

Applications of Hidden Markov Models to Detecting Multi-stage Network Attacks

Dr. Dirk Ourston, Ms. Sara Matzner, Mr. William Stump, and Dr. Bryan Hopkins,
Applied Research Laboratories University of Texas at Austin
P.O. Box 8029, Austin, TX 78713-8029
1-512-835-3200
{ourston, matzner, stump, bhopskins}@arlut.utexas.edu

Abstract. *This paper describes a novel approach using Hidden Markov Models (HMM) to detect complex Internet attacks. These attacks consist of several steps that may occur over an extended period of time. Within each step, specific actions may be interchangeable. A perpetrator may deliberately use a choice of actions within a step to mask the intrusion. In other cases, alternate action sequences may be random (due to noise) or because of lack of experience on the part of the perpetrator. For an intrusion detection system to be effective against complex Internet attacks, it must be capable of dealing with the ambiguities described above. We describe research results concerning the use of HMMs as a defense against complex Internet attacks. We*

describe why HMMs are particularly useful when there is an order to the actions constituting the attack (that is, for the case where one action must precede or follow another action in order to be effective). Because of this property, we show that HMMs are well suited to address the multi-step attack problem. In a direct comparison with two other classic machine learning techniques, decision trees and neural nets, we show that HMMs perform generally better than decision trees and substantially better than neural networks in detecting these complex intrusions.

Keywords: *Coordinated Internet attacks, Hidden Markov Models, rare data, noise, multi-stage network intrusions, partial data.*

Introduction

Among the computer network attacks most destructive and difficult to detect are those that occur in stages over time. In accomplishing a compromise of this type, an intruder must go through a process of reconnaissance, penetration, attack, and exploitation. Whereas current intrusion detection methods may be able to identify the individual stages of an attack with more or less accuracy and completeness, the recognition of a sophisticated multi-stage attack involving a complex set of steps under the control of a master hacker remains difficult. The benefits of applying machine learning techniques to this domain are that they eliminate the manual knowledge acquisition phase required by rule-based approaches and provide a generalization of the models for the attack types. However, several challenges also face the use of machine learning methods in the construction of intrusion detection systems in response to complex intrusions. The correlation of stages separated by a significant amount of time is difficult to model and

detect, and although a successful multi-stage attack normally requires that each phase of the attack be completed before the next stage, the specific actions taken within a given stage may be interchangeable. Moreover, certain stages may not be present or discernable and data corresponding to a stage may be fragmentary or lost, particularly in the case where wireless communications are being utilized. Truly sophisticated Internet attacks of this type may occur only rarely when compared to the more ubiquitous network intrusion activities such as probes and scans. The small number of examples of complex Internet attacks contributes to the difficulty associated with applying machine learning (ML) techniques to this problem, since most ML algorithms require a training set containing many examples. In addition, the small number of examples also makes it difficult to measure the performance of the detection algorithm. In response to the difficulties outlined above, we have chosen an approach using Hidden Markov Models (in the remainder of the paper, HMMs will be used as the

plural of HMM) to describe the sequential and probabilistic nature of multi-stage Internet attacks [1].

The remainder of this paper is organized as follows. Based on the problem domain being addressed and the techniques used, we compare our work to that of other research groups. We present in detail the salient properties of an HMM that motivated our approach. We present our system architecture, showing the HMM within the deployed system. Then we discuss the process that we employed to prepare the incoming data for use by the HMM algorithm. We present experimental results using a variety of measures, and we compare the performance of HMMs against two classic machine learning algorithms, decision trees and neural networks. We provide an introduction to techniques that can be applied to the rare data problem because, as we have already mentioned, complex Internet attacks may occur only rarely. Finally, we discuss our plans for future work, including investigation of the rare data problem, investigation of methods for recognizing the early stages of an attack, and methods for coping with noisy or missing data.

Related Work

Other researchers have also studied multi-stage Internet attacks. For example, work at Stanford Research Institute (SRI) [2] has included a probabilistic approach to intrusion detection. The SRI approach calculates the similarity between alerts of various types, emanating from multiple sensors. If the alerts are sufficiently similar, they are fused into a meta-alert that summarizes the information contained in the individual alerts. In the SRI approach, the probability of transitioning from one attack phase to another (attack states in our formulation) is specified by the incident class similarity matrix. This matrix is comparable to the state transition matrix in the HMM representation. Therefore, a significant difference between our approach and the SRI approach is that we compute the state transition matrix automatically, using ML techniques, whereas the SRI approach derives the incident class similarity matrix manually, using human judgment. However, an HMM could be used as a source for the correlated attack reports, discussed in Ref. [2], using the meta-alerts as input.

A group at Purdue [3] has included temporal sequence learning in their approach to intrusion detection. This work has focused on host-based intrusions, rather than network intrusions. The ML algorithm that they have selected is instance-based learning [4]. In this technique, a set of exemplars is chosen for the class to be modeled. These exemplars are then compared with example data using a similarity metric to determine if the example is a member of the given class. The HMM representation could be applied to this domain directly using the same format for the examples that was used at Purdue.

Researchers at IBM [5] developed a system that incorporates network alarms instead of raw network data. This system is limited in that it is sensitive to

extraordinary events and does not target misuse specifically, as does the HMM representation that we have implemented. The IBM group uses an association rule mining algorithm for identifying events that frequently occur together within the incoming alert data stream. These rules are then used to characterize “normal” operation of the network. Any deviation from the conditions characterized by these rules is considered to be an anomaly, or a potential attack. An approach using an HMM could be applied to this problem domain, with the exception that the HMM output would be various specific attack types, rather than the single broad category of “anomalies.”

Our approach, using Hidden Markov Models [6], has only recently been applied to the intrusion detection problem [7]. The work at the University of New Mexico [7] considered HMMs as a tool for detecting misuse based on operating system calls. This study found that HMMs provided greater classification accuracy than other methods, but at the expense of additional computations.

The HMM technique has been extensively applied to problems in speech recognition [8], text understanding [9], image identification [10], and microbiology [11]. Finally, several institutions have addressed the problem of complex Internet attacks [12], [13], applying non-probabilistic ML techniques [14], [15].

The HMM

The Hidden Markov Model (see Figure 1) starts with a finite set of states. Transitions among the states are governed by a set of probabilities (transition probabilities) associated with each state. In a particular state, an outcome or observation can be generated according to a separate probability distribution associated with the state. It is only the outcome, not the state, that is visible to an external observer. The states are “hidden” to the outside; hence the name Hidden Markov Model. The Markov Model used for the hidden layer is a first-order Markov Model, which means that the probability of being in a particular state depends only on the previous state. While in a particular state, the Markov Model is said to “emit” an observable corresponding to that state. One of the goals of using an HMM is to deduce from the set of emitted observables the most likely path in state space that was followed by the system.

Given the set of observables contained in an example corresponding to an attack, an HMM can also determine the likelihood of an attack of a specific type. In our case, the observables are the set of alerts contained in the example. Each example is constructed to contain all of the alerts that occurred between a specified source host and a specified target host, over a specified time period (in our case 24 hours). The goal for the HMM is therefore to determine the most likely attack type corresponding to a sequence of alerts.

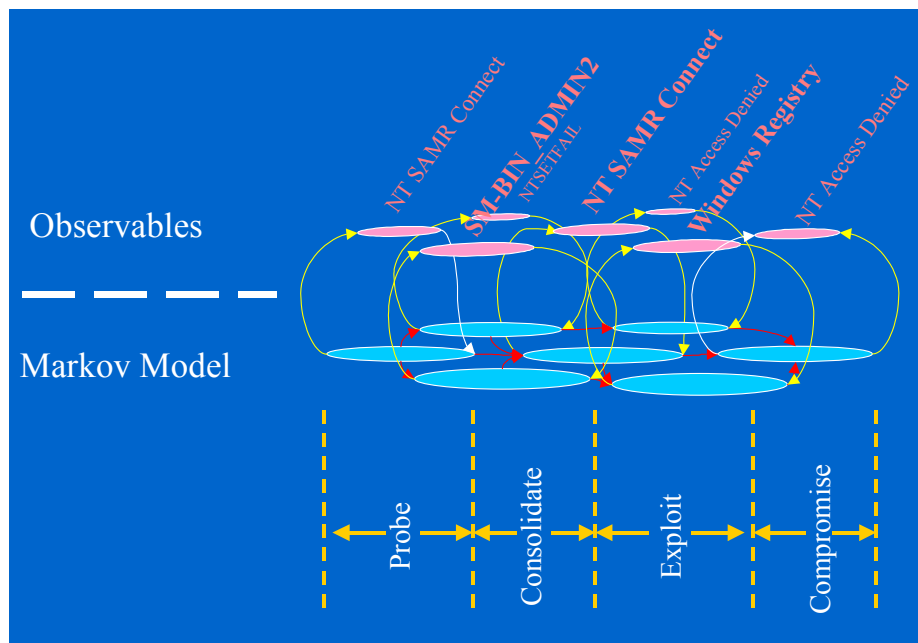


Figure 1. A simple HMM

To train the HMM, a set of training examples is created, with each example labeled by category (see *Developing Labeled Data*, below). The maximum likelihood method [16] is used to adjust the HMM parameters for optimal classification of the example set. The HMM parameters are the initial probability distribution for the HMM states, the state transition probability matrix, and the observable probability distribution.

The state transition probability matrix is a square matrix with size equal to the number of states. Each of the elements represents the probability of transitioning from a given state to another possible state. The matrix is not symmetric. For example, the likelihood of transitioning from the state corresponding to an *ip scan* into the state corresponding to a *port probe* is very high, since these two events are likely to occur in proximity to each other in the order given. However, the transition *port probe* to *ip scan* is much less likely, since port probes are usually conducted on hosts that have been identified by ip scans. The observable probability distribution is a non-square matrix, with dimensions number of states by number of observables. The observable probability distribution represents the probability that a given observable will be emitted by a given state. For example, in the late stages of an attack, the observable “root login” would be much more likely than “port probe,” since port probes typically happen at the beginning of an attack and a root login typically happens towards the end.

We implemented our system using a separate HMM for each classification category since the standard HMM is designed to simulate a single category and the intrusion detection domain includes

several attack categories. This approach facilitates parallelization in the operational system, if that is desired.

Why Use an HMM?

There are several properties of alert sequences corresponding to multi-stage Internet attacks that match well with the HMM representation. A multi-stage Internet attack often consists of several steps, each of which is dependent on the outcome of the previous step. These steps are identified in an input example by the alert corresponding to the step. In the HMM, each step is *probabilistically* dependent on the previous step. Also, in the HMM representation, each step is identified by the observable, that is, the alert, corresponding to the step. The steps themselves are modeled internally in the HMM as states.

An intruder mounting an Internet attack has many ways to accomplish his goals. Within the HMM, each alternative is represented as a specific state sequence. Furthermore, the HMM calculates a probability that a particular state sequence was followed during the attack. This probability is related to the frequency with which the set of transitions contained in the sequence was seen in the HMM training data.

There may also be different ways to accomplish each step in the state sequence. For example, suppose that the goal for a given step is to discover whether or not a host exists at a given ip address. One method to achieve this goal is to attempt to initiate a telnet connection with the host. A second is to consult the DNS server responsible for the network. A third method might involve attempting to establish an FTP connection. Out of the various possible actions, only

one is chosen, resulting in the observable for the step. The likelihood associated with each alert type is then calculated according to the frequency with which the particular alert type had been associated with the given state in the training data. The HMM therefore can simulate variations in both the attack step sequence and the choice of methods for accomplishing the goals of each step. The hidden and visible layers of the HMM also have semantic value and could support the development of a cognitive model of the way attackers accomplish their goals. For example, attackers sometimes attempt to disguise their work as normal network activities. The attacker only reveals his motives when he has gained his objectives—and

sometimes not then. The HMM observable layer models the overt portion of the attack, whereas the hidden level represents the attacker's true intentions, a useful property when the attacker attempts to hide his true intentions.

System Architecture

Figure 2 shows the architecture for the intrusion detection system. The current status of the system is that all of the modules shown in the architecture have been implemented as research prototypes, and the system is currently undergoing extensive test and evaluation (the results of some of our experimentation form the basis for this paper).

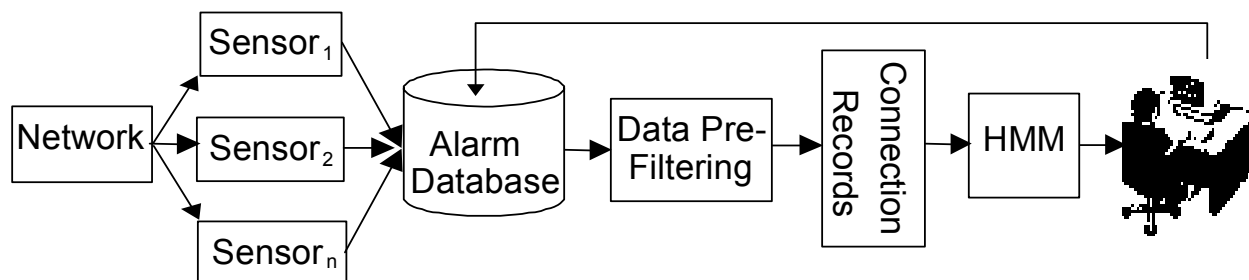


Figure 2. System architecture

The flow through the system is as follows: One or more sensors monitor network traffic and declare alerts when potential intrusions are detected. The alerts from the sensors are stored in an alarm database for later processing by the system. Data pre-filtering (see Preparing the Data, below) is used to eliminate redundant data from the input data. After the alarm data has been pre-filtered, it is assembled into examples for analysis by the HMM classification module. Currently, the example input to the HMM consists of a source address (the source host), a destination address, and an ordered sequence of alerts that occurred between the source ip and the destination ip during a 24-hour period. The HMM highlights high-interest examples for further analysis. In the event that an example is incorrectly classified by the HMM, the network analyst corrects the classification and sends the example back to the alert database so that the HMM can be updated using that example.

Preparing the Data

In the real-time data stream, alerts come in as independent events from each of many network sensors. A single record of the real-time data consists of the type of alert being reported by the network sensor, as well as various types of context information including bytes transferred, source host address, target host address, and so on. Viewing the data in this manner tends to obscure correlations among the alerts, and makes identification of coordinated attacks more difficult. Consequently, we elected to group the data when forming our examples.

Each example consists of the temporally ordered sequence of alerts that occurred between a given source/destination host pair over a 24-hour period.

Using the network sensor data as input allows us to work at a higher level of abstraction than is possible with raw packet data. The network sensors filter out a large number of normal connections, letting our algorithms focus on events that more probably correspond to intrusions. Although using sensors improves the quality of the data being processed by our algorithms, we still have to deal with the “false positive” problem. This problem originates from the fact that most, if not all, network sensors adhere to the philosophy that it is preferable to include many erroneously identified intrusions in the input data stream (false positives) rather than to miss one real intrusion (false negative).

In addition, our input data originally suffered from the alert repetition problem: a single alert type or a set of alert types being repeated over and over in the example. A particular example of this phenomenon occurs during probe activities. In this case the source host sends out service requests over all possible port numbers on the target machine, attempting to find vulnerabilities in the service applications. An alert is generated for each probe attempt, resulting in *...pppp...* appearing in the example, where p corresponds to a probe alert. It can also happen that a particular alert sequence gets repeated in the example as *...abcabcabc...*, where a, b, and c are each unique alert types. However, for our purposes in identifying coordinated attacks these repetitive signatures provide little or no information

content to the learning algorithm. As a result, we eliminated these repetitive signatures from the input examples used for training and testing, replacing them with a single alert or alert sequence of the given type.

Developing Labeled Data

One of the main problems associated with using machine-learning algorithms is that of finding realistic training data. In general, there is a paucity of labeled data available for most problem domains. By labeled data we refer to an example that has been annotated with its associated category. Labeling examples requires both a domain expert and a great deal of time.

We have implemented a semi-automatic approach that greatly accelerates the labeling process by using subject matter experts to develop category models. These models are based on the feature sets of the examples. The feature sets result in a simplistic assignment of examples to categories. Specifically, what we required of the subject matter experts is that they identify a mapping between alert types and intrusion categories. For example, an alert of type “port probe” might be mapped to intrusion category “initial recon.” This mapping was accomplished using the following simple rule to assign an alert to a category. “Assign the alert to the most interesting category in which the alert might appear at the end of an attack sequence.” Thus, a port probe would appear in category 1 (least interest) by this rule, since a port probe does not normally appear at the end of any attack sequence, and a port probe by itself is not a particularly interesting event. Note that this mapping only required assignment of alert *types* to categories; in our case there were about 200 alert types, as

compared to manually labeling thousands of examples. Next, we *automatically* assigned examples to categories using this alert mapping. This was done by assigning each example to the category associated with the highest category alert contained in the example. Using this approach, we were able to avoid labeling individual examples by hand instead automatically creating an initial set of labeled examples. For research purposes, we have used the examples labeled by this method in order to generate comparisons among candidate learning algorithms.

In the operational system, incorrectly identified examples would be re-labeled correctly by the analyst, and then the HMM would be updated using the corrected examples. It should be noted that while we are not able to discuss in detail the characteristics of the input data, due to security issues, the information used in our examples is real network data, generated by real network sensors connected to an operational network.

Empirical Results

Figure 3, below, compares the performance of the HMM algorithm against the 4.8 (latest public) version of the C4.5 decision tree algorithm and a standard neural network implementation (both the C4.5 algorithm and the neural network software were taken from the WEKA [17] machine learning library). By “performance” we mean the fraction of test examples that were correctly classified. The HMM software that we used for the experiments was obtained from HmmLib [18], an HMM software library written in Java. HmmLib is an exact implementation of the HMM algorithm described in Ref. [6].

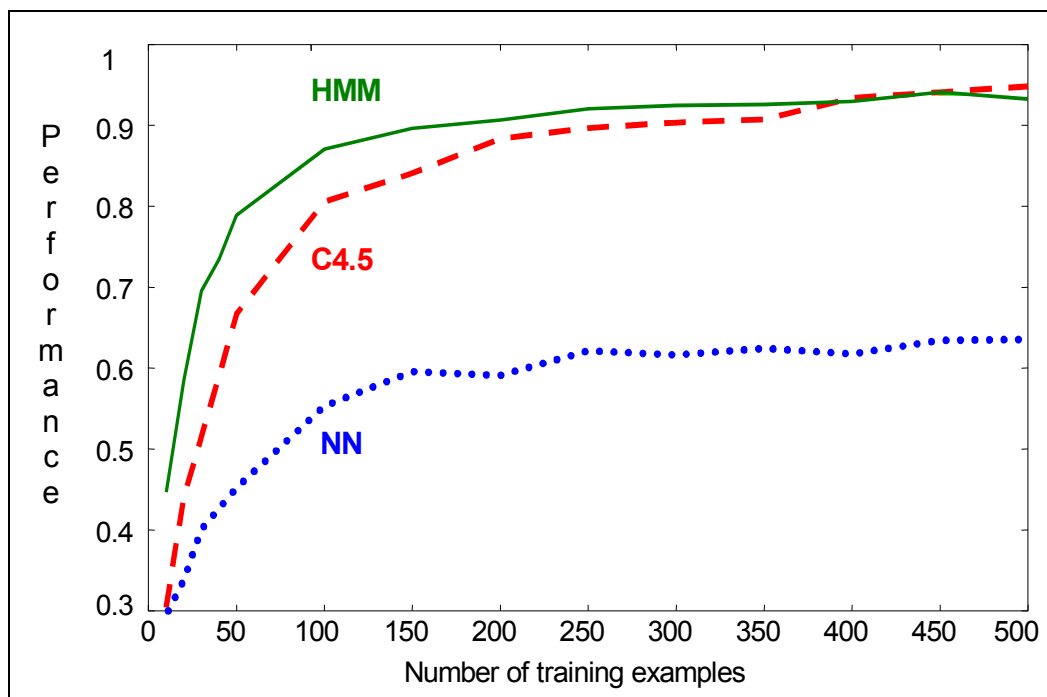


Figure 3. Performance versus number of training examples

We extended the HmmLib software by adding the capability to learn from multiple examples and to process HMMs for all attack categories in parallel. For these test results, we used 10 to 500 training examples, 100 test examples, and averaged the results over 100 samples. The training and test populations were obtained using sampling without replacement (i.e., the training and test populations were disjoint). The parameters used for the neural net, which were default values for the WEKA [17] neural network implementation, are as follows (note that the WEKA neural network uses the standard back propagation algorithm for learning):

Learning rate = .3
Momentum = .2
Number of epochs = 500
Number of hidden layers = 1
Nodes per hidden layer = 20

Figure 4 presents the confusion matrix [19], precision, recall, and F-measure data for our current system. The confusion matrix presents the possibility of assigning an input example of a particular class to a particular output category, while precision, recall, and F-measure are defined as follows [20]:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$\text{F-measure} = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}}$$

where tp is the number of positive examples in the input data that have been correctly identified by the classifier; fp is the number of negative examples that have been incorrectly identified as positive by the classifier; fn is the number of positive examples that have been identified as negative by the classifier; α is a parameter designed to allow a weighting of the importance ascribed to precision or recall; P is precision and R is recall. For equal weighting of precision and recall, as was done for our experiments, α is set to 0.5.

Figure 4 was generated from the 300 training example case data point shown above in Figure 3. The confusion matrix is shown in the box in Figure 4. Each element of the confusion matrix indicates the proportion of examples from a particular category that were assigned to another category by the classifier. For example, the value of “.96” in the upper left cell of the matrix indicates that 96 percent of the test examples in

When compared with the C4.5 algorithm, the HMM algorithm is competitive with C4.5 at all levels and significantly out-performs C4.5 at the lower training levels. For instance, with only 10 examples from which to learn, the HMM algorithm scores 45 percent versus 30 percent for the C4.5 algorithm. In addition, the HMM algorithm learns much more quickly. By 50 training examples, the HMM algorithm is already achieving 79 percent performance versus 67 percent for C4.5.

The neural network algorithm was clearly inferior to the HMM for this domain, where its likelihood of correctly classifying an example only slightly exceeded chance at the highest training level. It should also be noted that the neural network required vastly more time than the other two algorithms to complete its training. The significant performance advantage displayed by the HMM at low training levels indicated potential applicability to the “rare data” problem: the case where only a few examples of a complex Internet attack are available. category 1 were correctly assigned to category 1 by the classifier.

The second column of the figure shows the average number of test examples in a category, averaged over the number of iterations for testing. The row in the figure that is just above the confusion matrix, labeled “Examples assigned to category,” indicates the average number of examples that were classified into a particular category. Points along the diagonal of the confusion matrix (where one would like to see “1’s”) are shown in bold. Precision, recall, and F-measure are given in the last three rows of the figure (just below the confusion matrix). Each element in these rows corresponds to a given category, starting a category 1 on the left and running to category 13 at the far right. Values containing question marks indicate cases where the denominator of the corresponding expression was zero.

In order to avoid data security issues, the category labels do not indicate the specific category definitions. It can be said that the progression among the categories is from less interesting to more interesting in terms of analyst interest. Category 1 is the least interesting category, and category 13 is the most interesting. The point should also be made that the data used for our analysis is “real” data corresponding to suspected intrusions.

As can be seen in Figure 4, good performance can be obtained from this level of training (overall performance is 92.5 percent). However, cases for which there are few examples tend to have a lower performance result. For example, category 6 had, on the average, only .2 examples in the test population; it was classified with an accuracy of 35 percent. Category 7 also had, on average, .2 examples and achieved only 15 percent classification accuracy.

Training examples 300		Test examples 100		Performance 0.9255		Sampling iterations 100								
		Examples assigned to category												
		21.1	0.0	15.0	29.7	4.6	0.3	0.0	9.8	2.6	12.1	4.4	0.3	0.0
Cat	Exs in cat	Confusion matrix												
1	20.2	**0.96**	0.00	0.02	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.0	0.80	**0.20**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	15.7	0.02	0.00	**0.92**	0.01	0.03	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00
4	32.4	0.01	0.00	0.00	**0.90**	0.06	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00
5	2.2	0.05	0.00	0.03	0.05	**0.83**	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00
6	0.2	0.00	0.00	0.00	0.00	0.00	**0.35**	0.00	0.65	0.00	0.00	0.00	0.00	0.00
7	0.2	0.85	0.00	0.00	0.00	0.00	0.00	**0.15**	0.00	0.00	0.00	0.00	0.00	0.00
8	9.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
9	2.5	0.03	0.00	0.01	0.03	0.00	0.02	0.00	0.02	**0.90**	0.00	0.00	0.00	0.00
10	12.7	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	**0.93**	0.00	0.01	0.00
11	4.4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00
12	0.1	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.33	0.00	**0.17**	0.00
13	0.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.00**
Precision		0.92	1.00	0.96	0.98	0.47	0.25	1.00	0.96	0.87	0.97	1.00	0.06	?
Recall		0.95	0.25	0.92	0.90	0.81	0.35	0.13	1.00	0.93	0.93	1.00	0.17	?
F-measure		0.93	1.00	0.94	0.93	0.60	0.89	0.89	0.98	0.90	0.95	1.00	0.67	?

Figure 4. Confusion matrix, precision, recall, and F-measure

Considering the precision, recall, and F-measure calculations, these values are very good, except for those categories where there are insufficient training and test examples. In addition, for categories 2 and 7, the classifier attained a high value for precision, but low value for recall. This means that for those categories the classifier had few false positives but had a high proportion of false negatives. This could mean that the training population, with only a few examples, had almost no positive examples, causing the classifier to learn that almost all examples should be labeled as negative. Figure 5 shows the ROC curves [21] obtained from the testing activity. Note that the ROC curves presented in Figure 6 were obtained from the category 1 testing data. ROC curves for the other categories are similar. The point that should be made regarding this figure is that the ROC curves gain most of their performance in the region from 0 to 40 percent false alarm rate, and the ROC curves improve as the training level is increased.

The improvement in performance with training level is shown more clearly in Figure 6, which presents area under the ROC curve as a function of training level. Note that categories for which there were insufficient test examples to generate a valid curve are not included in Figure 6. Categories that start out at lower performance values increase performance with training, as expected. However, the top two categories, 8 and 11, actually appear to decrease in performance as training is increased, an effect for which we currently have no explanation.

The Rare Data Problem

In general, the most dangerous attacks happen only rarely. Therefore the training set for the HMM contains only a few examples of these attack types. In addition, the input alert data stream contains many low priority alerts that tend to mask identification of these rare,

high priority attacks. For example, in the confusion matrix of Figure 4, there are very few example of category 2, and almost all of these examples are mistakenly assigned to category 1, a very common category. Similarly, the examples of category 5 (rare) are also assigned to category 1, and the examples of category 6 are mistakenly assigned to category 8. This problem is known in the literature as the problem of “skewed distributions” or “imbalanced data sets” [22], [23], [24], [25]. This phenomenon has only recently been addressed from within the ML community and to date is not well explored. Because of the incipient nature of the work in this area, in this section we only provide a brief list of approaches that have been used to attack the rare data problem in the past. It is our intention to look more deeply at this problem in the next stage of our research

The standard approaches for handling the rare data problem are as follows:

Over-sample the rare data population [25]. In this case, the samples of the rare data population are duplicated so as to increase their representation during the learning process. *Under-sample the abundant data population* [25]. This is simply the inverse of the method described above. In both cases the objective is to obtain a balance between the most populous sample types and the rare samples.

Apply recognition filters to the data to eliminate samples of one class or another [25]. “Boost” the performance of the algorithm against rare data [24].

Teach the classification algorithm in two stages consisting of: 1) training the classifier to properly classify all of the positive examples, and 2) amending the classifier by teaching it to properly classify all of the negative examples that were improperly classified as positive during the first stage [22], [26].

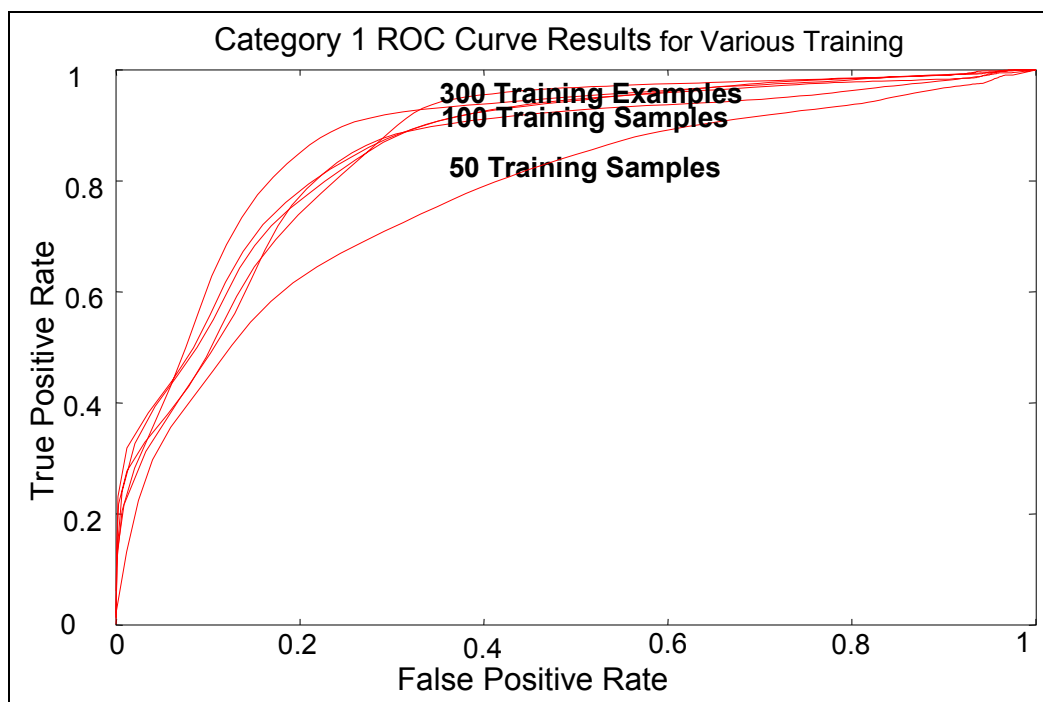


Figure 5. ROC curves as a function of training level

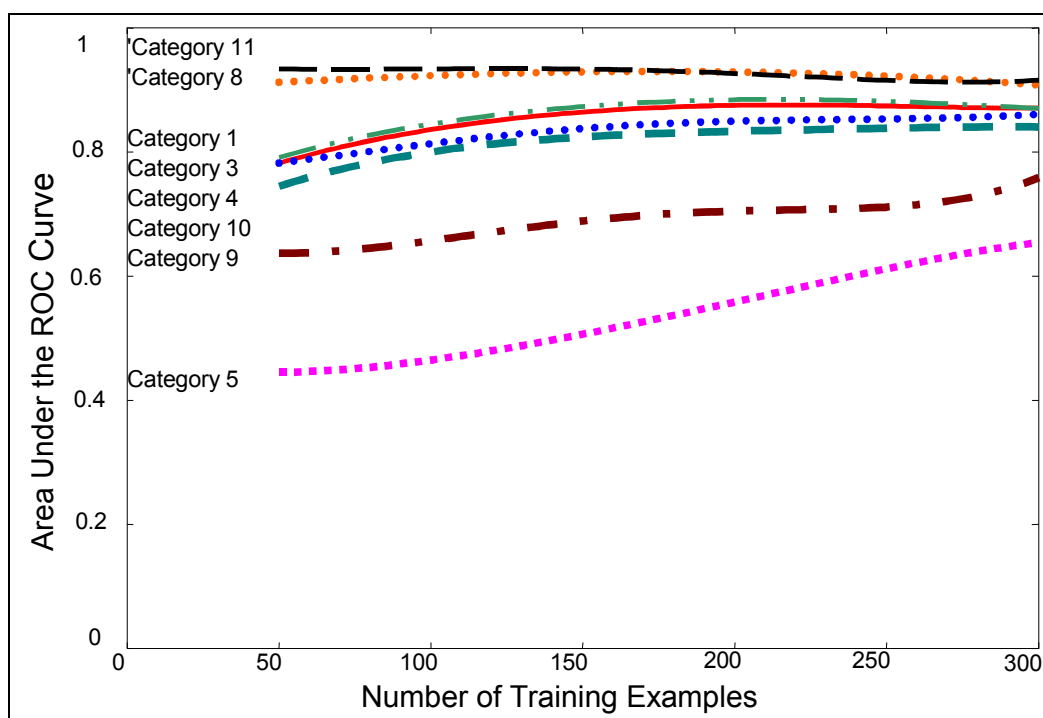


Figure 6. Area under the ROC curve as a function of training level

Each of the approaches listed has strengths and weaknesses in the context of the intrusion detection problem. When only a few examples (i.e., < 10) of a particular attack exist in the training data, none of these approaches may work and we may have to resort to other approaches to solve the rare data problem.

The approaches listed above represent possible starting points for the work that needs to be done in

accounting for rare data in the intrusion detection domain. This work is especially vital in regard to multi-stage attacks, as these attacks are almost always rare.

Future Work

The work done so far concerning coordinated Internet attacks has identified areas where further work is required. One problem that needs to be addressed is

the “rare data” problem, as we have discussed above. In addition, we need to investigate the “partial data” problem, that is, how to identify slowly developing attacks at an early stage. Many coordinated Internet attacks take place over a long period of time (in some cases, weeks or months).

We need a method for detecting early actions that are likely preludes to more complex intrusions. Such early detections would allow us to monitor these “slow rolling” attacks as they are developing and take appropriate action. Some work that may be applicable to this problem has been done at the University of Waikato on prediction by partial matching (PPM) [27], a technique that has been used in text mining for recognizing partial sequences. The final area requiring research is the problem of noise contaminating the input data [28]. This problem can occur when there are problems with the communications channel, particularly in the case of wireless communications, or when the sensors malfunction, or when the hacker mounting the intrusion simply makes a mistake in the attack sequence. This area has been studied previously in the field of automated knowledge base repair [29],[30].

References

1. Ourston, D., Matzner, S., Stump, W., Hopkins, B., Richards, K. Identifying Coordinated Internet Attacks. In: Proceedings of the Second SSGRR Conference. Rome, Italy (2001)
2. Valdes, A., Skinner, K.: Probabilistic Alert Correlation. In: The Proceedings of Recent Advances in Intrusion Detection (RAID). Springer Verlag Lecture Notes in Computer Science (2001)
3. Lane, T., Brodley, C.: Temporal Sequence Learning for Anomaly Detection. In: 5th ACM Conference on Computer & Communications Security. San Francisco, California, USA (1998) 150-158
4. Aha, D.W., Kibler, D., Albert, M.K.: Instance-Based Learning Algorithms, *Machine Learning* 6 (1991) 37-66
5. Manganaris, S., et al.: A Data Mining Analysis of RTID Alarms. In: 2nd International Workshop on Recent Advances in Intrusion Detection. Purdue University, West Lafayette, Indiana, USA (1999)
6. Rabiner, L.R., Juang, B.H.: An Introduction to Hidden Markov Models. In: IEEE ASSP Magazine, 1986
7. Warrender, C., Forrest S., and Pearlmuter, B.: Detecting Intrusions Using System Calls: Alternative Data Models. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy. (1999)
8. Huang, X.D., Ariki, Y., Jack, M.A.: Hidden Markov Models for Speech Recognition, Edinburgh University Press (1990)
9. Wen, Y.: Text Mining Using HMM and PPM. Master's thesis. Department of Computer Science, University of Waikato (2001)
10. Bunke, H., Caelli, T. (eds.): Hidden Markov Models: Applications in Computer Vision. World Scientific, Series in Machine Perception and Artificial Intelligence, Vol. 45. (2001)
11. Boys, R.J., Henderson, D.A., Wilkinson, D.J.: Detecting Homogeneous Segments in DNA Sequences by Using Hidden Markov Models. *Applied Statistics* 49(2) (2000) 269
12. Green, J., Marchette, D., Northcutt, S., and Ralph, B.: Analysis Techniques for Detecting Coordinated Attacks and Probes. In: Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring (1999)
13. Cuppens, F.: Managing Alerts in a Multi-intrusion Detection Environment. In: 17th Annual Computer Security Applications Conference. New Orleans, Louisiana (2001)
14. Bloedorn, E., et al.: Data Mining for Network Intrusion Detection: How to Get Started. Mitre Corporation. http://www.mitre.org/support/papers/tech_papers_01/bl_oedorn_datamining/index.shtml (2001)
15. Sinclair, C., Pierce L., Matzner, S.: An Application of Machine Learning Techniques to Network Intrusion Detection. Proceedings of the 15th Annual Computer Security Applications Conference (1999)
16. DeGroot, M.H.: Probability and Statistics. 2nd edn. Addison Wesley Longman
17. Witten, I.H., Eibe, F.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (2000) Ch. 8. <http://www.cs.waikato.ac.nz/ml/weka/>
18. Yoon, H.: Hmmlib: Hidden Markov Model Library in Java. <http://www.vilab.com/hmmlib/home.html>
19. Townsend, J.T.: Theoretical Analysis of an Alphabetic Confusion Matrix. *Perception and Psychophysics* 9(1A) (1971) 40-50
20. Manning, C., Schutze, H: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts, USA 267-270
21. Egan, J.P.: Signal Detection Theory and ROC Analysis. Series in Cognition and Perception. Academic Press, New York (1975)
22. Agarwal, R., Joshi, M.: PNRule: A New Framework for Learning Classifier Models in Data Mining (A Case-Study in Network Intrusion Detection). Technical Report TR 00-015, Department of Computer Science, University of Minnesota, USA (2000)
23. Provost, F., Fawcett, T.: Robust Classification for Imprecise Environments. *Machine Learning* 42 (2001) 203-231
24. Freitag, D., Kushmerick, N.: Boosted Wrapper Induction. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence. Austin, Texas, USA (2000)
25. Japkowicz, N.: Learning from Imbalanced Data Sets: A Comparison of Various Strategies. In: Papers from the AAAI Workshop on Learning from Imbalanced Data Sets. Technical Report WS-00-05, AAAI Press, Menlo Park, California, USA (2000)
26. Joshi, M., Agarwal, R., Kumar, V.: Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction. In: Proceedings of ACM SIGMOD'01 Conference on Management of Data. Santa Barbara, California, USA (2001)
27. Hand, D., Till, R.: A Simple Generalization of the Area under the Roc Curve for Multiple Class Classification Problems. *Machine Learning* 45 (2001) 171-186
28. Eskin, E.: Anomaly Detection over Noisy Data Using Learned Probability Distributions. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000) (2000)

29. Ourston, D.: Using Explanation-Based and Empirical Methods in Theory Revision. Ph.D. thesis. The University of Texas, Austin, Texas, USA (1991)
30. Mooney, R. J., Ourston, D.: Theory Refinement with Noisy Data. Technical Report AI91-153, Artificial Intelligence Laboratory. The University of Texas, Austin, Texas, USA (1991)