User Profiling for Intrusion Detection using Dynamic and Static Behavioral Models *

Dit-Yan Yeung and Yuxin Ding

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

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 user profiles built from normal usage data. In particular, user profiles based on Unix shell commands are modeled using two different types of behavioral models. 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. To determine whether 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 static modeling outperforms dynamic modeling for this application. Moreover, the static modeling approach based on cross entropy is similar in performance to instance-based learning reported previously by others for the same dataset but with much higher computational and storage requirements than our method.

1 Introduction

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 [6]. 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 caused by successful exploitation of system vulnerabilities.

Host-based intrusion detection systems detect possible attacks into individual host computers. Such systems typically utilize 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

* This research was supported 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.

to the exact operating systems used by individual computers as long as these computers can communicate among themselves using the same network protocol.

Two general approaches are commonly used for building intrusion detection systems. *Misuse detection* systems detect evidence of attacks based on prior knowledge about known attacks. *Anomaly detection* systems, on the other hand, model normal system behavior to provide a reference against which deviations are detected. In other words, the major difference between the two approaches is on whether normal or abnormal (i.e., intrusive) behavior is modeled explicitly. In this paper, data mining methods based on the anomaly detection approach are proposed for host-based intrusion detection. We consider normal user profiling based on shell command sequences from audit logs [20, 9, 14, 16, 18, 21].

Typical *classification* problems studied in pattern recognition can be formulated as classifying each pattern into one of c (≥ 2) classes with as low classification error as possible. To build such a discriminative classifier, training examples from all c classes are needed. While this formulation is commonly used in pattern recognition, there also exists another formulation called *novelty detection* [4, 2, 11]. In a probabilistic sense, it corresponds 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. The novelty detection approach is particularly attractive under situations where novel or abnormal patterns are expensive or difficult to obtain for model construction. In this paper, the novelty detection approach is adopted.

2 Dynamic versus Static Behavioral Models

Normal 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 essential for discriminating abnormal 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 feature representation.

Different anomaly detection methods have been used, e.g., instance-based learning [15, 16], multi-layer perceptrons [7, 20], decision trees [9], hidden Markov models (HMM) [14, 23], frequent episodes [18], correlation analysis [8], statistical likelihood analysis [7], rule learning [10, 17, 23], and uniqueness method [21]. 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 need both normal and intrusion data for model construction.

In this paper, intrusion detection systems based on profiling shell command sequences are first studied with the dynamic modeling approach using HMMs. Afterwards, we will propose an information-theoretic static modeling approach based on the usage frequencies of shell commands. Comparative studies will then be performed.

3 Dynamic Modeling Approach

3.1 Hidden Markov Models

HMMs are stochastic models of sequential data. Each HMM contains a finite number of unobservable states. State transitions are governed by a stochastic process to form a Markov chain. 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 are discrete symbols, as opposed to other models not studied here, e.g., continuous-density HMMs.

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.

To estimate the parameters of an HMM for modeling normal behavior, sequences of normal events (shell commands in our case) collected from normal system usage are used as training examples. An *expectation-maximization* (EM) algorithm [5] known as the *Baum-Welch re-estimation algorithm* [1] for mixture density estimation is used to find the maximum-likelihood (ML) parameter estimate. More details of the algorithm can be found in [19].

3.2 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 6.3.

4 Static Modeling Approach

4.1 Occurrence Frequency Distributions

Suppose the occurrence frequencies of different events (shell commands) are measured over a period of time. A probability distribution can represent the overall occurrence pattern. Since the event order is not considered, we refer to this as a static modeling method. Using this scheme, a normal user behavioral model is simply represented as an occurrence frequency distribution, with which possible system intrusions can be detected.

Let $P(M)$ denote the probability distribution characterizing a normal model M and let $P_i(M)$ denote the occurrence probability of event i among 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. For simplicity, the dependencies on M and S will not be shown.

4.2 Cross Entropy between Distributions

We need a dissimilarity measure between distributions. An information-theoretic measure that can serve this purpose is *cross entropy* [22, 12], which is also related to *Kullback-Leibler information measure* [13].

We use this definition of cross entropy: $C(P, Q) = \sum_{i=1}^{N} (Q_i - P_i) \log (Q_i / P_i)$. Note that these properties hold: (a) $C(P, Q) = C(Q, P)$; (b) $C(P, Q) \geq 0$; (c) $C(P, Q) = 0 \Leftrightarrow P = Q$. Thus, by checking whether the cross entropy between P and Q is larger than a certain threshold, one can decide if S should be considered a possible intrusion with respect to the model M. We will describe how to determine the threshold later in Section 6.3.

5 Data Preprocessing and Partitioning

5.1 Preprocessing of Shell Command Data

The shell command datasets are available at the public-domain KDD archive.[1] Since it is difficult to obtain real intrusion data, only normal data were collected via the history file mechanism from eight different Unix users. For each user login session, 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 letter.tex`), the set of all distinct tokens would become too large. To reduce the token set size, only a count of the files or directories is represented as a token (e.g., `ls -laF <3>`). All tokens in a login session form a *trace*.

Note that the datasets contain no real intrusion data because it is difficult to collect such data for this kind of intrusion detection applications. 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.

[1] `http://kdd.ics.uci.edu/databases/UNIX_user_data/UNIX_user_data.html`

5.2 Partitioning of Datasets

Each set of data is partitioned into three subsets: *training set* (normal data only), *threshold determination set* (normal data only), and *test set* (both normal and intrusion data). The training set of data is for estimating model parameters. Only normal data are needed for the novelty detection approach. As the model is built using normal data only, we need a criterion to decide when an observed behavior should be considered normal or intrusive. In particular, it corresponds to finding a threshold for some similarity measure (e.g., likelihood) or dissimilarity measure (e.g., cross entropy). The threshold determination set of normal data is used for determining this threshold. 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 model performance. More details about the performance measures used will be discussed in Section 6.1.

Table 1 summarizes the dataset sizes in our experiments. For each user, the (normal) data of all other users were treated as if they were "intrusive" data for that user. Since the available datasets are quite large, we used disjoint sets of data for training, threshold determination, and testing. Table 1 also shows the number of distinct tokens found in the data for each user. When the datasets for all eight users are combined together, we have a total of 2356 distinct tokens.

Table 1. Shell command datasets

	Training set		Threshold determination set		Test set		No. of distinct tokens in training set	Total no. of distinct tokens
User	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	151	286
1	196	5702	194	5327	365	5571	152	308
2	134	6776	134	3844	216	5120	174	484
3	313	13626	312	10309	286	11970	291	476
4	213	10826	212	11850	121	10981	375	561
5	832	20285	831	18009	762	19020	360	607
6	419	4485	418	5062	502	4738	228	447
7	562	16784	562	16586	466	16706	406	704

6 Model Construction and Performance Evaluation

6.1 Performance Criteria

We use two performance measures, namely, *true detection rate* (TDR) and *false detection rate* (FDR):

$$\text{TDR} = \Pr(\text{intrusive} \,|\, \text{intrusive}) \;=\; \frac{\#\text{ intrusive testing traces detected as intrusive}}{\#\text{ intrusive traces in test set}}$$

$$\text{FDR} = \Pr(\text{intrusive} \,|\, \text{normal}) \;=\; \frac{\#\text{ normal testing traces detected as intrusive}}{\#\text{ normal traces in test set}}$$

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 [23] 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* may also be used in place of TDR and FDR.

6.2 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. In our experiments, both fully-connected and left-to-right HMMs (denoted as FC-HMM and LR-HMM, respectively) were used. For the static modeling approach, all traces from the training set are used to create a distribution-based behavioral model.

6.3 Threshold Determination

After model training, the threshold determination set is used to determine an appropriate threshold for use later 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 from the training data.

For each chosen FDR for the threshold determination set, a corresponding threshold value can be obtained. In our experiments, different threshold values were tried by choosing different FDR values.

6.4 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.

For 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 be very long (if a user login session is long), we do not want to make a decision only after the trace ends. Instead, a distribution is computed

for each sub-trace sequence. The cross entropy between this distribution and the reference distribution of the model 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 1 below. The detection of possible intrusions in a trace can be performed immediately after the first K events have arrived. We refer to K as the *minimum sequence length.*

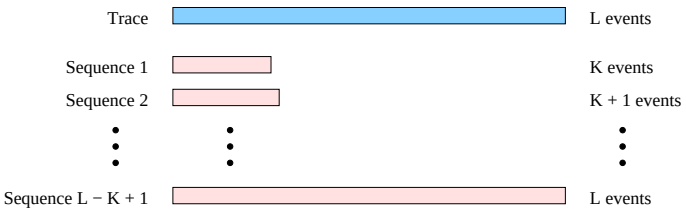

Fig. 1. Extraction of variable-length sequences from a trace

6.5 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 static modeling method.

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 and hence is a normal sequence. This problem can be formulated as applying a statistical test [3]. 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 $L(R) = \log \Pr(R|M)$. Similarly, the log-likelihood of S is denoted as $L(S) = \log \Pr(S|M)$. Based on the empirical probability distribution of $L(R)$ over $\mathcal{R}$, 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 reasonable 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.

7 Experiments

7.1 Results

Table 2 shows the results. For each method, only two choices of the sequence length or minimum sequence length are included to show the effect of varying the parameter, although more choices were actually tried. The threshold was chosen such that the FDR of the threshold determination set was equal to 5%, 10%, 15% or 20%. For each FDR value, the TDR shown is the average over the individual TDR values with the data from other users treated as intrusion data. The number of states shown is the minimum value that maximizes the TDR.

Table 2. Results for shell command data

	TDR (%) of FC-HMM (sequence length = 10)					TDR (%) of FC-HMM (sequence length = 30)				
User	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

	TDR (%) of LR-HMM (sequence length = 30)					TDR (%) of LR-HMM (sequence length = 50)				
User	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

	TDR (%) of cross entropy (min. sequence length = 30)				TDR (%) of cross entropy (min. sequence length = 50)			
User	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

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, increasing the sequence length would eliminate 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.

7.2 Discussions

In our experiments, the 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 weak and hence are not very useful for intrusion detection. The static shell command distribution seems to be sufficient for many users.

Although FC-HMM is usually slightly better 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 is similar in performance to the LR-HMM with sequence length 50. Note that the time complexity of each training iteration of an FC-HMM is $O(W^2T)$ for W states and sequence length T. 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 a Sun UltraSPARC 30 workstation with 256MB memory. Table 3 shows the CPU time required for the training and testing stages in the experiments reported in Table 2. Due to page limit, only the average statistics over all users are reported. It can be seen that LR-HMM is faster than FC-HMM for both the training and testing stages. Our static modeling method 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 changes in the system behavior.

7.3 Comparison with Previous Work

We also compared our results with those from previous work. To facilitate comparison, we performed another experiment using the same setup as in [15, 16]. The datasets were partitioned into training, parameter selection, and test sets as shown in Table 4. 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 earlier experiments. However, to facilitate comparison here, we used the same scheme as theirs for this experiment.

Table 5 shows the classification results obtained by [15] using *instance-based learning* (IBL), giving an average TDR of 34.2% with an average FDR of 5.3%, as well as our results using the static modeling method. The average TDR is 35.6%

Table 3. Execution time statistics for shell command data

CPU time (sec.) of FC-HMM (sequence length = 10)		CPU time (sec.) of FC-HMM (sequence length = 30)	
Training	Testing	Training	Testing
14537	14.3	19932	14.4

CPU time (sec.) of LR-HMM (sequence length = 30)		CPU time (sec.) of LR-HMM (sequence length = 50)	
Training	Testing	Training	Testing
12518	5.8	15591	8.9

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	12.9	0	11.8

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.

However, there are major differences between the two methods in terms of computational and storage requirements. 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 training set size. Similarly, during testing, the computational requirement is very low.

Table 4. Data partitioning for shell command datasets in comparative study

	Training set		Parameter selection set		Test set	
User	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

8 Concluding Remarks and Future Research

In this paper, we have presented two different anomaly detection approaches for a host-based intrusion detection problem based on shell command sequences. It was found that static behavioral models can give better results. It can be speculated that temporal dependencies are not very useful or may even be harmful for

Table 5. Results for shell command data in comparative study

Tested user	User model (instance-based learning)								User model (cross entropy)							
	0	1	2	3	4	5	6	7	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	99.0	71.1	26.6	27.8	20.9	2.0	11.4	44.8
1	14.9	92.9	12.4	64.2	16.3	0.9	4.0	6.0	25.2	92.8	50.6	56.5	43.1	12.0	27.0	75.8
2	41.3	58.7	94.7	43.6	71.1	0.3	47.9	8.3	8.7	54.3	94.7	48.5	17.6	2.3	12.6	43.6
3	64.8	91.7	46.7	90.0	86.4	0.6	69.0	15.1	21.3	87.2	56.8	90.0	29.5	12.0	16.7	19.9
4	34.4	21.2	18.6	72.1	92.7	1.3	8.6	3.0	24.2	75.4	66.6	30.2	92.5	17.9	16.6	16.1
5	50.4	68.3	39.7	70.3	78.0	99.9	57.2	29.4	9.7	68.0	15.9	25.1	15.2	99.0	8.6	56.9
6	41.8	15.4	17.7	82.3	48.7	0.6	91.7	4.7	22.7	77.4	44.6	54.4	27.5	8.2	91.7	76.2
7	24.7	11.0	8.7	40.7	22.1	0.6	5.8	96.2	32.4	99.8	73.2	20.3	16.1	12.0	53.6	96.1
FDR	0.7	7.1	5.3	10.0	7.3	0.1	8.3	3.8	1.0	7.2	5.3	10.0	7.5	1.0	8.3	3.9
Average TDR	38.9	46.2	25.1	62.0	56.8	0.7	33.0	10.9	20.6	76.2	47.8	37.5	24.3	9.5	20.9	47.6

this problem. Our information-theoretic static modeling method based on cross entropy is simple and computationally cheap, yet it can outperform the more sophisticated dynamic modeling method based on HMMs. A lesson to learn is that one should be careful in finding the best match between problems and methods.

A closer look at Table 5 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. Besides host-based intrusion detection problems, we are also conducting research on network-based intrusion detection. Some of the ideas learned from this research may also be relevant to network-based intrusion detection.

References

1. 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.
2. C.M. Bishop. Novelty detection and neural network validation. *IEE Proceedings: Vision, Image and Signal Processing*, 141(4):217–222, 1994.
3. P.R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1995.
4. W.J. Daunicht. Autoassociation and novelty detection by neuromechanics. *Science*, 253(5025):1289–1291, 1991.
5. 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.
6. D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.
7. 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.

8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
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. 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.
21. M. Schonlau and M. Theus. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1/2):33–38, 2000.
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. 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.