20	14.3	43.0	58.1	60.6
7	20	28.7	44.5	61.2	75.0	20	39.9	55.8	65.7	81.0
Average		34.0	50.1	60.7	69.0		43.1	57.1	69.5	75.2
Worst		11.1	18.9	36.8	44.5		14.3	27.5	45.3	52.0
Best		57.9	80.9	84.0	90.1		73.1	79.4	89.5	93.7

Table 9: Results for shell command data (LR-HMM)

User	TDR (%) of LR-HMM (sequence length = 30)					TDR (%) of LR-HMM (sequence length = 50)				
	No. of states	FDR =5%	FDR =10%	FDR =15%	FDR =20%	No. of states	FDR =5%	FDR =10%	FDR =15%	FDR =20%
0	5	40.8	48.2	64.0	68.0	5	45.1	60.3	64.7	68.3
1	10	71.3	79.1	87.2	89.7	30	73.5	81.4	82.4	83.8
2	5	43.2	62.1	79.6	88.0	10	61.8	69.4	79.4	82.9
3	10	23.9	59.9	71.7	76.4	10	35.5	58.5	73.4	78.4
4	20	21.1	22.4	37.4	49.4	20	14.8	36.0	38.8	52.3
5	10	59.0	67.7	77.5	83.5	5	49.6	63.3	69.7	76.3
6	10	12.0	39.8	50.3	55.8	10	15.5	19.7	21.1	44.6
7	5	32.7	47.2	56.8	74.7	10	52.9	63.6	72.4	75.0
Average		38.0	53.3	65.6	73.2		43.6	56.5	62.7	70.2
Worst		12.0	22.4	37.4	49.4		14.8	19.7	21.1	44.6
Best		71.3	79.1	87.2	89.7		73.5	81.4	82.4	83.8

Table 10: Results for shell command data (cross entropy)

User	TDR (%) of cross entropy (min. sequence length = 30)				TDR (%) of cross entropy (min. sequence length = 50)			
	FDR =5%	FDR =10%	FDR =15%	FDR =20%	FDR =5%	FDR =10%	FDR =15%	FDR =20%
0	52.9	62.7	79.3	82.2	46.2	78.5	80.5	81.1
1	56.0	81.2	93.2	95.4	54.4	71.4	89.8	92.7
2	43.3	49.4	73.4	95.9	43.3	49.4	73.4	96.0
3	46.8	76.5	90.7	95.1	46.8	76.5	88.5	95.1
4	48.4	76.1	85.7	88.0	55.0	73.9	81.0	82.2
5	56.7	74.3	78.7	82.8	50.3	70.2	81.2	89.4
6	25.6	44.0	57.2	58.8	36.1	40.1	56.0	60.0
7	60.8	85.7	93.9	97.1	83.4	95.4	97.6	99.5
Average	48.8	68.7	81.5	86.9	51.9	69.4	81.0	87.0
Worst	25.6	44.0	57.2	58.8	36.1	40.1	56.0	60.0
Best	60.8	85.7	93.9	97.1	83.4	95.4	97.6	99.5

Table 11: Execution time statistics for shell command data (FC-HMM)

User	CPU time (sec.) of FC-HMM (sequence length = 10)			CPU time (sec.) of FC-HMM (sequence length = 30)		
	No. of states	Training	Testing	No. of states	Training	Testing
0	10	4282	4	10	8014	6
1	50	22547	54	5	3969	1
2	30	17510	19	20	13269	20
3	10	4367	3	10	7862	8
4	20	15574	9	20	32696	20
5	20	19304	8	20	32466	20
6	20	17806	8	20	32616	20
7	20	14909	9	20	28561	20
Average		14537	14.3		19932	14.4

Table 12: Execution time statistics for shell command data (LR-HMM)

User	CPU time (sec.) of LR-HMM (sequence length = 30)			CPU time (sec.) of LR-HMM (sequence length = 50)		
	No. of states	Training	Testing	No. of states	Training	Testing
0	5	3522	4	5	7350	4
1	10	12194	4	30	30369	21
2	5	4122	5	10	14694	7
3	10	16095	4	10	12662	7
4	20	33532	12	20	21845	13
5	10	19264	6	5	10857	4
6	10	7783	7	10	11586	8
7	5	3634	4	10	15370	7
Average		12518	5.8		15591	8.9

Table 13: Execution time statistics for shell command data (cross entropy)

User	CPU time (sec.) of cross entropy (min. sequence length = 30)		CPU time (sec.) of cross entropy (min. sequence length = 50)	
	Training	Testing	Training	Testing
0	0	11	0	11
1	0	12	0	10
2	0	13	0	11
3	0	13	0	12
4	0	13	0	12
5	0	14	0	13
6	0	12	0	11
7	0	15	0	14
Average	0	12.9	0	11.8

Table 14: Data partitioning for shell command datasets in comparative study

User	Training set		Parameter selection set		Test set	
	No. of tokens	No. of traces	No. of tokens	No. of traces	No. of tokens	No. of traces
0	1557	49	1487	37	11992	356
1	1502	64	1714	63	11833	442
2	1995	76	1137	39	11877	330
3	1551	42	1474	40	12696	314
4	1500	45	1739	7	12255	311
5	1555	35	1507	55	11980	558
6	1500	90	1508	111	11277	1138
7	1590	52	1423	52	12250	456

Table 15: Classification results for shell command data in comparative study (instance-based learning [18])

Tested user	User model							
	0	1	2	3	4	5	6	7
0	99.3	57.0	31.7	61.0	75.1	0.6	38.5	10.1
1	14.9	92.9	12.4	64.2	16.3	0.9	4.0	6.0
2	41.3	58.7	94.7	43.6	71.1	0.3	47.9	8.3
3	64.8	91.7	46.7	90.0	86.4	0.6	69.0	15.1
4	34.4	21.2	18.6	72.1	92.7	1.3	8.6	3.0
5	50.4	68.3	39.7	70.3	78.0	99.9	57.2	29.4
6	41.8	15.4	17.7	82.3	48.7	0.6	91.7	4.7
7	24.7	11.0	8.7	40.7	22.1	0.6	5.8	96.2
FDR	0.7	7.1	5.3	10.0	7.3	0.1	8.3	3.8
Average TDR	38.9	46.2	25.1	62.0	56.8	0.7	33.0	10.9

Table 16: Classification results for shell command data in comparative study (cross entropy)

Tested user	User model							
	0	1	2	3	4	5	6	7
0	99.0	71.1	26.6	27.8	20.9	2.0	11.4	44.8
1	25.2	92.8	50.6	56.5	43.1	12.0	27.0	75.8
2	8.7	54.3	94.7	48.5	17.6	2.3	12.6	43.6
3	21.3	87.2	56.8	90.0	29.5	12.0	16.7	19.9
4	24.2	75.4	66.6	30.2	92.5	17.9	16.6	16.1
5	9.7	68.0	15.9	25.1	15.2	99.0	8.6	56.9
6	22.7	77.4	44.6	54.4	27.5	8.2	91.7	76.2
7	32.4	99.8	73.2	20.3	16.1	12.0	53.6	96.1
FDR	1.0	7.2	5.3	10.0	7.5	1.0	8.3	3.9
Average TDR	20.6	76.2	47.8	37.5	24.3	9.5	20.9	47.6