Temporal Sequence Learning and Data Reduction for Anomaly Detection

Terran Lane Carla E. Brodley
School of Electrical and Computer Engineering and
the COAST Laboratory
Purdue University, West Lafayette, IN 47907-1287
email: {terran,brodley}@ecn.purdue.edu

Abstract

The anomaly detection problem can be formulated as one of learning to characterize the behaviors of an individual, system, or network in terms of temporal sequences of discrete data. We present an approach to this problem based on instance based learning (IBL) techniques. To cast the anomaly detection task in an IBL framework, we employ an approach that transforms temporal sequences of discrete, unordered observations into a metric space via a similarity measure that encodes intra-attribute dependencies. Classification boundaries are selected from an *a posteriori* characterization of the valid user's behaviors, coupled with a domain heuristic. An empirical evaluation of the approach on user command data demonstrates that we can accurately differentiate the profiled user from alternative users when the available features encode sufficient information. Furthermore, we demonstrate that the system detects anomalous conditions *quickly* — an important quality for reducing potential damage by a malicious user. We present several techniques for reducing the data storage requirements of the user profile, including instance selection methods and clustering. An empirical evaluation shows that a new greedy clustering algorithm reduces the size of the user model by 70% with only a small loss in accuracy. A comparison of the greedy clustering technique to clustering with K-centers shows that greedy clustering is preferable in terms of accuracy and computation time for this domain.

1 Introduction

In this paper, we examine the problem of anomaly detection as one of learning to characterize the behaviors of an individual, system, or network in terms of temporal sequences of discrete data. Although we focus here on user oriented anomaly detection at the level of shell command input, the methods we present are generalizable to learning on arbitrary streams of discrete events such as GUI events, network packet traffic, or system call traces.

The anomaly detection problem is a difficult one, especially at the level of user command traces. It encompasses a broad spectrum of possibilities, from a trusted system user turning from legitimate usage to abuse of system resources, to system penetration by sophisticated and careful hostile outsiders, to one-time use by a co-worker 'borrowing' a workstation, to automated penetrations launched by a relatively naive attacker via a scripted attack sequence. Time spans of interest vary from a few seconds to months. Patterns may appear only in data gathered from a number of different hosts and networks, possibly spread over thousands of miles geographically. The amount of available data to sift through can be truly staggering, as security officers may be responsible for overviewing thousands of hosts, each of which can generate megabytes of audit data per hour. Selection of the data sources of interest can also be difficult. Do the patterns of interest evidence themselves most clearly in command data, system call traces, network activity logs, CPU load averages, disk access patterns, or any of the hundreds of other possible sources? The patterns of interest may be corrupted by noise or interspersed with examples of normal system usage. Indeed, normal usage may vary greatly as the user changes tasks or software and learns new behaviors and command actions. Differentiating innocuous anomalies from those associated with actual abuse, misuse, or intrusion is a further difficulty. On top of all of these difficulties, a practical security system must be accurate; false alarms will reduce user confidence in the system while falsely accepting anomalous or hostile activities will render the system useless.

Subsets of the general problem have been addressed by specialized techniques. Short term ('hit and run') attacks and attacks launched by automated scripts can often be detected by pattern matching to databases of known attack patterns (for example, [Kum95, SCCC$^+$96]). Similarly, there are numerous free and commercial programs for detecting the presence of known vulnerabilities and viruses by signatures, [FV95, Gor96].

We are interested in the subset of anomaly detection oriented to longer term patterns, in which known misuse signatures are insufficient to distinguish the space of possible anomalies. This subset covers not only intrusions but also hostile activities by a trusted user and even relatively 'innocuous' policy violations such as inappropriate use of system resources by an authorized user. We take a machine learning viewpoint of this problem, in which the task is to train a classifier with known 'normal' data to distinguish normal behaviors from anomalous. The field of machine learning (and artificial intelligence, in general) is strongly motivated by pattern detection and analysis problems and possesses many techniques for different pattern recognition problems.

To approach anomaly detection as a machine learning task, we must define both the learning model and representational format for the input data. We hypothesize that temporal interactions carry a significant amount of identifying information, and so our learning model should explicitly examine such interaction. We present a source independent representation that encodes some temporal aspects of a data stream.

One popular and highly general class of machine learning techniques is *instance based learning* (IBL), [AKA91]. In this model, the concept of interest is implicitly represented by a set of instances that exemplify the concept (the instance dictionary). A new instance is classified according to its relation to stored instances. A typical scheme is k-nearest-neighbor classification, in which a new instance is given the label of the majority of the k dictionary instances closest to it, where 'closest' is a domain specific measure but is often taken to be the Euclidean distance. IBL techniques may be contrasted to learning techniques that build explicit models of the data, such as summary statistics or decision trees [Qui93].

Some work is required to adapt the anomaly detection task to the IBL learning framework. In particular, we need to settle on a fixed-length vector (feature vector) representation of the data and to define the concept of 'closeness' or similarity of two vectors. We also need a different decision process than the popular nearest-neighbor rules. Because the space of possible malicious behaviors and intruder actions is potentially infinite, it is impractical to characterize normal behavior as a contrast to known abnormal behaviors. It is also desirable, for privacy reasons, that an user based anomaly detection agent only employ data that originates with the profiled user or is publicly available. This requirement leads to a learning situation in which only positive instances are available. Learning from positive examples only presents a challenge for classification as it can easily lead to overgeneralization [Iba79].

A widely acknowledged difficulty with instance based learning techniques is the overhead incurred by explicitly storing a set of class exemplars. In a dynamic environment with no fixed set of training data, such as anomaly detection, the size of the instance dictionary can conceivably grow without bound. Thus, it is necessary to consider data reduction techniques to reduce the resource consumption of the IBL system. Possible solutions include removal of instances from the dictionary and re-representation of instances in another, less space intensive, form. In this paper, we explore the use of clustering algorithms to reduce dictionary size. In this formulation, a group of similar instances is replaced with a single exemplar instance.

In the rest of this paper, we examine methods for representing the anomaly detection domain as an IBL task, including a temporal encoding of discrete data streams and a definition of similarity that encodes some aspects of temporal sequence data. We present a clustering technique for data reduction in this domain. We finish with an empirical examination of performance at differentiating users under this learning scheme.

2 Learning from Temporal Sequence Data

Many traditional approaches to learning from temporal sequence data are not applicable to the anomaly detection domain, when the base data consists of discrete, unordered (i.e. nominal-valued) elements such as command strings. For time series of numeric values, techniques such as spectral analysis [OS89], principle component analysis [Fuk90], nearest neighbor matching, (γ, ϵ)-similarity [BDGM97, DGM97], and neural networks [CO96] have proven fruitful. Such techniques typically employ a Euclidean distance or a related distance measure defined for real-valued vectors.

There are a number of learning algorithms that are amenable to learning on spaces with nominal-valued attributes, but they typically make the assumption of independence of attributes. For example, decision trees [Qui93] are well suited to representing decision boundaries on discrete spaces. The bias used to search for such structures generally employs a greedy search that examines each feature independently of all others. This bias ignores internal relations arising from causal structures in the data generating process.

One method of circumventing this difficulty is to convert the data to an atemporal representation in which the causal structures are represented explicitly. Norton (1994) and Salzberg (1995) each independently used such a technique for the domain of learning to recognize coding regions in DNA fragments. DNA coding, while not temporal, does exhibit interrelations between positions that are difficult for conventional learning systems to acquire directly. The features extracted from the DNA sequences were selected by domain experts, and cannot be generalized to other sequential domains. Although such an approach could be applied to the anomaly detection domain, it would require considerable effort on the part of a domain expert, and the developed features would apply only to that data source. We are interested in developing techniques that can be applied across different data sources and tasks.

Our approach is based on a similarity measure that transforms the native data format (a stream of discrete events) into a metric space. Classification is performed via an IBL technique that selects decision thresholds not on distance to members of different classes, but on probability of falling within known patterns. In this section, we describe the similarity measure we employ and describe the classification procedure in the transformed space. We end with a description of several data reduction methods for this domain.

2.1 The Similarity Measure

Currently, our similarity measure treats only sequences of tokens of equal, fixed length. Tokens may be any symbols drawn from a discrete, finite, unordered alphabet (e.g. GUI events, UNIX command names, keystrokes, system calls, etc.). For a length, l, the similarity between sequences $X = (x_0, x_1, \ldots, x_{l-1})$ and $Y = (y_0, y_1, \ldots, y_{l-1})$ is defined by the pair of functions:

$$w(X,Y,i) = \begin{cases} 0 & \text{if } \quad i < 0 \text{ or } \mathrm{x_i} \neq \mathrm{y_i} \\ 1 + w(X,Y,i-1) & \text{if } \quad x_i = y_i \end{cases}$$

(where $w(X,Y,i) = 0$ for $i < 0$ so that $w(X,Y,0)$ is well defined when $x_0 = y_0$) and

$$\mathrm{Sim(X,Y)} = \sum_{\mathrm{i=0}}^{\mathrm{l-1}} \mathrm{w(X,Y,i)}.$$

The converse measure, distance, is defined to be:

$$\mathrm{Dist(X,Y)} = \mathrm{Sim_{max}} - \mathrm{Sim(X,Y)}.$$

The function $w(X, Y, i)$ accumulates weight linearly along matching subsequences, and $\mathrm{Sim}(X, Y)$ is the integral of total weight over time. In the limiting case of identical sequences, this measure collapses to $\mathrm{Sim}_{\max} = \mathrm{Sim}(X, X) = \sum_{i=1}^{l} i = \frac{l(l+1)}{2}$. Thus, a run of contiguous matching tokens will accumulate a large similarity, while changing a single token in the middle of the run can greatly reduce the overall similarity. This measure depends strongly on the interactions between adjacent tokens as well as comparisons between corresponding tokens in the two sequences (i.e. tokens at the same offset, i, within each sequence). The sequence length, l, is a user-dependent parameter and was explored in [LB97a] where $l = 10$ was found to be an acceptable compromise across users.

A user profile is a collection of sequences, $\mathbf{D}$, selected from a user's observed actions.[1] The similarity between the profile and a newly observed sequence, X, is defined to be:

$$\mathrm{Sim}_{\mathbf{D}}(X) = \max_{Y \in \mathbf{D}} \{\mathrm{Sim}(Y, X)\}.$$

This rule is related to the 1-nearest-neighbor classification rule, although we are not actually performing classification at this stage but, rather, are defining similarity to known patterns. We examined the possibility of using an average similarity to the entire profile, but found that such a measure had much lower accuracy than the measure given here. Such an average decreases the ability of the classifier to resolve fine-structure patterns in the classification space.

The design of this similarity measure was motivated by the observation that human-computer interaction is a fundamentally causal process; the computer responds to the human's request and the human, in turn, responds to the computer's output. Weighting of adjacent matches is an attempt to capture the short-term causal linkages in the user's input stream. Other similarity measures for the anomaly detection domain have been examined in [LB97c], and it was shown that the similarity measure described here is effective for anomaly classification across a number of different profiled users.

2.2 Segmenting the Event Stream

Because our similarity measure is defined only for fixed length sequences, it is necessary to partition the raw event stream into component sequences. This raises the question of optimal sequence alignments: where should each sequence be defined to start? Our approach is *post hoc*, initially segmenting the data stream into all possible overlapping sequences of length l (thereby replicating each token l times). Thus, every position, i, of the event stream is considered to be the starting point for a sequence of length l referred to as the i^{th} sequence or the sequence at time step i. After instance selection (see below), the sequences remaining in the profile are considered to define the desired alignments.

2.3 Classification Procedure

The similarity-to-profile measure defines a transformation from the original, l-ary nominal space to a one-dimensional, real-valued space in which a point set (command trace), $\mathbf{T}$, appears as a probability distribution, $P_{\mathbf{T}}$ over possible similarity values, v. Given sufficiently accurate models of the distribution of normal and abusive actions, we could simply construct a Bayes-optimal decision boundary [Fuk90] and proceed with classification. Because we possess data only from the profiled user, the Bayes-optimal boundary is unobservable to us. Furthermore, for most of the data sets we have examined, the *unweighted* Bayes-optimal threshold is overly critical of the profiled user. In the example of Figure 1, normal (U3) and anomalous (U6) similarity distributions are displayed together with the Bayes-optimal classification threshold and an alternative possible classification threshold (the acceptable false alarm threshold, described below). Sequences whose similarity to the profile falls to the right of the classification threshold are labeled normal while points falling to the left are labeled abnormal. The area under distribution U3 and to the left of the threshold is then the false alarm probability (the probability of the valid user being falsely accused of being anomalous) while the area under distribution U6 and to the right of the threshold is the probability of falsely accepting an anomalous user. In this example, employing the unweighted Bayes-optimal threshold for classification yields an unacceptably high false alarm rate. In light of these considerations, we must seek another method for selecting a decision boundary. Conveniently, the constraints of our domain provide us with a practical heuristic: reduce the false alarm rate. For a given profile, D, we choose an 'acceptable' false alarm rate, r, and set the decision boundaries according to the rule:

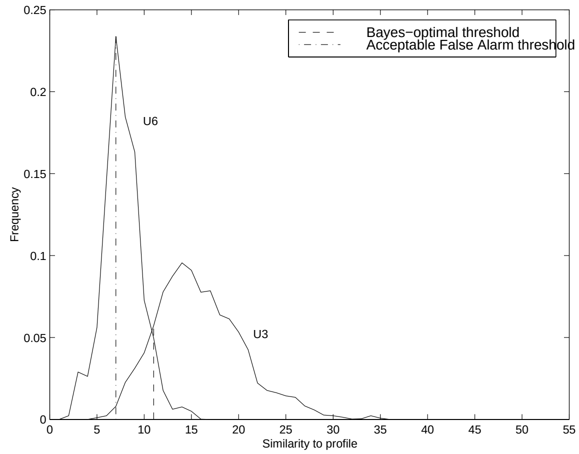

Figure 1: Comparison of unweighted Bayes-optimal decision boundary and acceptable false alarm rate boundary. The rightmost curve (user U3) represents the profiled user.

$$class(v) = \begin{cases} 1 & \text{if} \quad P_{\mathbf{T}}(v) \geq r \\ 0 & \text{if} \quad P_{\mathbf{T}}(v) < r \end{cases}$$

where v is the similarity of a sequence to be classified to D, 1 denotes 'normal', and 0 denotes 'anomalous'. Acceptable false alarm rate is a site-specific value, defined by security policy.

In practice, we have found that the similarity stream, produced by comparing an input data stream to a profile, is far too noisy for effective classification (Figure 2, (**a**)). We attribute the high degree of noise to natural variations in the user's actions and patterns. For example, the user may temporarily suspend writing a paper to deal with urgent incoming email, thus disrupting his or her standard paper writing routine. Such a disruption will appear as a

[1] The question of guaranteeing that the observed history used to profile a user actually originates with that user is a critical one, but we do not examine that problem here, instead taking the known data to be accurate by assumption.

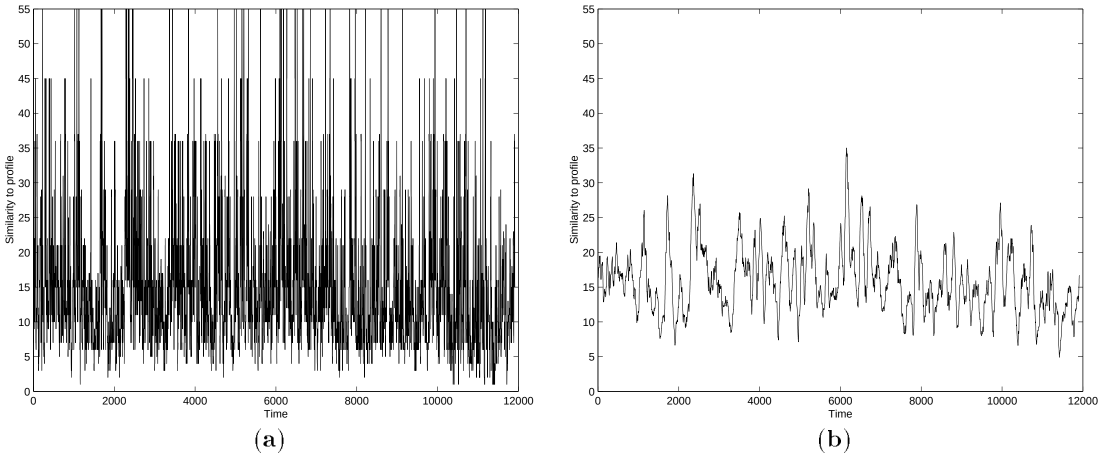

Figure 2: Unsmoothed (**a**) and mean-smoothed (**b**) similarity stream.

spuriously low similarity spike within an overall high similarity period. A time average of the similarity signal yields a much more stable data stream (Figure 2, (**b**)). We therefore employ a noise reduction filter before selecting decision thresholds or performing classification. For the work described here, we employ a trailing window mean value filter defined as:

$$v_{\mathbf{D}}(j) = \frac{1}{W} \sum_{i=j-W+1}^{j} \mathrm{Sim}_{\mathbf{D}}(\mathrm{i})$$

where $\mathrm{Sim}_{\mathbf{D}}(\mathrm{i})$ is the similarity of the i^{th} token sequence to the user profile $\mathbf{D}$, W is the window length, and $v_{\mathbf{D}}(j)$ is the final value of sequence j with respect to $\mathbf{D}$. A small W is desirable because the window length defines the minimum time before any detection can occur. While a great deal of damage can be inflicted in less than the window length, such short term attacks can be more readily handled by matching known attack signatures [KS94]. We are primarily concerned here with the class of long-term, low-profile attacks such as resource theft or industrial data theft.

2.4 Storage Reduction: Instance Selection

A widely acknowledged weakness of instance-based learning algorithms is the large data storage requirement for accurate classification. A number of techniques have been examined for reducing this memory overhead, many of which are reviewed by Wilson and Martinez (1997). In an operational setting, data reduction is critical as the size of the data directly impacts the time required for classification.

We note, first, that the chosen similarity measure selects only a single historical sequence as most similar to a given input sequence. If we assume that the characteristics of a user's behavior change relatively slowly, we can invoke locality of reference to predict that recently matched dictionary sequences will be used again for detection in the near future. This suggests an analogy to tasks in operating systems, such as page replacement, in which some resources must be released in favor of others.

To examine this analogy, we implemented the least-recently-used (LRU) pruning strategy. As new instances are acquired and classification is performed, the profile instance selected as most similar is time-stamped. The profile is constrained to the desired size by removing the least-recently-used sequences. By analogy, we also constructed and tested the pruning heuristics FIFO (equivalent to preserving the most recently stored n sequences), LIFO (preserve the oldest n sequences), and LFU (remove least frequently used sequences). In other work, [LB97b], we have examined the properties of each of these methods. We found that instance selection could reliably reduce the data storage requirements with small or no accuracy losses. The best instance selection method was found to be user dependent.

3 Storage Reduction: Clustering

A second method of reducing data storage is to modify the representation of sets of points within the data space. For example, Salzberg (1991) represented sets of points as hyperrectangles. We propose a greedy clustering algorithm which builds individual clusters consecutively, attempting to minimize the criterion:

$$\mathrm{val}(\mathbf{C}) = \frac{\sum_{\mathbf{x}\in\mathbf{C}} \sum_{\mathbf{y}\in\mathbf{C}} \mathrm{Dist}(\mathrm{x},\mathrm{y})}{|\mathbf{C}|^2}$$

for each cluster $\mathbf{C}$. This measure is a generalization of the mean inter-cluster distance employed for clustering [Fuk90]. From an initial seed point, the cluster is grown incrementally by including the point that increases $\mathrm{val}(\mathbf{C})$ the least, until the halting criterion is reached. Growth is halted when the cluster value reaches a local minimum. Because, in some cases, the cluster value monotonically approaches $\mathrm{Sim}_{\mathrm{max}}$, the halting criterion we actually use is that the first derivative of $\mathrm{val}(\mathbf{C})$ be within ϵ of 0 for some (empirically selected) value of ϵ. As each sequence is added to a cluster, it is removed from the set of available sequences. When the cluster is complete, we define the center of the cluster, $\mathbf{C}_{\mathrm{cent}}$, to be the point possessing the minimum total distance to all other points in $\mathbf{C}$. The similarity between a sequence, X, and a cluster is then $\mathrm{Sim}(\mathrm{X}, \mathbf{C}_{\mathrm{cent}})$.

In practice, we have found that this cluster selection algorithm is somewhat too lenient — it accepts points that decrease the cluster's effectiveness in classification. We solve this in a manner analogous to the pruning process employed in decision tree learning [Qui93]. After growing a single cluster to completion according to the halting criterion, the clustering algorithm removes outlying points and returns them to the pool of available sequences (so that they have the possibility of contributing to different clusters). Our pruning function removes points from the cluster that fall outside the cluster mean radius — i.e. points whose distance to the center is greater than the mean distance to the center of all points in the cluster. Points falling within the mean radius are discarded and removed from further consideration and the final cluster is represented only by its center and mean radius. Because similarity to a cluster is computed only in terms of the cluster center, we realize substantial space savings by discarding all cluster elements other than the center.

The complete clustering algorithm is structurally similar to the single cluster construction algorithm. We sequentially select individual clusters by their ability to maximize the analog of mean intra-cluster distance:

$$\mathrm{val}\{\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_n}\} = \sum_{\mathbf{i=1}}^{\mathbf{n}} \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathrm{Dist}(\mathbf{C}_{\mathbf{i},\mathrm{cent}}, \mathbf{C}_{\mathbf{j},\mathrm{cent}}).$$

In this case, we found the single cluster halting criterion to be ineffective because, typically, all of a data set's points were exhausted before the derivative of the intra-cluster distance approached 0. When we allowed the clustering process to absorb all available points, many of the clusters were found to either not contribute to classification accuracy or to be actively harmful. Instead, we halt the clustering process when the the minimum inter-cluster value of all current clusters falls below a threshold, C. Currently, we select C empirically.

4 Empirical Evaluation

In this section we describe the requirements for an anomaly detection system and the measures that we use to characterize our technique in terms of those requirements. We proceed to present summaries of our experimental results, characterizing the data sets (users) for which our approach is successful. Finally, we demonstrate that the greedy clustering algorithm is effective in reducing profile size while maintaining accuracy, but that K-centers clustering is unable to do so in this domain.

4.1 Performance Criteria

The goal in the anomaly detection task is to identify potentially malicious occurrences while falsely flagging innocuous actions as rarely as possible. We shall denote the rate of incorrectly flagging normal behaviors as the *false alarm* rate and the rate of failing to identify abnormal or malicious behaviors as the *false acceptance* rate. Under the null hypothesis that all behavior is normal, these correspond to type I and type II errors, respectively. For the detector to be practical, it is important that the false alarm rate be low. Users and security officers will quickly learn to ignore the 'security system that cried wolf,' if it flags innocuous behavior too often. Finally, a practical security system must be resource-conservative in both space and time. The goal of security is to enhance productivity, not inhibit it through consumption of system resources.

Detection accuracy does not, however, reveal the full story. A second issue of importance is *time to detection*. This quantity is defined to be the average time between when the detector is initialized and when it flags an anomalous condition, and is a measure of how quickly an anomalous or hostile situation can be detected. In the case of false alarms, the time to detection represents the average time from initialization until a false alarm occurs. Thus, we wish the time to detection to be short for hostile users so that they can be dealt with quickly and before doing much harm, but long for the valid user so that normal work is interrupted by false alarms as seldom as possible.

Because we are examining command line data in this work (see below), we measure all detection times in token counts rather than wall clock time. Token count is more nearly correlated with the quantity of interest — how much damage can be accomplished by a hostile user before detection — than is wall clock time.

4.2 Comparison to Current Anomaly Detection Systems

Baselining our results relative to other well known anomaly detection systems such as (N)IDES [LJ88, Lun90], HAYSTACK [Sma88], and NSM [HDL+90] proved to be impossible. The descriptions of these systems tend to focus on architecture and omit performance measures. Spafford [Spa98] reports being unaware of the publication of any performance measures for intrusion and anomaly detection systems other than IDIOT [Kum95] in refereed forums. IDIOT is an intrusion detection system which employs a pattern matching algorithm to detect known attack signatures in audit data. Its patterns are not intended to generalize to unknown cases, so rather than accuracy, time and space performance measures are reported.

4.3 Data

Of the literally thousands of possible data sources and features that might characterize a system or user, we chose to examine UNIX shell command data. The UNIX operating system is widely used and has been extensively studied in both the security and operating systems communities. The user environment is highly configurable with a rich command language and permits a large range of possible behaviors. In the UNIX model, most user interactions take place through a command line environment (a shell), so command data is strongly reflective of user activities. Finally, there are available mechanisms to make collection of shell command data convenient in the UNIX environment.

Lacking shell traces of actual misuse or intrusive behaviors, we demonstrate the behavior of the detection system on the task of differentiating different authorized users of the UNIX hosts in the Purdue MILLENNIUM machine learning lab. In this framework, an anomalous situation is simulated by testing one legitimate user's command data against another legitimate user's profile. This framework simulates only a subset of the possible misuse scenarios — that of a naive intruder gaining access to an unauthorized account — but it allows us to evaluate the approach. It is to be hoped that the "naive intruder" scenario constitutes a large enough fraction of all attacks that progress in this domain will be of practical benefit. Nonetheless, we acknowledge our inability to generalize these result to broader definitions of abuses until we are able to test these techniques against real misuse data.

We have acquired shell command data from eight different UNIX users over the course of more than a year. The data events were tokenized into an internal format usable by the anomaly detector. In this phase, command names and behavioral switches were preserved, but file names were omitted under the assumption that behavioral patterns are at least approximately invariant across file names. The pattern '`vi <file> gcc <file> a.out`', for example, represents the same class of action regardless of whether `<file>` is `homework1.c` or `cluster.c`. The amount of data available varied among the users from just over 15,000 tokens to well over 100,000 tokens, depending on their work rates and

Tested	Profiled User							
User	U0	U1	U2	U3	U4	U5	U6	U7
U0	99.3	57.0	31.7	61.0	75.1	0.6	38.5	10.1
U1	14.9	92.9	12.4	64.2	16.3	0.9	4.0	6.0
U2	41.3	58.7	94.7	43.6	71.1	0.3	47.9	8.3
U3	64.8	91.7	46.7	90.0	86.4	0.6	69.0	15.1
U4	34.4	21.2	18.6	72.1	92.7	1.3	8.6	3.0
U5	50.4	68.3	39.7	70.3	78.0	99.9	57.2	29.4
U6	41.8	15.4	17.7	82.3	48.7	0.6	91.7	4.7
U7	24.7	11.0	8.7	40.7	22.1	0.6	5.8	96.2

Table 1: Correct classification percentages for all profiles and test sets.

when each user entered and left the study. For uniformity in testing, we selected the earliest (in calendar time) 15,000 tokens from each user, representing an average of approximately six months of usage per person.

4.4 Unclustered Profile Results

Each user's data was split into separate train set (1,500 tokens), parameter selection set (1,500) tokens, and test set (12,000 tokens). Each train set was then used as the basis for a profile, and the classification boundaries were selected according to the distribution of the parameter selection data with respect to the profile. Finally, false accept and false alarm rates were generated for each profile and each test set. We present the results in Table 1 for the best global choice of parameters. The values along the main diagonal of this table are generated by comparison of a user to his or her profile and represent percent true acceptance. The off-diagonal elements are generated by testing one user against a different user's profile and represent percent true detections. These results were achieved with acceptable false alarm rate $r = 2\%$, sequence length $l = 10$, and mean filtering with window length $W = 21$ sequences. Note that if we vary the parameters on a per-user basis we can achieve higher accuracies.

A number of points are key in this table. First, the main diagonal (correct classifications of the valid user) is uniformly high. For six of the users the achieved self-detection accuracy is lower than the 98% specified by the chosen acceptable false alarm rate. This is a result of the parameter selection data (used for decision threshold selection) failing to reflect the true distribution of the user's behaviors. This problem also leads to increased false accept rates. The most dramatic illustration of this problem is visible in user 5's profile. On examination of user 5's command data, we discovered that the user had largely changed tasks between generation of the training and threshold selection data (from tasks concerned mainly with system maintenance to programming tasks). In this case, the particular skew of the parameter estimation data resulted in an extremely wide range of similarities, yielding decision thresholds that classified nearly everything as the valid user. The overly lenient decision boundaries also produced a spuriously high self-detection accuracy.

A second source of false accept error is demonstrated in Figure 3. Here, U1 (the valid user) and U7 have many behaviors in common — mostly 'generic' account maintenance such as directory creation and file copy and remove operations. This high degree of similarity is reflected in the substantial overlaps in the similarity distributions, making differentiation impossible within this space. By contrast, U3 was engaged mainly in programming and writing during this time. There are two possible sources for the degree of overlap between U1 and U7. First, the underlying observations do not encode sufficient information to distinguish the two users. Many other data sources are available for user profiling and could be used in conjunction with the techniques presented here in an operational security system.[2] The second, and more fundamental, source of error is in the similarity measure itself. The measure presented in this paper is fairly coarse (having only $O(l^2)$ possible values for a sequence length of l), and models only a single type of temporal interaction. We are currently investigating more sophisticated similarity measures such as edit distance [CLR92].

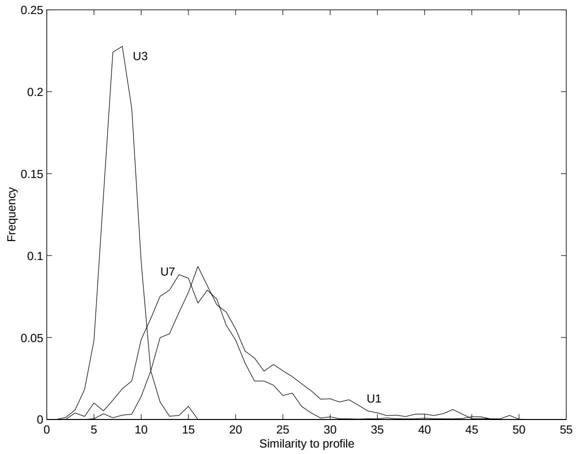

Figure 3: False accept errors: U7's data bears high resemblance to the profiled user's (U1).

4.5 Time to Detection

As explained previously (Section 4.1), we also wish to examine the mean time to detection for the base system. We measured detection times on data from the same users tested above, with false alarm rate $r = 2\%$, sequence length $l = 10$ tokens, and smoothing window length $W = 81$ sequences. The longer window length improves overall time to detection figures and is presented here to demonstrate potentially achievable times. The relation between window length and time to detection is covered in more detail in [Lanar]. For reasons of time, smaller subsets of the available data were chosen, with 5,000 tokens devoted to training, 1,000 to parameter selection, and 1,000 to test.[3] A larger training set was chosen to mitigate the effects of behavioral changes that we see in the previous section, by representing a larger range of behaviors in the user profile.

Time to detection values for all profile/test pairs are given in Table 2. Here we wish the values to be high for a user tested against him or herself (indicating infrequent false alarms), but small for other pairings (indicating rapid detection of anomalous circumstances on average). Thus, we desire large values on the main diagonal of the table, and small values off the diagonal. Delays in time to detection introduced by the sequence length ($l = 10$ tokens,

[2] A number of such data sources are described in [Den87, LJ88, Sma88, HDL+90].

[3] In the extended version of this paper, we will present unified results under the same parameter settings used above, though we expect such changes to have little impact on the general nature of the results.

Tested	Profiled User							
User	U0	U1	U2	U3	U4	U5	U6	U7
U0	58.8	76.6	2.8	3.7	2.1	21.0	0.0	0.0
U1	20.3	172.7	2.1	1.8	1.9	48.0	0.0	0.6
U2	0.0	0.0	94.3	0.0	0.6	0.0	0.0	0.0
U3	335.4	395.4	133.7	376.1	28.8	110.9	181.8	70.6
U4	58.4	67.9	0.5	6.7	157.7	12.0	0.1	6.8
U5	238.1	229.5	227.6	456.0	171.5	108.6	456.0	456.0
U6	179.0	100.8	21.5	44.9	56.5	50.0	205.5	33.6
U7	0.0	7.8	8.5	0.9	0.9	1.0	8.0	259.5

Table 2: Times to detection for each profile and test data set.

see Section 2.1) and the noise suppression smoothing window ($W = 81$ sequences, see Section 2.3) have been omitted from this table, as they represent constant factors across all tests.

We see from Table 2 that the detection system is displaying desirable behaviors. In general, the time to detection for the profiled user (i.e. time to generation of a false alarm, appearing on the main diagonal) is long relative to time to detection for alternate users against that profile (i.e. time to detection of an anomaly, in the non-diagonal elements and reading column-wise).

Examination of the raw classification data reveals that the time to detections are not equivalent to the (inverse) mean detections per unit time. Specifically, the false alarms tend to occur in tight clusters interspersed between long strings of true accepts, yielding an overall long time to detection for the true user. True detections, on the other hand, tend to be more evenly distributed, yielding a shorter expected time to detection. Indeed, in many cases, time to detection is 0.0, indicating that, on average, the anomalous situation is detected in $l+W = 91$ tokens, the minimum possible time. Thus, this detection system is biased toward detecting anomalous conditions *quickly*, while generating false alarms in rare clusters. This type of behavior is desirable because a valid user wishes to be bothered as little as possible and a tight group of false alarms can be investigated and disregarded as a group, while a malicious user can be discovered quickly from only a single true alarm.

4.6 Clustering Techniques

To examine the effectiveness of the greedy clustering algorithm, we produced clustered versions of all of the user profiles from the first experiment (Section 4.4) for various values of C (the clustering halting threshold). As baseline comparison, we implemented the K-centers clustering algorithm. K-centers is an iterative clustering algorithm that attempts to make class assignments to K different clusters simultaneously through gradient descent on the log-likelihood parameter space, and is a special case of the EM (expectation maximization) algorithm, [Rip96].[4] Table 3 shows the average and standard deviation of false alarm rate for each user (U0–U7) and clustering method. These values were obtained with greedy clustering halting criteria ranging across the values $\epsilon = 0.005$ (single cluster) and $C \in \{0.25, 0.5, 0.75\}$ (all clusters). K-centers was given $K \in \{50, 75, 100\}$ and allowed to run 10,000 iterations. All algorithms were run for both mean and median filters.

We see that the false alarm rates are generally low and that greedy clustering matches or outperforms k-centers clustering for five of the users. Both clustering methods have higher false alarm rates than does unclustered.

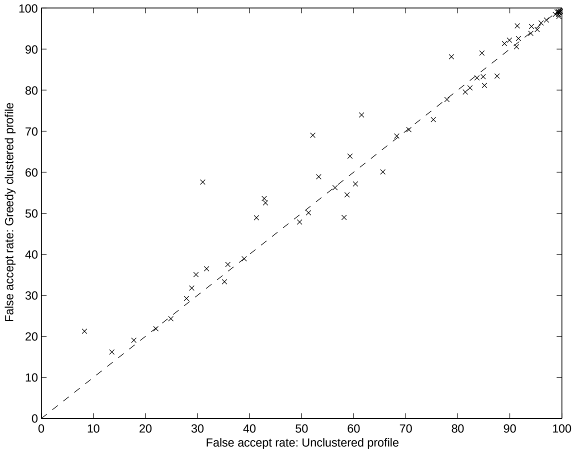

Figure 4: Relative false accept rates for the unclustered and greedy clustered profiles. The diagonal indicates equal performance.

The relative false accept rates of the unclustered and greedy clustering approaches are displayed in Figure 4. Here, the false accept rate for greedy clustering appears on the vertical axis while the rate for unclustered appears on the horizontal. Each plotted point is a single user/profile pair and the diagonal line indicates equal performance. Points to the left of the line indicate superior performance by the unclustered technique, while those below denote higher performance by greedy clustering. We see that most points fall fairly close to the line, indicating that greedy clustering generally incurs only a small accuracy hit. Mitigating the loss of accuracy is data reduction; for these parameters, greedy clustering achieved an average of 70% reduction. Because the classification algorithm runs in time $O(n|D|)$ for a n input sequences and a profile containing $|D|$ instances, we expect to obtain a corresponding 70% improvement in classification speed. In practice, we found that classification of the entire 12,000 token test set on a Sparc Ultra 1 required four minutes without clustering, but only one minute with.[5]

While the greedy clustering algorithm typically formed over 200 clusters per profile, we found that K-centers did not converge to an acceptable solution in a reasonable period of time (10,000 iterations) for $K = 200$. $K = 100$ also

[4] K-centers is closely related to K-means, but requires that the cluster exemplar be drawn from the cluster members rather than being an interpolation of them. For vectors of unordered, discrete elements (such as UNIX commands), no mean value is available, so the exemplar must be a cluster element.

[5] Amortized across the entire six-month period of the test data, this represents less than two thousandths of a percent of the CPU's time.

	Profiled User							
Clustering	U0	U1	U2	U3	U4	U5	U6	U7
Greedy	1.1/0.6	6.6/1.1	4.8/0.9	6.6/4.0	4.4/2.9	0.5/0.4	8.8/3.3	4.4/1.0
K-centers	7.5/3.0	6.6/2.2	5.7/2.4	4.7/1.8	2.4/2.1	3.4/1.7	22.4/3.1	1.7/0.7
None	0.3/0.4	2.2/3.1	1.4/2.1	1.9/3.7	1.6/2.7	0.2/0.3	3.2/4.5	1.4/2.1

Table 3: False alarm rates (percentage) for each clustering method, averaged across all parameter values and filtering methods. Standard deviations are given after the slash.

did not halt, but did achieve somewhat higher performance than did $K = 200$. Its false accept error rate, averaged across all users, parameters, and filtering methods, (77.0%) is dramatically worse than either that of greedy clustering (67.0%) or no clustering (66.6%).

We note, in passing, that the clusters constructed by the greedy clustering algorithm make intuitive sense, in terms of the actions being performed by the underlying sequences. For example, we have identified clusters that correspond to 'programming', 'paper writing', 'reading email', and 'navigating directories'.

5 Conclusions and Future Work

This work has demonstrated a technique for mapping the temporal data occurring in the anomaly detection task onto a space in which IBL learning can be formulated. The results demonstrate that this technique can be useful in such tasks when the underlying data supports sufficient class separation. Furthermore, the system is biased toward detecting anomalous conditions *quickly*, but generating false alarms rarely. We showed a new clustering technique based on sequential, greedy selection of clusters. The greedy clustering technique was able to produce a large saving in storage requirements, with an overall small loss in accuracy. K-centers clustering was unable to match the performance of greedy clustering in this domain in either convergence rate or detection accuracy.

While the algorithm reported here is probably insufficient for *standalone* anomaly detection, there are a number of ways in which it could easily be augmented to improve accuracy and decrease time to detection. We are currently examining other similarity measures for the anomaly detection domain based on edit distance and on hidden Markov models of behavioral patterns. More sophisticated similarity measures would likely improve class separability and, therefore, system accuracy. We are also examining the effects of changes in user behaviors over time. As noted in Section 4.4, a change in user behaviors between gathering the user profile and employing it for user classification can lead to an extremely distorted and inaccurate picture of the user's typical behaviors. We are developing techniques to dynamically adapt the profile to user changes over time, yet not adopt fluctuations arising from hostile or anomalous activities. We note, too, that the techniques developed here are intended to be task independent and we employed little domain knowledge in their design. By adding a greater degree of *a priori* knowledge (e.g. on the advice of a site security specialist), system performance could likely be improved yet further.

The system presented here possesses a number of parameters that must be set: sequence length, l, target false alarm rate, r, noise suppression window length, W, and greedy cluster halting criteria ϵ and C. The results presented here apply single parameter settings to all users and profiles simultaneously. We have found that there is a significant impact of parameter settings on both detection accuracy and time to detection. If the parameters can be properly set on a per-profile basis, then global accuracy can doubtless be further improved. We are currently investigating methods for automatically adapting system parameters to the profiled user.

Finally, the algorithms employed here could be implemented as single detection elements in an overall anomaly detection scheme that also employs alternative data sources such as biometric measurements, resource consumption measurements, and activity time-of-day. Such ensemble classification methods are well known in the machine learning community, [Sch94], and the body of theory surrounding them there could be directly applied to this domain.

In summary, we have presented a machine learning oriented approach to anomaly detection. We have demonstrated that it is possible to learn to distinguish anomalous behavior patterns from normal under this type of framework. We believe that, in general, both the computer security and machine learning communities can benefit from further interactions. The ML community has studied many pattern recognition techniques which could be valuable to a variety of security problems, while computer security tasks present a number of challenging issues that can motivate new research directions for the machine learning community.

References

[AKA91] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[BDGM97] B. Bollobás, G. Das, D. Gunopulos, and H. Mannila. Time-series similarity problems and well-separated geometric sets. In *13th Annual ACM Symposium on Computational Geometry.* Association for Computing Machinery, 1997.

[CLR92] Thomas HOA. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms.* The MIT Press, Cambridge, MA, 1992.

[CO96] T. Chenoweth and Z. Obradovic. A multi-component nonlinear prediction system for the S&P 500 index. *Neurocomputing*, 10(3):275–290, 1996.

[Den87] D. E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[DGM97] G. Das, D. Gunopulos, and H Mannila. Finding similar time series. In *Proceedings of The Fourth International Conference on Knowledge Discovery and Data Mining*, August 1997.

[Fuk90] K. Fukunaga. *Statistical Pattern Recognition (second edition).* Academic Press, San Diego, CA, 1990.

[FV95] D. Farmer and W. Venema. SATAN overview (Security Administrator Tool for Analyzing Networks). Electronic release, Mar 1995. Program documentation for the SATAN/SANTA tool.

[Gor96] S. Gordon. Current computer virus threats, countermeasures, and strategic solutions. White paper, McAfee Associates, 1996.

[HDL$^+$90] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296–304, May 1990.

[Iba79] G. A. Iba. Learning disjunctive concepts from examples. Master's thesis, Massachusetts Institute of Technology, September 1979.

[KS94] S. Kumar and E. Spafford. An application of pattern matching in intrusion detection. Technical Report CSD-TR-94-013, Purdue University, West Lafayette, Indiana, Jun 1994.

[Kum95] S. Kumar. *Classification and detection of computer intrusions.* PhD thesis, Purdue University, W. Lafayette, IN, 1995.

[Lanar] T. Lane. Filtering techniques for rapid user classification. In *Proceedings of the AAAI-98/ICML-98 Joint Workshop on AI Approaches to Time-series Analysis*, 1998, to appear.

[LB97a] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *National Information Systems Security Conference*, Baltimore, MD., 1997.

[LB97b] T. Lane and C. E. Brodley. Detecting the abnormal: Machine learning in computer security. Technical Report TR-ECE 97-1, Purdue University, School of Electrical and Computer Engineering, West Lafayette, IN, 1997.

[LB97c] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, 1997.

[LJ88] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 59–66, 1988.

[Lun90] T. F. Lunt. IDES: An intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*, Rome, Italy, 1990.

[Nor94] S. W. Norton. Learning to recognize promoter sequences in E. coli by modelling uncertainty in the training data. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 657–663, Seattle, WA, 1994.

[OS89] A. Oppenheim and R. Schafer. *Discrete-Time Signal Processing.* Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[Qui93] J. R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Rip96] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK, 1996.

[Sal91] S. Salzberg. A nearest hyperrectangular learning method. *Machine Learning*, 6(3):251–276, 1991.

[Sal95] S. Salzberg. Locating protein coding regions in human DNA using a decision tree algorithm. *Journal of Computational Biology*, 2(3):473–485, 1995.

[SCCC$^+$96] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of The 19th National Information Systems Security Conference.* The National Institute of Standards and Technology and the National Computer Security Center, Oct 1996.

[Sch94] C. Schaffer. Cross-validation, stacking, and bi-level methods for stacking: Meta-methods for classification learning. In P. Cheeseman and W. Oldford, editors, *Selecting Models from Data: Artificial Intelligence and Statistics IV.* Springer-Verlag, New York, 1994.

[Sma88] S. E. Smaha. Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, 1988.

[Spa98] E. H. Spafford. Personal communication, January 1998.

[Wil97] D. R. Wilson. *Advances in instance-based learning algorithms.* PhD thesis, Brigham Young University, Provo, UT, 1997.