Anomaly Detection over Noisy Data using Learned Probability Distributions

Eleazar Eskin EESKIN@CS.COLUMBIA.EDU
Computer Science Department, Columbia University, 450 CS Building, 500 W. 120th st., New York, NY 10027

Abstract

Traditional anomaly detection techniques focus on detecting anomalies in new data after training on normal (or clean) data. In this paper we present a technique for detecting anomalies without training on normal data. We present a method for detecting anomalies within a data set that contains a large number of normal elements and relatively few anomalies. We present a mixture model for explaining the presence of anomalies in the data. Motivated by the model, the approach uses machine learning techniques to estimate a probability distribution over the data and applies a statistical test to detect the anomalies. The anomaly detection technique is applied to intrusion detection by examining intrusions manifested as anomalies in UNIX system call traces.

1. Introduction

Anomaly detection is an important problem in intrusion detection (Denning, 1987). Intrusion detection is the problem of detecting attacks on systems by examining various audit data of a system such as TCP packets or system logs and differentiating between normal users and intruders. Typical approaches to anomaly detection methods require training over clean data (normal data containing no anomalies) in order to build a model that detects anomalies. There are several inherent drawbacks to this approach. The first is that clean data is not always easy to obtain. Second, training over imperfect (noisy) data has serious consequences. If there is an intrusion hidden in the training data, the anomaly detection method will assume that it is normal and not detect subsequent occurrences. We show empirically that this is the case by examining two traditional methods for anomaly detection (Warrender et al., 1999). Third, it is difficult to make these systems adaptive in the sense that they can train "online" because they need to train on data which must be guaranteed clean.

This paper describes a technique for detecting anomalies without clean data. The method identifies anomalies buried *within* the data set. This method makes the assumption that the number of normal elements in the data set is significantly larger than the number of anomalous elements. This technique can be applied to a broader class of problems than intrusion detection. This technique was applied to detecting tagging errors in the Penn Treebank corpus (Eskin, 2000). The technique can also be applied to detecting anomalies in activity monitoring problems (Fawcett & Provost, 1999).

We are interested in detecting anomalies inside a data set for two main reasons. First, once we identify the anomalies, we can apply traditional anomaly detection methods using the clean data remaining. By removing the anomalous elements which may contaminate the model, we can create better models of the normal data. Second, the anomalies themselves can be of interest as they may show rarely occurring events.

We present a framework for detection of anomalies. The problem of anomaly detection is inherently difficult because even the nature of an anomaly as well as why they occur in data is disputed. We present a formal mixture model explaining the presence of the anomalies. We then train a machine learning method over the data set to obtain a probability distribution over the data. Since the number of anomalies is very small, we first assume that every element is normal. Motivated by our model of anomalies, we use the probability distribution to test each element to determine whether or not it is an anomaly.

For typical problems involving a mixture model, the problem can be cast as in incomplete data problem and estimated using the EM algorithm (Dempster et al., 1977). However, since anomalies are extremely rare with respect to the normal data, we can use the much simpler direct approach presented in this paper. Empirically, we show that this approach suffices for intrusion detection data obtained from live environments. In addition, because the intrusion detection application requires real time detection of anomalies, the approach presented in this paper is favorable because it can be computed very efficiently.

We evaluate this method on intrusion detection data. We examine traces of system calls in processes in which some instances of the processes are intrusions. We attempt to detect which of the processes are intrusions and which ones are normal processes. We compare the method presented in this paper with two traditional methods of intrusion detection, *stide* and *t-stide*, which are shown to perform well on clean data (Warrender et al., 1999). We show that our method performs significantly better than the traditional methods over noisy data. In addition, we show that our method performs comparably over noisy data to the performance of the traditional methods over clean data when the number of anomalies in the data is small compared to the number of normal elements.

2. Related Work

Anomaly detection is extensively used within the field of computer security specifically in intrusion detection (Denning, 1987).

Many different approaches to modeling normal and anomalous data have been applied to intrusion detection. A survey and comparison of anomaly detection techniques is given in (Warrender et al., 1999). Stephanie Forrest presents an approach for modeling normal sequences using look ahead pairs (1996) and contiguous sequences (Hofmeyr et al., 1998). Helman and Bhangoo (1997) present a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data. Lee et al. (1997; 1998) uses a prediction model trained by a decision tree applied over the normal data. Ghosh and Schwartzbard (1999) use neural networks to model normal data. Lane and Brodley (1997; 1998; 1999) examined unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity under normal use.

The work most similar to ours in intrusion detection is a technique developed at SRI in the Emerald system (Javitz & Valdes, 1993). Emerald uses historical records as its normal training data. Emerald compares distributions of new instances to historical distributions and differences between the distributions signal an intrusion. However, the problem with this approach is that intrusions present in the historical distributions may cause the system to not detect similar intrusions in the new instances.

The mixture model presented in this paper for explaining the presence of anomalies is typically computed using the EM algorithm (Dempster et al., 1977). A survey of the literature on the EM algorithm is given in McLachlan and Krishnan (1997). We use a bootstrap method to detect the anomalies. A survey of the bootstrap method is given in Efron and Tibshirani (1993).

A related problem to anomaly detection is the study of outliers in the field of statistics. In statistics, techniques have been developed for detecting outliers in univariate data, multivariate data, and structured data using a given probability distribution. A survey of outliers in statistics is given in Barnett and Lewis (1994).

3. Anomaly Detection

3.1 Data Model for Explaining Anomalies

In order to motivate a method for detecting anomalies, we must first make assumptions about how the anomalies occur in the data. We use a "mixture model" for explaining the anomalies, one of several popular models in statistics for explaining outliers (Barnett & Lewis, 1994). In the mixture model, each element falls into one of two cases: with (small) probability λ, the element is an "anomalous" element and with probability $(1-\lambda)$ the element is a "majority" element or a normal element.

In intrusion detection we are assuming that with probability $(1-\lambda)$ a given set of system calls is a legitimate use of the system, while with probability λ the set of system calls corresponds to an intrusion.

In this framework, there are two probability distributions which generate the data, a *majority* distribution and an *anomalous* distribution. An element x_i is either generated from the majority distribution, $\mathbf{M}$, or with probability λ from the alternate distribution, $\mathbf{A}$. Our generative distribution for the data, $\mathbf{D}$, is then:

$$\mathbf{D} = (1-\lambda)\mathbf{M} + \lambda\mathbf{A} \tag{1}$$

The mixture framework for explaining the presence of anomalies in the data is independent of the properties or types of the distributions $\mathbf{M}$ and $\mathbf{A}$.

In intrusion detection $\mathbf{M}$ is a structured probability distribution which is estimated over the data using a machine learning technique. The distribution $\mathbf{A}$ is a model of the anomalous elements. Typically this is a uniform distribution because a priori we do not know which elements are anomalies or what they look like.

The set of elements D, generated by distribution, $\mathbf{D}$, is partitioned into two subsets, M and A, corresponding to which elements were generated by distribution $\mathbf{M}$ and which elements were generated by distribution $\mathbf{A}$. We use the notation M_t (A_t) to denote the set of normal (anomalous) elements after processing element x_t. Initially, we have not detected any anomalies so the set of majority elements is the entire data set ($M_0 = D$) and the set of anomaly elements is empty ($A_0 = \emptyset$).

3.2 Modeling Probability Distributions

We can use any machine learning technique to model the probability distributions. We assume that for the normal elements we have a function $\mathcal{L}_M$ which takes as a parameter the set of normal elements M_t, and outputs a probability distribution, P_{M_t}, over the data D. Likewise, we have a function $\mathcal{L}_A$ for the anomalous elements which takes as a parameter the set of anomalous elements A_t. In other words:

$$P_{M_t}(X) = \mathcal{L}_M(M_t)(X) \tag{2}$$

$$P_{A_t}(X) = \mathcal{L}_A(A_t)(X) \tag{3}$$

These functions $\mathcal{L}_M$ and $\mathcal{L}_A$ can be any probability modeling method (i.e. Naive Bayes, Maximum Entropy, etc.) Note that because the set of anomalies is initially empty, $P_{A_0}(X)$ is the prior probability distribution because there are no elements in the training set for the probability modeling method $\mathcal{L}_A$.

3.3 Detection of Anomalies

Detecting anomalies, in this framework, is equivalent to determining which elements were generated by the distribution $\mathbf{A}$ and which elements were generated by the distribution $\mathbf{M}$. Elements generated by $\mathbf{A}$ are anomalies, while elements generated by $\mathbf{M}$ are not.

For each element x_t we determine whether it is not an anomaly and should remain in M_{t+1} or is an anomaly and should be moved to A_{t+1}.

In order to make this determination, we examine the likelihood of the two cases.

The likelihood, L, of the distribution $\mathbf{D}$ at time t is:

$$L_t(\mathbf{D}) = \prod_{i=1}^{N} P_D(x_i) = \left((1-\lambda)^{|M_t|} \prod_{x_i \in M_t} P_{M_t}(x_i)\right)\left(\lambda^{|A_t|} \prod_{x_j \in A_t} P_{A_t}(x_j)\right) \tag{4}$$

For computational reasons, we compute the log likelihood (LL) at time t:

$$LL_t(\mathbf{D}) = |M_t|\log(1-\lambda) + \sum_{x_i \in M_t} \log(P_{M_t}(x_i)) + |A_t|\log\lambda + \sum_{x_j \in A_t} \log(P_{A_t}(x_j)) \tag{5}$$

In order to determine which elements are anomalies, we use a general principle for determining outliers in multivariate data (Barnett, 1979).

We measure how likely each element x_i is an outlier by comparing the difference change in the log likelihood of the distribution if the element is removed from the majority distribution M_{t-1} and included with the anomalous distribution A_{t-1}. In other words, we examine the change in LL_t if:

$$M_t = M_{t-1} \setminus \{x_t\} \tag{6}$$

$$A_t = A_{t-1} \cup \{x_t\} \tag{7}$$

If this difference ($LL_t - LL_{t-1}$) is greater than some value c, we declare the element an anomaly and permanently move the element from the majority set to the anomaly set. Otherwise, the element remains in the normal distribution.

$$M_t = M_{t-1} \tag{8}$$

$$A_t = A_{t-1} \tag{9}$$

We repeat this process for every element and in the end we get a partition of data set into a set of majority elements and a set of anomalous elements.

Note that we have to use the machine learning method to recompute the probability distributions P_{M_t} and P_{A_t} using $\mathcal{L}_{\mathcal{M}}$ and $\mathcal{L}_{\mathcal{A}}$ at every step because of the change in the sets M_t and A_t.

This test is an invocation of the Neyman-Pearson Lemma, choosing the most likely hypothesis. If $c = 0$ then this test performs strictly a maximum likelihood test choosing the most likely case of whether or not an element is an anomaly. If the difference is greater than c then according to the test it is more likely the element is an anomaly.

The two parameters in this test, c and λ, can be set to optimize performance of this method on a given problem. The choices are influenced by intuition about the actual anomaly rate and the sensitivity of the problem to misclassified anomalies. The parameter c affects the number of anomalies that are detected by the system. With very low values of c only the most extreme anomalies are detected while with higher values of c more elements are declared anomalies.

4. Anomaly Detection Applied to Intrusion Detection

We applied the anomaly detection framework to detect intrusions based on the analysis of process system calls. We examined two sets of system call data containing intrusions.

The data analyzed is a set of system call *traces* for a given program. A trace is the history (or list) of system calls made by a process of the given program from beginning of execution to the termination of the process. The arguments to the system calls are ignored for the analysis. In both of these sets, there was a set of clean traces and a set of intrusion traces.

Typical attacks that we examine were exploits targeted at certain programs on UNIX machines. The programs under attack all run as superuser which allow for a "user to root" attack. This attack allows an unprivileged user to obtain root user privileges. Each of these attacks exploits bugs in the programs which allow for the change in user privileges. The underlying premise is that the sequences of system calls during an intrusion are noticeably different from normal sequences of system calls.

The first set of data is from the BSM (Basic Security Module) data portion of the 1999 DARPA Intrusion Detection Evaluation data created by MIT Lincoln Labs (1999). The data consists of 5 weeks of BSM data of all processes run on a Solaris machine. We examined three weeks of traces of the programs which were attacked during that time. The programs attacked were: *eject*, *ps*, and *ftp*.

The second set of data was obtained from Stephanie Forrest's group at the University of New Mexico. This data set is described in detail in Warrender et al. (1999). This data contains up to 15 months of normal traces for certain programs as well as intrusion traces. The data provides normal and intrusion traces of system calls for several processes. We examine the data for the processes that were attacked with a "user to root" attack. The processes examined correspond to the programs: *named*, *xlock*, *login*, and *ps*.

Since our method assumes that the number of intrusions is small compared to the size of the normal data, we only compare our method to the traditional methods for anomaly detection over traces of programs where the intrusions compose less that 5% of the total number of system calls. Later we show the performance of our method over traces of programs which contain a higher percentage of intrusion traces. Tables 1 and 2 summarize the data sets and list the number of system calls and traces for each program.

4.1 Detecting Anomalies in Sequences of System Calls

We assumed that when anomalies occur, they are random. Thus we set $\mathcal{L}_A$ to be a function that always returns a uniform distribution over all sequences which represents the anomaly distribution P_{A_t} for all t. We used a fixed order Markov chain probability modeling method ($\mathcal{L}_{M_t}$) over the current set of normal elements to build the probability model P_{M_t}. Any probability model can be applied to this problem such as naive Bayes, models estimated using maximum entropy, hidden Markov models, variable order Markov chains, etc.[1]

Our probability model is computed by examining what the next symbol is following a sequence of a given length L by counting the number of times that symbol followed the sequence in the training data. Thus we compute:

$$P(X_t | X_{t-1}, X_{t-2}, .., X_{t-L}) \tag{10}$$

In order to avoid probabilities of 0, we use a pseudo count predictor and add an initial value to each count. For all of our experiments we used $L = 3$. In principal, L can be set optimally to best estimate the probability distribution.

In this application, each *element* is a single system call in a process trace. The probability distribution for the element is conditioned on the the previous L system calls in the process trace.

Initially we computed the model over all elements, $M_0 = D$. Then for each element we computed the difference in log likelihoods (see Equation 5) to determine whether or not the element is an anomaly.

We repeat this process for every element and in the end we have partition of the data set into a set of majority elements and a set of anomalous elements.

Note that in principal we have to retrain our learning algorithm at every step to compute the log likelihood. This can be done efficiently because only the elements with the same preceding sequence need to be recomputed. In addition, the difference in log likelihoods between the two distributions was computed directly. Essentially, detection of intrusions took only 2 passes through the data. The first pass was to train the probability distributions assuming that every element was normal. The second pass computed the change in log likelihood of the distribution if a given element was declared an anomaly. Since the change in log likelihood

[1] In fact we obtained slightly better results using Sparse Markov transducers but omitted them from the paper due to space considerations (Eskin et al., 2000).

Table 1. Lincoln Labs Data Summary

Program Name	# Intrusion Traces	# Intrusion System Calls	# Normal Traces	# Normal System Calls	% Intrusion Traces
ftpd	1	350	943	66842	0.05%
ps (LL)	21	996	208	35092	2.7%
eject	6	726	7	1278	36.3%

Table 2. University of New Mexico Data Summary

Program Name	# Intrusion Traces	# Intrusion System Calls	# Normal Traces	# Normal System Calls	% Intrusion Traces
xlock	2	949	72	16,937,816	0.006%
named	2	1,800	27	9,230,572	0.01%
login	9	4,875	12	8,894	35.4%
ps (UNM)	26	4,505	24	6,144	42.3%

was computed efficiently, the actual computation took only seconds to perform on each data set.

4.2 Baseline Comparison Methods: stide and t-stide

We compare our method against two methods, *stide* and *t-stide*, shown to be effective in detecting intrusions in system call data when trained over clean data in experiments performed on the University of New Mexico data set (Warrender et al., 1999).

The sequence time-delay embedding (stide) algorithm keeps track of what sequences were seen in the training data and detects sequences not seen in training. The method builds a model of normal data by making a pass through the training data and storing each unique contiguous sequence of a predetermined length in an efficient manner. We used a length of six because that is the length of the sequences used in the published results of the method.

When the method is used to detect intrusions, the sequences from the test set are compared to the sequences in the model. If a sequence is not found in the normal model, it is called a *mismatch* or anomaly.

The threshold sequence time-delay embedding (t-stide) algorithm is an extension of the stide algorithm which incorporates a threshold. In addition to unknown sequences, rare sequences are also counted as mismatches. In this method, any sequence accounting for less than 0.001% of the total number of sequences is considered rare.

To detect intrusions, these methods compare the number of mismatches in a local region of 20 consecutive sequences. A threshold is set for these local regions between 1 and 20. If the number of mismatches reaches or exceeds the local mismatch threshold, the process is declared an intrusion.

4.3 Experimental Results

We compare the performance of the method presented in this paper with the baseline methods described above. We first empirically show that the method presented in this paper out performs the baseline methods when trained over noisy data. Then we empirically show that the performance of this method trained over noisy data performs comparably to the baseline methods trained over clean data.

If a process trace contains an anomaly, we declare that process an intrusion. We consider an intrusion detected if either the intrusion process is detected, or one of the processes spawned by the intrusion is detected.

We compare the anomaly detection methods in both sets of experiments using ROC curves which graph the false positive rate versus the detection rate (Provost et al., 1998). The detection rate is the percentage of intrusions which are detected. In order to be consistent with previous published results on these data sets, the false positive rate is defined to be the percentage of normal system calls which are declared anomalous (Warrender et al., 1999). The parameter settings of the methods are varied to obtain multiple points on the ROC curve. The ROC curves have few points because of the small amount of intrusion traces in each data set.

The performance over noisy data were evaluated as follows. We combined the intrusion data and the normal data into a single unlabeled data set and applied our method to detect anomalies. For stide and t-stide, we used the entire data set as both the training set and test set. The stide method, does not detect any

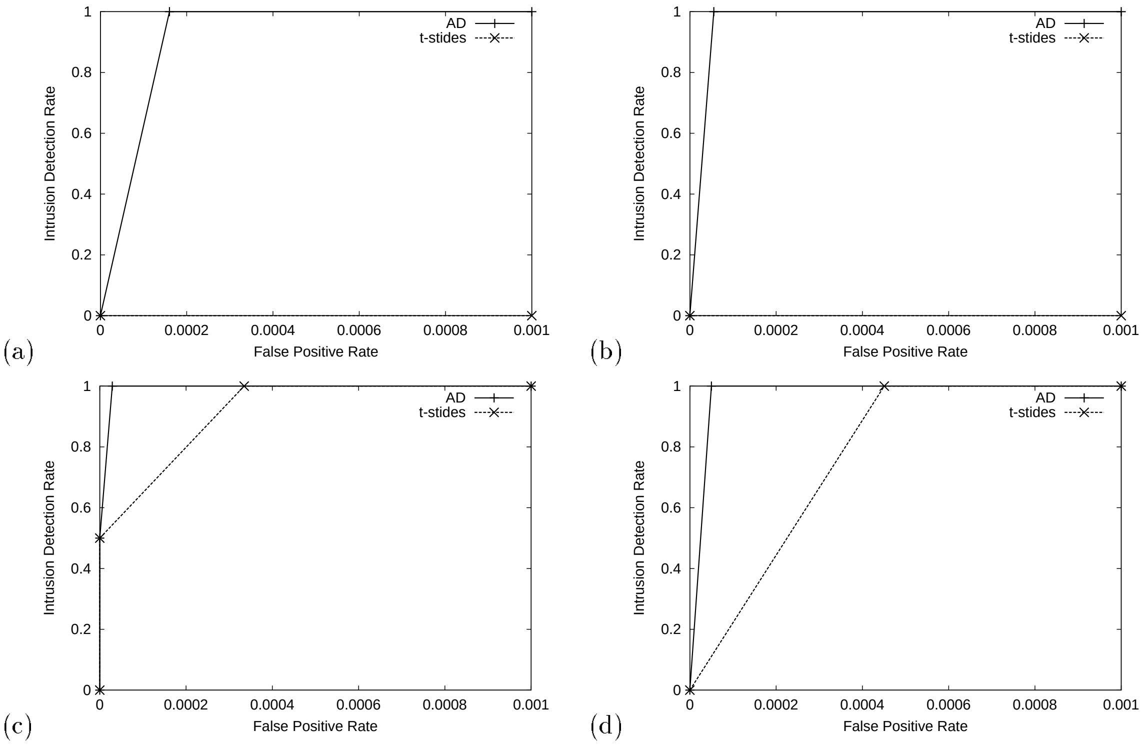

Figure 1. ROC curves showing a comparison with t-stide over noisy data. The method presented in the paper is labeled AD in each graph. The curves are shown for all programs where the amount of intrusion system calls compose less that 5% of the data: (a) ftpd, (b) ps, (c) xlock, and (d) named.

anomalies because every sequence is seen in the training. Thus, no "mismatches" occur. Since the "rare" threshold for the t-stide method is fixed a priori and determined by the total number of system calls in the data set, there is no guarantee that any "rare" sequences will appear in he data. In the case of the *ftp* and *ps (LL)* programs, no sequence was below the threshold thus no anomalies were detected. A comparison of the performance of the t-stide method and our method for each data set is shown in Figures 1(a)–(d).

We also compare the performance of our method over noisy data to traditional methods over clean data. The stide and t-stide methods are trained on 1/3 of the clean data and the test set is the remaining data. Figures 2(a)–(d) show the performance comparisons.

5. Analysis

The anomaly detection method presented in this paper makes three important assumptions. The first assumption is that the normal data can effectively be modeled using the probability distribution. The second is that the anomalous elements are sufficiently different from the normal elements in order to be detected. The third is that the number of anomalies is small compared to the number of number of normal elements.

In the case of intrusion data, both the first two assumptions hold. System calls for the normal processes are very regular and can be modeled effectively. The intrusion traces are significantly different from the normal traces because the intrusions exploit bugs in the program to obtain a root shell. Since this never happens in normal processes, the system call traces are significantly different.

The third assumption is required so that the anomalies can be observed against the background of normal data. If there are too many anomalies, the model of the normal distribution will be significantly distorted by the anomalies that the anomalies are difficult to detect. We verify this empirically by showing the results of the method when applied to data sets which have a high proportion of intrusion system calls. In these data sets, the method does not perform as well. We compare the ROC curves in Figure 3 to demonstrate that the the method performs better when there are relatively few intrusions. As expected on the data sets of the programs which have few intrusions (*ftpd*, *ps (LL)*, *xlock*, *named*) the method perform better than on the data sets which have a high proportion of intrusions (*login*, *ps (UNM)*). The system performs well on

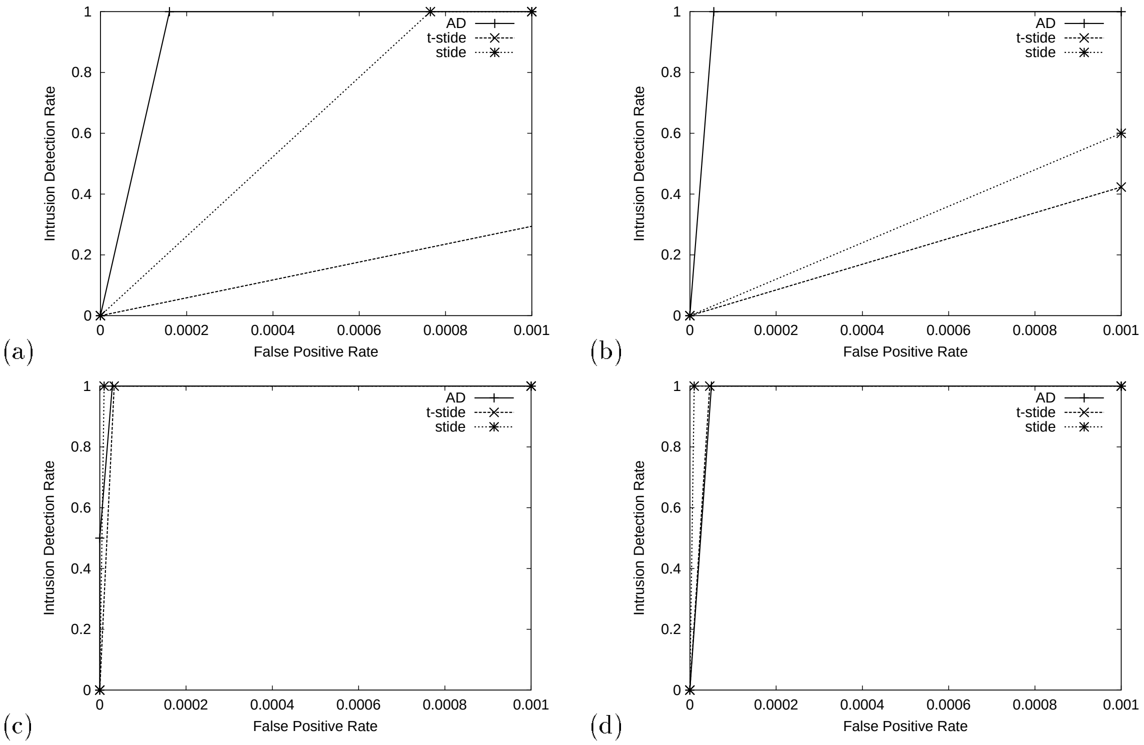

Figure 2. ROC curves showing the comparison of this method trained over noisy data and stide and t-stide trained over clean data. The method presented in the paper is labeled AD in each graph. The curves are shown for all programs where the amount of intrusion system calls compose less that 5% of the data: (a) ftpd, (b) ps, (c) xlock, and (d) named.

eject although there is a high proportion of intrusions in the data. This may be because the eject program is typically always used in the same way. This makes it easier for the system to learn the normal pattern even though there is little data and a significant amount of noise.

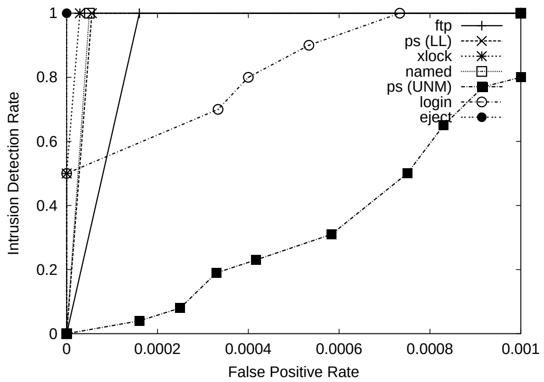

Figure 3. ROC curves illustrating the performance of the method over data sets with different percentage of intrusion traces.

6. Conclusion

We have presented a probabilistic approach for detecting anomalies without a set of normal data. The approach leverages the fact that the anomalies are rare within data as compared to the number of normal elements. This is an improvement over traditional anomaly detection methods which required clean training data. In many applications, clean data is difficult to obtain and difficult to ensure that it contains no anomalies.

We evaluated the anomaly detection approach by applying it to intrusion detection and compared it to traditional methods for anomaly detection. However, the framework presented for detecting anomalies can be applied to a broader class of problems. For example the technique can be used to detect errors in large data sets.

Any probability modeling method could be used for detection of anomalies. In addition over sequences of system calls, different portions of the sequence (beginning of a process versus the end of a process) can be modeled separately to obtain more accurate models. In this framework, more accurate probability estimations may provide better results in intrusion detection.

Within intrusion detection, future work includes building truly adaptive anomaly detection systems which detect anomalies using data obtained in live conditions. Future work also includes using the probabilistic framework to incorporate knowledge of known intrusions into an anomaly detection system.

References

Barnett, V. (1979). Some outlier tests for multivariate samples. *South African Statist*, *13*, 29–52.

Barnett, V., & Lewis, T. (1994). *Outliers in statistical data*. New York: John Wiley and Sons.

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

Denning, D. (1987). An intrusion detection model. *IEEE Transactions on Software Engineering*, *SE-13*, 222–232.

Efron, B., & Tibshirani, R. (1993). *An introduction to the bootstrap*. Boca Raton, FL: Chapman & Hall.

Eskin, E. (2000). Detecting errors within a corpus using anomaly detection. *Proceedings of First Conference of the North American Association for Computational Linguistics*.

Eskin, E., Grundy, W. N., & Singer, Y. (2000). Protein family classification using sparse Markov transducers. *Proceedings of Eighth International Conference on Intelligent Systems in Computational Biology*.

Fawcett, T., & Provost, F. (1999). Activity monitoring: Noticing interesting changes in behavior. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*.

Forrest, S., Hofmeyr, S. A., Somayaji, A., & Longstaff, T. A. (1996). A sense of self for unix processes. *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (pp. 120–128). IEEE Computer Society.

Ghosh, A., & Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. *Proceedings of the Eighth USENIX Security Symposium*.

Helman, P., & Bhangoo, J. (1997). A statistically base system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, *27*, 449–466.

Hofmeyr, S. A., Forrest, S., & Somayaji, A. (1998). Intrusion detect using sequences of system calls. *Journal of Computer Security*, *6*, 151–180.

Javitz, H. S., & Valdes, A. (1993). *The nides statistical component: Description and justification* (Technical Report). SRI International.

Lane, T., & Brodley, C. E. (1997). Sequence matching and learning in anomaly detection for computer security. *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management* (pp. 43–49). Menlo Park, CA: AAAI Press.

Lane, T., & Brodley, C. E. (1998). Temporal sequence learning and data reduction for anomaly detection. *In Proceedings of the Fifth ACM Conference on Computer and Communications Security* (pp. 150–158).

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

Lee, W., & Stolfo, S. J. (1998). Data mining approaches for intrusion detection. *In Proceedings of the Seventh USENIX Security Symposium*.

Lee, W., Stolfo, S. J., & Chan, P. K. (1997). Learning patterns from unix processes execution traces for intrusion detection. *In Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management* (pp. 50–56). Menlo Park, CA: AAAI Press.

McLachlan, G. J., & Krishnan, T. (1997). *The EM algorithm and extensions*. New York: John Wiley and Sons.

MIT Lincoln Labs (1999). 1999 DARPA intrusion detection evaluation. *http://www.ll.mit.edu/IST/ideval/index.html*.

Provost, F., Fawcett, T., & Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*.

Warrender, C., Forrest, S., & Pearlmutter, B. (1999). Detecting intrusions using system calls: alternative data models. *Proceedings of the 1999 IEEE Symposium on Security and Privacy* (pp. 133–145). IEEE Computer Society.