Temporal Sequence Learning and Data Reduction for Anomaly Detection

Terran Lane
and
Carla E. Brodley
School of Electrical and Computer Engineering and CERIAS
Purdue University

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.

Categories and Subject Descriptors: D.4.6 [**Software**]: Security and Protection—*anomaly detection, user profiling*

General Terms: Security, Experimentation

Additional Key Words and Phrases: Anomaly detection, user profiling, machine learning, instance based learning, clustering, data reduction, empirical evaluation

Terran Lane's work was supported, in part, by contract MDA904-97-C-0176 from the Maryland Procurement Office, and by sponsors of the Purdue University Center for Education and Research in Information Assurance and Security (CERIAS). Carla E. Brodley's work was carried out under NSF grant number 9733573.
Address: 1285 Electrical Engineering Building, Purdue University, W. Lafayette, IN 47907-1287
Address: Tel: (765) 494-3462; Fax: (765) 494-6440; Email: {terran,brodley}@ecn.purdue.edu

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 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, [Kumar 1995; Staniford-Chen et al. 1996]). Similarly, there are numerous free and commercial programs for detecting the presence of known vulnerabilities and viruses by signatures [Farmer and Venema 1995; Gordon 1996]. In this work, we address the subset of anomaly detection oriented to long term, low profile, human-generated patterns in which known misuse signatures are insufficient to distinguish the space of possible anomalies. By long term, we mean penetrations or exploitations that occur over a period of days or weeks rather than hit and run scenarios and by low profile we mean attacks that are unlikely to flag conventional alarms (such as data theft and as opposed to data destruction or denial of service which are more likely to be noticed through their effects). Human generated patterns are those that occur at the user interface level (such as the command line or GUI), rather than machine-generated events such as network packet traces, system calls, or scripted attack sequences.

The subset of anomaly detection described above encompasses 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. Host-based anomaly detection at the user level can be viewed as a continuous monitoring process of the *internal* state of the system, working in conjunction with *perimeter* defenses such as passwords, firewalls, and network-based intrusion detection systems [Heberlein et al. 1990]. Additionally, multi-sensor intrusion detection systems such as AAFID, [Balasubramaniyan et al. 1998], or EMERALD, [Porras and Neumann 1997], allow integration of a user monitor, such as the one presented here, with other knowledge sources to yield a comprehensive and robust view of the system state.

We take a machine learning viewpoint of the anomaly-detection problem, in which the task is to train a classifier with known "normal" data to distinguish normal from anomalous behaviors. We focus on an *instance-based learning* (IBL) model in which query data is classified according to its relation to a set of previously encountered exemplar instances. The system we present stores historical examples of user behavior to reference when assessing the normalcy of newly encountered behavioral data.

In the rest of this paper, we examine methods for representing the anomaly detection domain as an instance-based learning task, including a temporal encoding of discrete data streams and a definition of similarity suitable for discrete temporal sequence data. We present two classes of methods for data reduction in this domain: one based on instance selection through accumulated activity statistics and one based on instance clustering. We finish with an empirical examination of performance at differentiating users under this learning scheme.

2. LEARNING FROM TEMPORAL SEQUENCE DATA

To approach anomaly detection as a machine learning task, we must define both the learning model and representational format for the input data. One popular and highly general class of machine learning techniques is *instance based learning* (IBL) [Aha et al. 1991]. In this model, the concept of interest is implicitly represented by a set of instances that exemplify the concept (the instance *dictionary*). A previously unseen, query 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. In continuous domains, for example, the similarity measure 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 parametric statistical models [Fukunaga 1990], artificial neural networks [Ripley 1996], or decision trees [Quinlan 1993]. To adapt the anomaly detection task to the IBL learning framework, we need to choose a fixed-length vector (feature vector) representation of the data and to define the concept of "closeness", or similarity, of two vectors.

An important difference between our task and the traditional classification task of IBL is that we do not have labeled instances of multiple classes. We clearly have instances of the valid user's normal working behaviors, but instances of hostile behaviors are a different matter. Leaving aside the practical difficulty of obtaining instances of hostile activity[1], there is an issue of coverage. Because the space of possible malicious behaviors and intruder actions is potentially infinite, it would be difficult or impossible to demonstrate complete coverage of the space from a finite training corpus. Furthermore, it is often the previously unseen attack that represents the greatest threat (indeed, the very purpose of this work is to augment systems which use pattern databases to detect known threats). Finally, it is desirable, for privacy reasons, that a user based anomaly detection agent only employ data that originate with the profiled user or that are publicly available. Releasing traces of one's own normal behaviors, even for the purpose of assisting the training of someone else's anomaly detector, runs the risk that the data will be abused for the purpose of subverting the original user's security mechanisms. Thus, we are faced with a learning situation in which only positive instances are available (where we assign positive to the class "normal behaviors" and negative to the class "anomalous behaviors"). When only positive examples are available, many standard IBL algorithms (such as the k-nearest-neighbor rule, for example) would automatically classify all new examples as positive. In Section 2.2.5, we discuss a classification rule that employs only positive examples. Learning from only positive examples presents a challenge for classification as it can easily lead to overgeneralization [Iba 1979].

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 such as anomaly detection, the size of the instance dictionary can conceivably grow without bound, requiring 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 instance selection and clustering algorithms to reduce dictionary size. Instance selection methods use accumulated usage statistics, such as the number of times a dictionary instance was most similar to a query instance, to choose items to remove from the dictionary. In the clustering formulation, a group of similar dictionary instances is replaced with a single exemplar instance without regard to usage statistics.

2.1 Alternate Approaches to Sequence Learning

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 [Oppenheim and Schafer 1989], principle component analysis [Fukunaga 1990], linear regression [Casella and Berger 1990], linear predictive coding [Rabiner and Juang 1993], nearest neighbor matching, (γ, ϵ)-similarity [Bollobás et al. 1997; Das et al. 1997], and neural networks [Chenoweth and Obradovic 1996] have proven fruitful. Such techniques typically employ a Euclidean distance or a related distance measure defined for

[1]Examples (usually simulated) of machine-level attack logs (such as network packet logs or system call traces) are available, but traces of *real attacks* at the *human command* level are considerably rarer. A recent call for examples of such data by the CERIAS security research center has, to date, yielded no instances of such data. Currently such data are often not stored, not because of the non-existence of such threats, but because of the lack of adequate automated analysis tools.

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 [Quinlan 1993] 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, [Norton 1994], and Salzberg, [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.

There also exist learning methods explicitly developed to model sequence data. Methods for learning the structure of deterministic finite-state automata, for example, have been widely studied [Angulin 1987; Rivest and Schapire 1989; Aslam and Rivest 1990]. DFA's, however, are not well suited to modeling highly noisy domains such as human-generated computer interface data. There exist stochastic extensions to finite-state automata such as hidden Markov models, [Rabiner 1989], which are often more effective in noisy domains. We are currently investigating the application of such models to this domain.

2.2 The IBL Representation of Anomaly Detection

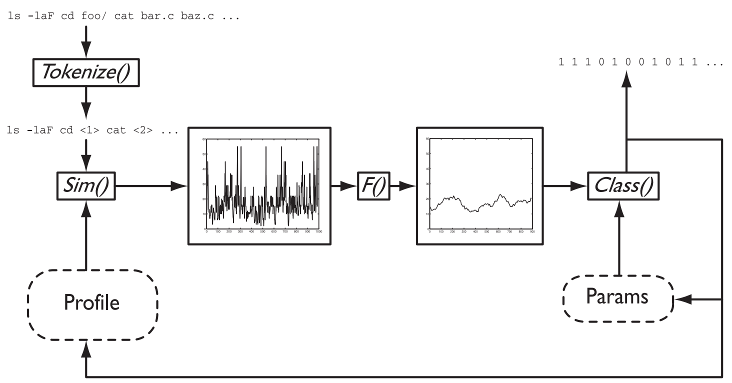


Fig. 1. Information flow in the instance-based anomaly detection system

We have developed a prototype anomaly detection system which employs an instance-based learning framework to classify observed user behaviors as normal or anomalous. We'll give a brief overview of the system's flow of information (shown schematically in Figure 1) here and describe the individual blocks in more detail below. Data enters the system, in the upper left, as an undifferentiated sequence of discrete symbols (UNIX shell command lines in this work) and is passed through a parser (`Tokenize()`) which reduces the data stream to an internal format and does preliminary feature selection. The resulting data stream is compared to the user's historical profile via a similarity measure (`Sim()`), yielding a temporal sequence of real valued similarity measures (indicating instantaneous similarity of observed data to the profile). Because the instantaneous similarity measure stream is highly noisy, classification based on this signal is difficult. To solve this problem, we introduce a noise-suppression filter (`F()`). Classification of the smoothed data stream is via a threshold decision module (`Class()`) whose decision boundaries are set through examination of an independent, parameter-selection set of the user's historical data. The final, binary class stream (upper right) is the detector's estimation of the current state of the input data (1 being "normal" and 0 "abnormal").

The components described above suffice to operate the detection system in a batch mode. In this mode, the detector accumulates a single, fixed profile and employs it for all further classifications. In an operational setting we face the additional difficulty that human behaviors are dynamic and what is considered "normal" behavior is likely to change over time. This problem is known in the machine learning community as *concept drift* and requires a system which can dynamically update the user profile and classification parameters. In such an online learning mode, the detector employs the feedback loop shown in Figure 1 to update the profile and classification thresholds. In the work described in this paper, we are interested in examining the effects of profile data reduction techniques in isolation. To prevent interactions with the additional complexities of the online mode feedback loop, we perform all experiments in batch mode. Elsewhere, [Lane and Brodley 1998], we have examined some of the online learning issues involved with adapting user models to changing behaviors.

2.2.1 *Feature Extraction.* In our environment we have been examining UNIX shell command data, captured via the `(t)csh` history file mechanism. The history data are parsed by a recognizer for the `(t)csh` command language, and are emitted as a sequence of tokens in an internal format. Each "word" of the history data (e.g. a command name, group of command flags, or shell meta-character) is considered to be a single token. The resulting alphabet is very large (over 35,000 distinct symbols in our complete data) and, as the frequency of some of these tokens are quite low, gathering adequate statistics over this large an alphabet is difficult. We have investigated different methods of reducing the alphabet size and have found that omitting file names in favor of a filename count (e.g. `cat foo.c bar.c gux.c` is converted to `cat <3>`) greatly constrains the alphabet size (to just over 2,500 distinct tokens) and in an empirical evaluation, yielded improved anomaly detection performance.

2.2.2 *The Similarity Measure.* We have examined several measures for computing the similarity between two discrete-valued temporal instances, [Lane and Brodley 1997b]. We describe here the measure that we have found to perform best on

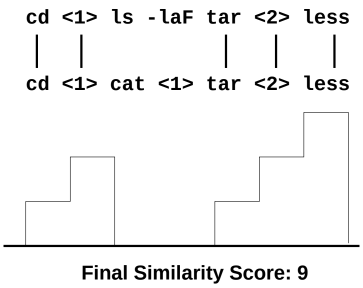


Fig. 2. Example of sequence similarity calculation. Two sequences are compared, element by element. The bottom curve represents the weight contributed by each match, and the final similarity is the area under this curve.

average in empirical evaluations.

The similarity measure operates on token sequences of equal, fixed length. Although we examine only UNIX shell command data in this work, tokens may in general be any symbols drawn from a discrete, finite, unordered alphabet (e.g. GUI events, 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:

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

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

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

The converse measure, *distance*, is defined to be:

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

where $\mathrm{Sim}_{\max}$ is the maximum attainable similarity value for a given sequence length: $\mathrm{Sim}_{\max} = \mathrm{Sim}(X, X)$.

An example similarity calculation is depicted in Figure 2. The function $w(X, Y, i)$ accumulates weight linearly along matching subsequences (bottom curve), and $\mathrm{Sim}(X, Y)$ is the integral of total weight over time (area under the weight curve). In the limiting case of identical sequences, this measure reduces to $\mathrm{Sim}_{\max} = \sum_{i=1}^{l} i = \frac{l(l+1)}{2}$. Thus, a run of contiguous matching tokens will accumulate a large similarity, while changing a single token, especially 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 [Lane and Brodley 1997a] where the best value was found to be dependent on the profile and the opponent being detected. Plausible values for l are small integers in the range 8..15 and the setting $l = 10$ was found to be an acceptable compromise across users.

A user profile is a collection of sequences, **D**, selected from a user's observed actions.[2] 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 [Fukunaga 1990], although we are not actually performing classification at this stage but, rather, are defining similarity to known patterns. We have 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. An average across the entire profile decreases the ability of the classifier to resolve fine-structure patterns in the classification space.

2.2.3 *Segmenting the Event Stream.* Because the similarity measure is defined only for fixed length sequences, it is necessary to partition the raw event stream into component subsequences. This raises the question of optimal sequence alignments: where should each sequence be defined to start? Our approach is based on the data reduction techniques presented in Section 4. Initially, the system 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. For example, under sequence length $l = 6$, the tokenized data stream "`cat <3> > <1> ls -l | more`" would be converted to the three instance sequences "`cat <3> > <1> ls -l`", "`<3> > <1> ls -l |`", and "`> <1> ls -l | more`". After data reduction via instance selection or clustering (see below), the sequences remaining in the profile are considered to define the desired alignments.

2.2.4 *Noise-Suppression.* In practice, we have found that the instantaneous similarity stream, produced by comparing an input data stream to a user profile, is far too noisy for effective classification. 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 spuriously low similarity spike within an overall high similarity period. We therefore employ a noise reduction filter before selecting decision thresholds or performing classification. We employ a trailing window mean value filter defined as:

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

where $\mathrm{Sim}_{\mathbf{D}}(i)$ 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 $\mathbf{D}$. In a comparison of the mean-value filter with a median-value filter, we found that, while the median filter is generally more effective at short window lengths

[2] 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.

($W < 80$), performance for the two methods is approximately equivalent at longer window lengths, [Lane and Brodley 1997b]. We use the mean-value filter here because we are employing $W = 100$ in this work and the mean filter can more easily be made to run quickly. We note that 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 [Kumar and Spafford 1994]. We are primarily concerned here with the class of long-term, low-profile attacks such as resource theft or industrial data theft.

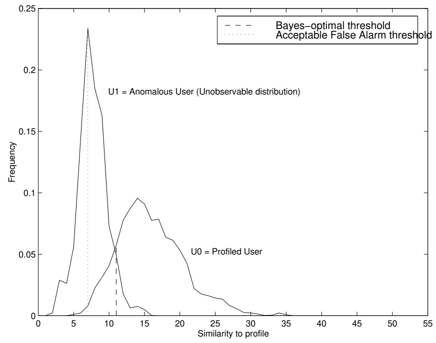


Fig. 3. Comparison of unweighted Bayes-optimal decision boundary and acceptable false alarm rate boundary. The rightmost curve (user U0) represents the profiled user.

2.2.5 *Classification and Threshold Selection.* The similarity-to-profile measure transforms the nominal vector (vectors of l discrete, unordered elements) representation into a real-valued time series on which numerical classification can be performed. Assessing the similarities of all points in a new command trace to the user profile yields a probability distribution, $P_{\{}\mathbf{T}\}$ over similarity values. Because of the structure of the similarity measure (Section 2.2.2), the range of similarity values (and thus $P_{\{}\mathbf{T}\}$) is $0 \ldots \frac{l(l+1)}{2}$. When multiple classes are observable their probability distributions can be used to construct Bayes-optimal decision boundaries, [Fukunaga 1990]. In the Anomaly Detection domain, however, we possess data only from the profiled user so 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 Figure 3, normal (U0) and anomalous (U1) 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 U0 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 U1 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 the above, we must seek another method for selecting a decision boundary. Conveniently, the constraints of our domain provide us with a practical heuristic: constrain the false alarm rate. This yields a Neyman-Pearson hypothesis test, [Casella and Berger 1990], for a non-parametric distribution of the form:

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

where v is the similarity of a sequence to be classified to a particular profile, D, 1 denotes "normal", and 0 denotes "anomalous", and r is the specified false alarm rate. The test, as given above, does not uniquely determine decision thresholds. We implement this test by selecting two decision thresholds, $t_{\max}$ and $t_{\min}$, based on the upper and lower $r/2$ quantiles of the similarity distribution observed on a "parameter selection" data set which is independent of both training and testing data. Similarity values falling between the decision thresholds are labeled normal and those falling outside are labeled abnormal. The lower threshold, $t_{\min}$, detects sequences which are too different from known behaviors, while the upper threshold, $t_{\max}$, detects sequences which are improbably *similar* to historical behaviors, perhaps indicating a replay attack.

The parameter r selects the width of the acceptance region on the similarity-to-profile axis. Thus, it encodes the tradeoff between false alarm and false accept error rates. A smaller value of r yields a wider acceptance region and corresponds to a lower false alarm rate. Simultaneously, however, anomalous values have a greater chance of falling into the broader acceptance region, so false accept rates *rise*. The choice of a particular point on the error rate tradeoff curve will depend on site-specific factors such as security policy and estimated costs of errors.

3. EMPIRICAL ANALYSIS: BASE SYSTEM

Before presenting methods for reducing the amount of data that must be stored in the user profile, we give an empirical performance analysis of the base classification system as described in Section 2.2. In this mode, the profile contains all available training data.

3.1 Performance Criteria

We employ two methods for evaluating the performance of anomaly detection systems. In addition to the traditional *accuracy* measurements, we argue that the *mean time to generation of an alarm* is a useful quantity to consider.

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. The converse accuracy rates are referred to as the true accept (ability to correctly accept the profiled user as normal) rate and the true detect (ability to correctly detect an anomalous user) rate. 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.

Detection accuracy does not, however, reveal the complete story. A second issue of importance is *time to alarm* (TTA), which is a measure of how quickly an anomalous or hostile situation can be detected. In the case of false alarms, the time to alarm represents the expected time until a false alarm occurs. Thus, we wish the time to alarm 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. We define TTA to be the mean run length of "normal" (i.e. non-alarm) classifications. This value represents the mean number of commands over which no alarm will be generated for a given user. For the valid user, this gives a sense of how long work can progress before being interrupted by a false alarm, while for an opponent this measures how long malicious use can continue before being spotted.

3.2 Data Sources and Collection

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 comprises 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 two years. 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 when each user entered and left the study. Because of computational constraints and for testing uniformity, we employed a subset of 7,000 tokens from each user, representing approximately three months of computer usage.

3.3 Experiment Structure

Because user behaviors change over time, the effective lifetime of a *static* user profile, as is employed in the work described here, is limited. Thus, we have constructed experiments to evaluate the detector's performance over a limited range of future activities. The separation of the 7,000 token training data into three groups (or

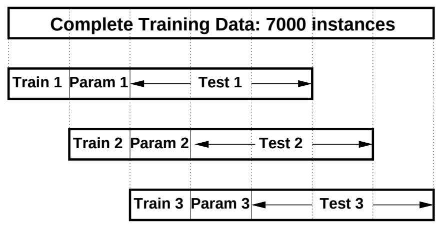


Fig. 4. Division of training data into train, parameter selection, and test data sets.

folds) of train, parameter selection, and test data is shown in Figure 4. The initial 1,000 tokens of each user's data were taken as training (profile construction) data, the following 1,000 tokens were taken for parameter selection data (used to set the decision thresholds $t_{\max}$ and $t_{\min}$), and 3,000 following tokens were taken to test performance for that profile. To guard against isolated data anomalies,[3] three folds of train, parameter selection, and test data were produced for each user. All tests were repeated for each fold.

From each test set, a profile was constructed with $l = 10$ (the fixed sequence length for the similarity measure) and $W = 100$ (the window length for the noise-suppression filter). The resulting profile was tested against the corresponding test set for each user (a total of 8^2 test pairings). A "self" test pairing — testing the profiled user's data against his or her own profile — allows us to examine false alarm rates while a "non-self" pairing allows us to examine false accept rates.

The acceptable false alarm rate, r, determines how the classification thresholds, $t_{\max}$ and $t_{\min}$, are set and has a substantial impact on the tradeoff between false alarm and false accept errors. Because the notion of "acceptable" false alarm rate is a site-dependent parameter, we wish to characterize the performance of the system across a spectrum of rates. We took $r \in \{0.5, 1, 2, 5, 10\}\%$ which yields a performance curve for each profile/test set pair. This curve, which expresses the tradeoff between false alarm and false accept errors with respect to r, is known as a Receiver Operating Characteristic (ROC) curve, [Provost and Fawcett 1998]. An ROC curve allows the user to evaluate the performance of a system under different operating conditions or to select the optimal operating point for a given cost tradeoff in classification errors.

It should be noted that this experimental structure *assumes* that the training data is "pure" — i.e. that no instances of anomalous or hostile behaviors appear in the training data. In practice, the issue of initializing the system with known behaviors is critical and the assumption of data purity may not hold. Although we do not attempt to address this issue, it has been examined by other researchers. HAYSTACK, for example, initializes users as members of hand-crafted behavioral classes [Smaha 1988].

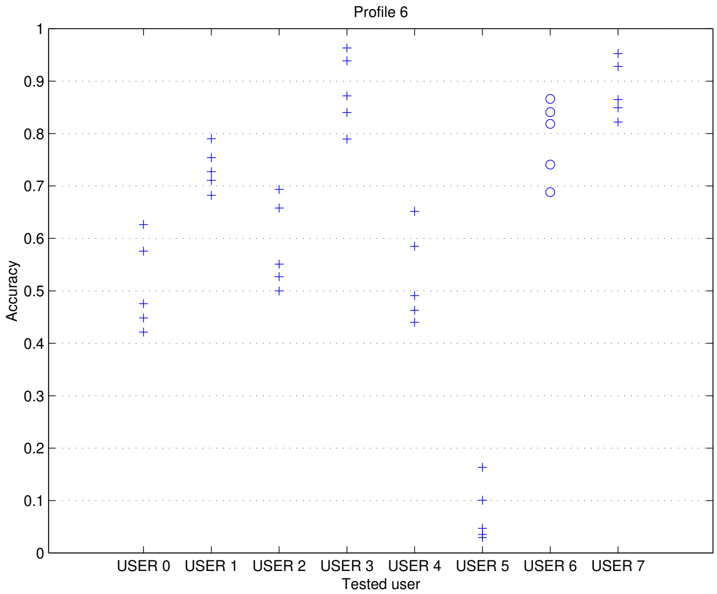


Fig. 5. Accuracy results for base system, single fold, profile for USER6. Each column shows accuracy for one test set against profile 6. "+" symbols denote adversaries, while "o" symbols denote USER6.

3.4 Accuracy Results

An example of accuracy results for a test fold of a single profile (that of USER6) is shown in Figure 5. Each column in this plot displays the accuracy results for a single test set when tested against the profile. When the test set originates with the profiled user (i.e. USER6 tested against Profile 6), the results indicate the ability to correctly identify the valid user (true accept rate). This condition is denoted with an "o" symbol on the plot. When the test set originates with a different user (e.g. USER3 tested against Profile 6), the results indicate the ability to correctly flag an anomalous condition (true detect rate). This condition is denoted with a "+" symbol on the plot. In both cases, accuracy is increasing in the positive direction on the Y axis. The spectrum of results in each column is generated by testing at different values of r, the acceptable false accept rate, as described in Sections 2.2.5 and 3.3. Because r encodes the size of the acceptance region, it yields a tradeoff in detect versus accept accuracies. The smallest value of r tested ($r = 0.5\%$) yields the widest acceptance region and corresponds to the highest (most accurate) point on the true accept column (USER6). But because the acceptance region is wide, more anomalous points fall into it and are accepted falsely. Thus, this value of r corresponds to the *lowest* accuracy in each of the true detect columns (USER{0-5,7}).

Profile 6 was selected to highlight a number of points. First is that accuracy is

[3]E.g. we have found that our users tend to experience large behavioral changes at the beginning of academic semesters. The batch mode detection system presented here is highly sensitive to such changes.

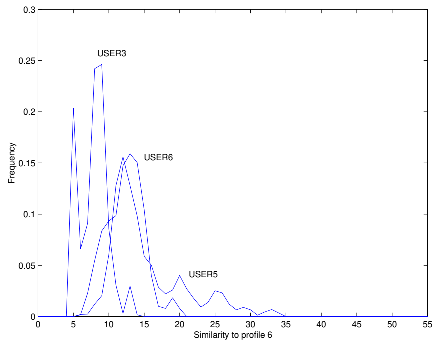


Fig. 6. False accept errors: USER5's data bears high resemblance to the profiled user's (USER6).

highly sensitive to the particular opponent. USER1 and USER3, for example, display quite different detection accuracies. Because of this variance, simple statistics such as mean detection accuracy are insufficient to evaluate the system's performance. Second, although the acceptable false alarm rate parameter, r, was tested across the range 0.5%–10%, all of the observed false alarm rates are greater than this (13.4%–21.2%). This is a result of the training and parameterization data failing to fully reflect the behavioral distribution present in the testing data. Because the user has changed behaviors or tasks over the interval between the generation of training and testing data, the profile does not include all of the behaviors present in the test data. This phenomenon is actually exacerbated by the batch-mode experimental setup used here. In tests of the online version of this system, [Lane and Brodley 1998], we have found that continuously adapting to the user's behaviors (thus shortening the delay between training and testing) improves true accept accuracy.

A third source of false accept error is demonstrated in Figure 6. Here, the profiled user (USER6) and USER5 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, USER3 was engaged mainly in programming and writing during this time. There are two possible sources for the degree of overlap between USER5 and USER6. 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.[4] 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 [Cormen et al. 1992] and hidden Markov models [Rabiner 1989].

[4] A number of such data sources are described in [Denning 1987; Lunt and Jagannathan 1988; Smaha 1988; Heberlein et al. 1990].

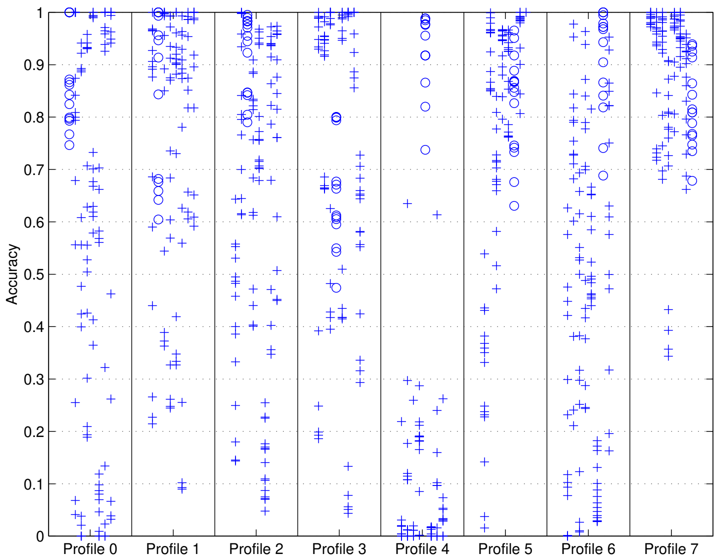


Fig. 7. Base system accuracies for all users and folds. Each column now displays a single profile tested against all test sets (the equivalent of Figure 5). Again, "+" symbols denote adversary accuracies, while "o" symbols denote valid user accuracies.

The complete set of all accuracies for all folds and all profiles is displayed in Figure 7. This plot is intended not as a reference for individual accuracy values, but to convey a sense of the general performance of the system under different operating conditions and to highlight some behavioral characteristics of the detection system. In this figure, each column displays the same data as shown in Figure 5 — all tests against a single profiled user. Now, however, all three folds are given for each profile.

The overall impression in this plot is that many of the accuracy points are clustered toward the top of the plot, indicating that accuracy performance is generally high. The notable exception is Profile 4, which has high accuracy only for true accept (USER4 tested against Profile 4). This is an example of the decision thresholds being set to artificially extreme values, resulting in a spuriously large acceptance region. Thus, the system has effectively decided that "everything is USER4", and no real differentiation is being done — it is simply accepting most behaviors as normal. Examination of USER4's training data reveals that this user appears to devote entire shell sessions to single tasks (such as the compile-debug cycle) which appear as rather repetitious and monotonous patterns. Because this user is working in the X-Windows environment, tasks can be assigned to single shell sessions, and those shell sessions may be long-lived (some were over 2,000 commands). Thus, the training data may display only one or two sessions and a very small number of behaviors, while the parameter selection data displays a different (but also small) set of behaviors. Because there may be little overlap between training and parameter selection data, the observed similarity-to-profile frequency distribution will be distorted and the selected decision thresholds are poorly chosen.

A converse behavior occurs with Profile 3. Although not as dramatic as the Profile 4 case, this Profile displays relatively low true accept rates (denoted by the "o" symbols) in comparison to other profiles. This is an example of the system deciding that "nothing is USER3" because the acceptance region has been set too narrowly. As with USER4, this arises because different behaviors are displayed in the training and testing data. In this case, the parameter selection data reflects the training data well, but the test data is different from both of them. As a result, the acceptance range is narrowly focused to high-similarity behaviors, but the behaviors encountered in the testing data are lower similarity.

Both of these cases can be ameliorated by online training. When the system is constantly updating the profile, the chances of missing important behaviors are smaller. In general, however, changing user behaviors present a serious problem, because sudden large changes may appear very similar to the very anomalies that the system is designed to detect. We explore these issues in [Lane and Brodley 1998].

3.5 Time to Alarm Results

We measure all alarm 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.

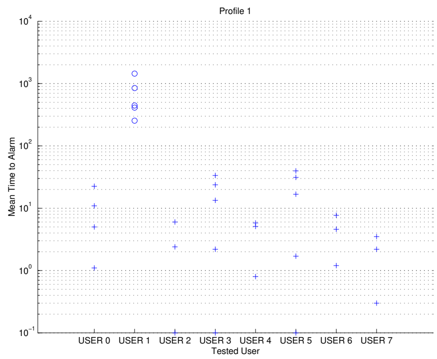


Fig. 8. Time-to-alarm results for the base system, single fold, profile for USER1. Each column shows mean time-to-alarm for one test set against Profile 1. "+" symbols denote adversaries, while "o" symbols denote USER1. Times are in token counts, and the time axis is logarithmic.

An example of time-to-alarm results is shown in Figure 8. Analogously to Figure 5, this plot displays a single fold of tests against a single profile (USER1's profile, in this case). Each point denotes a mean time to generation of an alarm. The circle symbols represent the generation of false alarms, which we desire to occur rarely, while the plus symbols are true alarms which we wish to receive as quickly as possible. Each column shows a spectrum of results, corresponding to the different settings of the acceptable false alarm rate, r (Section 2.2.5). Note that the time axis here is logarithmic, and that the false alarms occur nearly an order of magnitude more slowly than do the true alarms. This difference is disproportionally greater than the accuracy difference between true accept and true detect, indicating that *false alarms occur in clusters separated by long stretches while true alarms occur more sporadically but more often.* This is a desirable behavior, as it leads to rapid detection of adversaries, while false alarms occur rarely and in blocks so that many false alarms can be verified at once.

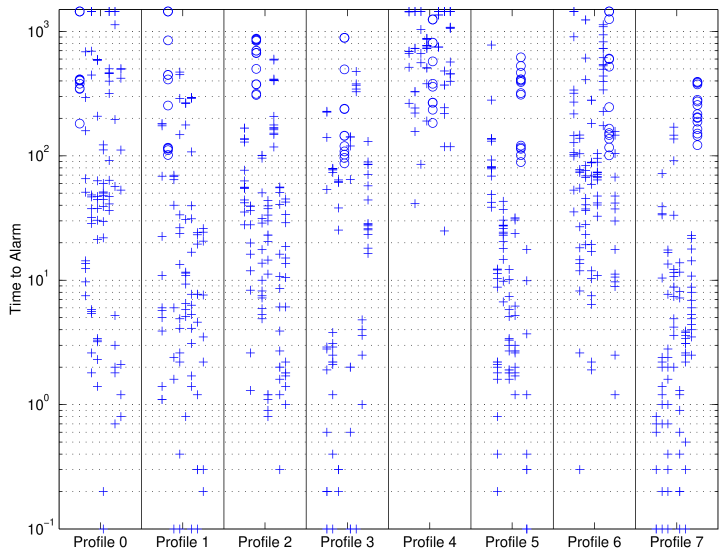


Fig. 9. Base system times-to-alarm for all users and folds. Each column displays a single profile tested against all tests sets (the equivalent of Figure 8). "+" symbols denote adversary times, while "o" symbols denote valid user times.

The complete time-to-alarm results for the base system are shown in Figure 9. Here, each column shows the results of all tests and all folds for a single profiled user. The important result in this plot is that the times to generation of false alarms (represented by the "o" symbols) are generally much longer than the times to true detections (denoted by the "+" symbols) for a given profile. The difference is often better than an order of magnitude, and is sometimes better than two. When the underlying accuracy results are poor, as in the case of USER4's profile, the time-to-alarm results are similarly poor. In this case, since the system is marking most behaviors as normal, true alarms are generated nearly as rarely as false alarms.

4. STORAGE REDUCTION

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, [Wilson and Martinez 1999]. In an operational setting, data reduction is critical as the size of the profile directly impacts the time required for classification. In this section, we describe two classes of techniques that we have investigated for reducing the storage requirements of the user profile and give empirical results demonstrating the performance of these data reduction methods.

4.1 Instance Selection

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 the utility of this analogy, we implemented the least-recently-used (LRU) pruning strategy. As parameter selection instances are examined and classified, the profile instance selected as most similar to the query instance 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) and LFU (remove least frequently used sequences). Finally, we employ uniformly random instance selection as a base-line instance selection method.

We employed the experimental structure described in Section 3.3, with the addition of a final profile size parameter, S, which sets the number of sequences to remain in the user profile after instance selection. S describes the severity of the pruning to be done and can have a substantial impact on the descriptiveness of the profile. Again, the tradeoff between available system resources and desired classification performance is a site-dependent issue, so we examined the performance of each pruning method for $S \in \{100, 250, 500, 750\}$ instances.

4.1.1 *Accuracy Results.* Performance comparisons of the instance selection methods to the base system (described in Sections 3.4) for accuracies are given in Figure 10. Each of these figures plots the accuracy results for one instance selection technique on the vertical axis against those for the base system on the horizontal axis. The diagonal lines are equal-performance lines and in the region to the right of this line the base system has superior performance, while in the region above the diagonal the instance selection technique has superior performance. The points shown represent all folds and tests for all profiles for a single setting of the profile size parameter, S. The results displayed here are for $S = 500$ (i.e. preserve only 500 of the original thousand instances in the profile after instance selection). Other settings of S yield more-or-less extreme versions of the results displayed here, but do not change the fundamental nature of the results. The "o" symbols denote true accept accuracies (rate of correctly recognizing the profiled user) while "+" symbols denote true detection accuracies (rate of correctly distinguishing an imposter).

The first feature of note in these graphs is that, overall, the base system has superior true accept accuracy but inferior true detect accuracy.[5] Accuracy impacts are a natural result of reducing the profile size because as information is removed from the profile, it becomes more likely that new query instances will fail to find a good match in the profile. Potentially, this can affect only true accept accuracy while

[5]Although, visually, it may appear that the distinction is complete, the base system actually has superior true detection accuracy in approximately 16% of the cases. These points all occur at the extreme ends of the accuracy scale, and the margin of victory is slight.

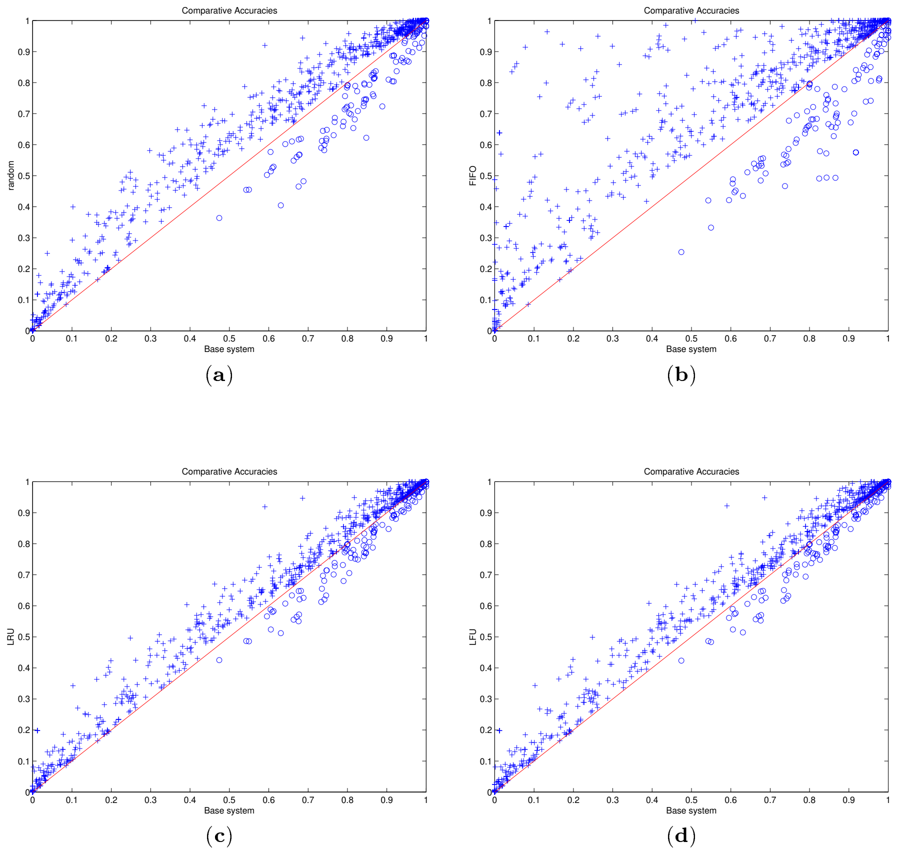


Fig. 10. Comparison of accuracy results for base system versus random (**a**), FIFO (**b**), LRU (**c**), and LFU (**d**) instance selection strategies. Base system accuracies are plotted on the horizontal scale while accuracies of the instance selection techniques are plotted on the vertical scale. Points falling to the right of the diagonal indicate higher accuracies for the base system, while points above the diagonal indicate higher accuracies for the instance selection strategy. The "o" symbols denote true accept accuracies and the "+" symbols are true detect accuracies.

leaving true detect accuracy unchanged. The results from the instance selection techniques, however, demonstrate a different class of behavior. Here, true detect rates are actually increased. In these cases, the range of similarity values accepted as normal is being set too narrowly. This improves true detections because an adversary's behaviors has a smaller chance of being similar enough to be accepted, but decreases true acceptance. The instance selection systems have biased their concepts toward rejecting more classes of behaviors. Effectively, they have decided that "nobody is the profiled user".

The basic process by which accuracy is impacted is the same for all of the instance selection techniques. The important difference between them is in the ability to control the informational loss. By carefully choosing the instances to preserve in the profile, the accuracy impacts can be minimized. Smaller accuracy impacts appear as clustering of the points near the iso-performance line at the diagonal. The worst technique in terms of accuracy impacts is the FIFO method, (**b**), which keeps only the S newest instances in the profile. Clearly, a great deal of important information is being lost by discarding older instances (a result that casts some doubt on the assumption of locality of reference employed by the instance selection techniques discussed here). Simply randomly selecting S instances to keep, (**a**), is superior to FIFO. Both LRU, (**c**), and LFU, (**d**), beat random. The difference between the two is slight, even by non-visual measures. LRU sacrifices an average of 3.6% on true accept rate to gain 3.5% on true detect rate, while LFU loses 3.7% to gain 3.4%.[6] At smaller values of S (final profile size), the effect is more pronounced, but the margins are still not dramatic. At $S = 100$, for example, (reducing the profile to a tenth of its original size), the corresponding figures for LRU are 28.6% and 17.8%, compared to 23.7% and 16.1% for LFU.

4.1.2 *Time to Alarm Results.* Comparative results for the time-to-alarm measure are given in Figure 11. The plots are analogous to those in Figure 10, but give times-to-alarms for the same conditions. Times are in token counts, and both time axes are logarithmic. In the region to the right of the diagonal, the base system has longer time-to-alarm, while above the diagonal the instance selection technique has longer TTA. It is apparent that instance selection decreases (or leaves unchanged) time-to-alarm in all cases. This is a consequence of the accuracy tradeoff observed in the previous section. Improved ability to detect an adversary corresponds to a shorter TTA, while poorer ability to validate the profiled user also yields shorter TTA (i.e. more false alarms per unit time). Again, points close to the iso-performance diagonal correspond to minimal performance impact because of instance selection. Not surprisingly, we observe the same relations among the instance selection techniques that we found in the last section. In particular, FIFO incurs the largest performance changes, while LRU and LFU are approximately equivalent, both beating random instance selection.

[6]To measure the relative accuracy performance between two systems, we employ a mean of accuracy value differences. Thus, the difference in true detect rates between method 1 and method 2 is:

$$\frac{1}{N} \sum_{t \in \mathrm{opponent\_test\_sets}} (\mathrm{accuracy}_{method1}(t) - \mathrm{accuracy}_{method2}(t))$$

where N is the number of opponent test sets.

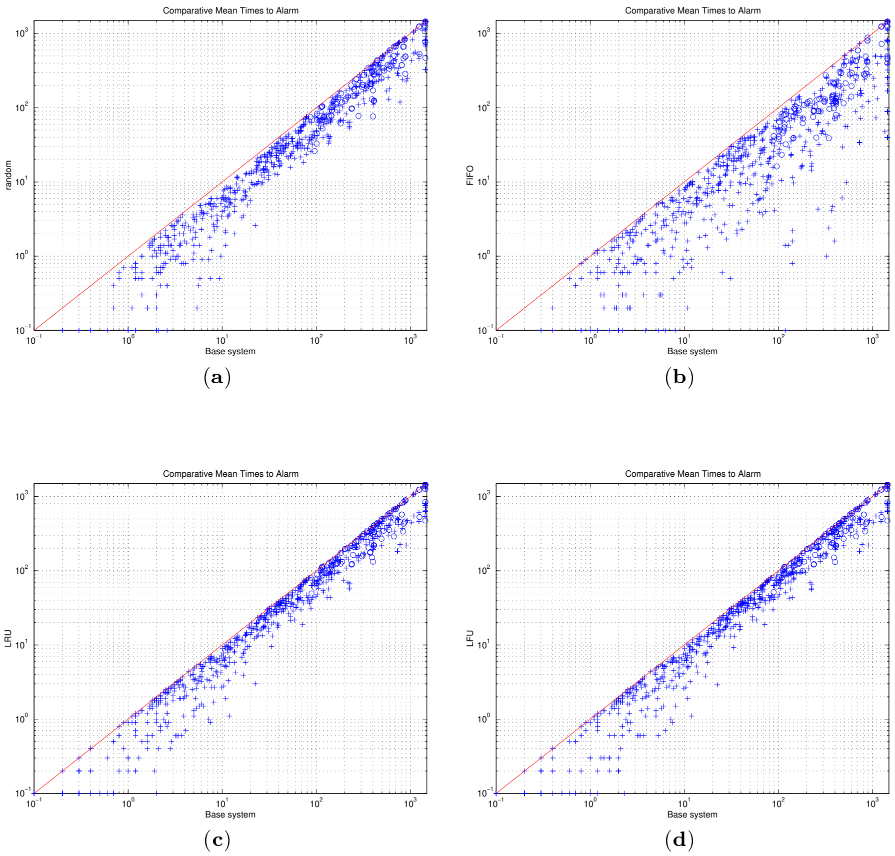


Fig. 11. Comparison of time-to-alarm results for base system versus random (**a**), FIFO (**b**), LRU (**c**), and LFU (**d**) instance selection strategies. Base system TTA's are plotted on the horizontal scale while TTA's of the instance selection techniques are plotted on the vertical scale. Points falling to the right of the diagonal indicate higher TTA's for the base system, while points above the diagonal indicate higher TTA's for the instance selection strategy. The "o" symbols denote time to generation of a false alarm and the "+" symbols are times to generation of true alarms. All times are in token counts, and both time axes are logarithmic.

4.2 Clustering

An alternate method of reducing data storage is to modify the representation of sets of points within the data space. For example, Salzberg, [Salzberg 1991], represented sets of points as hyper-rectangles, while Domingos, [Domingos 1995], induced rules that cover subsets of the instance base. We have examined techniques which attempt to locate *clusters* of similar instances (with respect to the similarity measure defined in Section 2.2.2) within the data. A cluster can be represented by a single exemplar instance, or center point, which is the instance having the smallest distance to all other instances in the cluster. By discarding all other elements of the cluster, substantial space and time savings can be realized. Although the practical effect of this process is the same as that of the instance selection methods described above, the clustering process employs knowledge about the relationships among elements within the profile while the pruning methods employ knowledge about the relationship between individual profile elements and the "external" parameter selection data.

One popular class of clustering algorithms is based on the Expectation-Maximization (EM) procedure, [Moon 1996]. These methods attempt to simultaneously maximize an optimality criterion across all clusters through gradient descent on the cluster likelihood space. The basic process is to assign all points to clusters and evaluate the optimality criterion under that labeling. The evaluation yields a parameter set which is used to reassign points to clusters. This basic loop is repeated until the optimality criterion converges to a stable point. A common implementation of this process is the K-means algorithm, or its discrete analog, K-centers. In these algorithms, clusters are parameterized by their centroid (continuous) or center (discrete) points and radii and the search locates K clusters.

Clustering methods based on EM are popular because they are general and often highly effective. However, when many local optima are present in the likelihood space, the quality of the solution produced can be very sensitive to the initial assignment of points to clusters. A larger difficulty for the anomaly detection domain is that K, the the number of clusters to be sought, must be known *a priori* yet it is not clear how to determine the number of "natural" clusters in the user behavioral data. Furthermore, the convergence rate of these methods is not guaranteed and is often related to the number of clusters sought. For a large K, the search time for a stable solution can be prohibitive.

As a response to these difficulties, we propose a greedy clustering algorithm which builds individual clusters consecutively, attempting to minimize the criterion:

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

for each cluster $\mathbf{C}$. This measure is a generalization of the mean inter-cluster distance employed for clustering [Fukunaga 1990]. 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 value of the cluster's criterion function reaches a local minimum. Because, in some cases, the cluster value monotonically approaches $\mathrm{Sim}_{\max}$, 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, $\mathbf{C}_{\text{cent}}$, 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 $\text{Sim}(X, \mathbf{C}_{\text{cent}})$.

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 [Quinlan 1993]. 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 the final cluster is represented only by its center and mean radius. We realize substantial space savings by discarding all cluster elements other than the center.

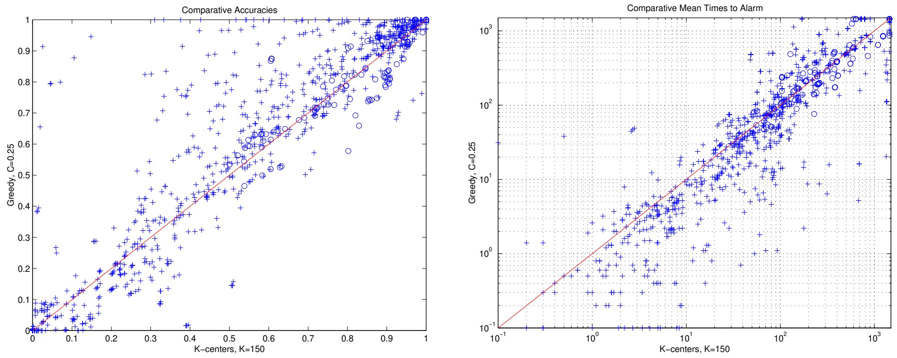
The complete clustering algorithm is structurally similar to the single cluster construction algorithm. We sequentially select individual clusters by their ability to maximize the analog of mean intra-cluster distance:

$$\text{val}\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \text{Dist}(\mathbf{C}_{i,\text{cent}}, \mathbf{C}_{j,\text{cent}})}{n^2}.$$

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. C determines when the clustering process will be halted and how many clusters will be constructed. A small value of C produces many clusters, while a large one allows many of the original profile points to remain. We examine the effects of the parameter C below.

4.2.1 *Comparison of Clustering Techniques.* We examined the performance of the K-centers and greedy clustering methods for storage reduction under the experimental conditions described in Section 3.3. The additional parameters examined were K, the number of clusters to locate for the K-centers method, and C, the global halting criterion for the greedy clustering method. For K-centers, we ran the search to 10,000 cycles or convergence, and examined $K \in \{50, 75, 100, 125, 150\}$ instances. We note that this corresponds to a fairly extreme degree of data compression — retaining only K sequences total in the profile — but we were unable to examine the system behavior for larger K, as the search failed to converge or reach an acceptable solution in the allotted number of search cycles. For the greedy clustering method, we examined $C \in \{0.25, 0.5, 0.75\}$ and used a fixed, empirically selected $\epsilon = 0.005$.

A comparison of the accuracies of the two clustering techniques is given in Figure



(**a**) (**b**)

Fig. 12. Comparisons of accuracy, (**a**), and time-to-alarm, (**b**), for greedy clustering method with $C = 0.25$ (vertical axis) versus K-centers clustering with $K = 150$ (horizontal axis). The diagonal is the equal performance line. Circles denote true accept rate/time to false alarm and pluses denote true detect rate/time to true detection.

12 (**a**). In this figure, the test accuracies of the greedy clustering system with $C = 0.25$ are plotted vertically against the accuracies for the K-centers clustering system with $K = 150$ on the horizontal. The diagonal line is the iso-performance surface. In the region to the right of this line the K-centers system has superior performance, while in the region above the diagonal the greedy-clustering technique has superior performance. The "o" symbols represent true acceptance rates, while the "+" symbols represent true detection rates. The corresponding times-to-alarm are given in (**b**).

The interpretation of these plots is not as clear-cut as are those of the plots in Section 4.1. Because both methods are employing a reduced profile, neither has uniformly superior performance in the way that the base system is uniformly superior to the instance selection methods. Nonetheless, some trends are evident. The greedy clustering method has generally stronger true detection accuracy (at a mean accuracy value difference of 4.5% over K-centers), but slightly poorer true acceptance accuracy (0.5% lower). The important distinction is revealed in the time-to-alarm results. In this measure, the greedy system has superior results in both time to false alarm (an average of 5.3 longer) and time to true detection (an average of 16.3 shorter). Thus, in terms of practical impact, although the greedy clustering system has a marginally lower true acceptance rate, it is generating false alarms more rarely.

In part, the disparity between the two systems can be attributed to the final profile size. The settings $K = 150$ and $C = 0.25$ were chosen for display here because the profiles they produce are most nearly comparable in size. After reduction by clustering, the K-centers profiles contain only 150 instances, while the greedy profiles contain approximately 300 instances of which about 180 were cluster centers and the other 120 were original instances not assigned to any cluster. Reaching complete parity in profile sizes is difficult; increasing K beyond 150 could, in theory, produce profiles of equivalent size to those located by the greedy clustering algorithm, but it was found that the K-centers algorithm did not converge

```
**EOF** **SOF** elm findall finger <1> elm cd <1> ls
cd <1> cat <1> finger <1> cd cd <1> ls
<1> elm nn findall finger <1> elm cd <1> ls
**EOF** **SOF** elm findall finger <1> finger <1> chfn finger
**EOF** **SOF** elm findall finger <1> exit **EOF** **SOF** **EOF**
fg cd elm findall finger <1> talk <2> findall **EOF**
```

Fig. 13. An example cluster produced by the greedy clustering algorithm from User 7's data. Each line is a single subsequence of commands and flags. The first line shown is the center point of the cluster. The symbols `**SOF**` and `**EOF**` denote the start and end of shell sessions, respectively.

to an acceptable solution in the allowed time frame (10,000 search iterations) for $K >> 150$. The search also did not converge for $K = 150$, but the solutions located were generally of much higher performance than those for larger values of K. This occurs because the K-centers algorithm is attempting to optimize all K clusters *simultaneously*, and the combinatorics of assigning points to K clusters makes the search difficult for large K. Decreasing K improves the convergence of the algorithm, but yields a smaller final profile and poorer performance. For the data sets tested here, $K = 150$ was found to be most effective.

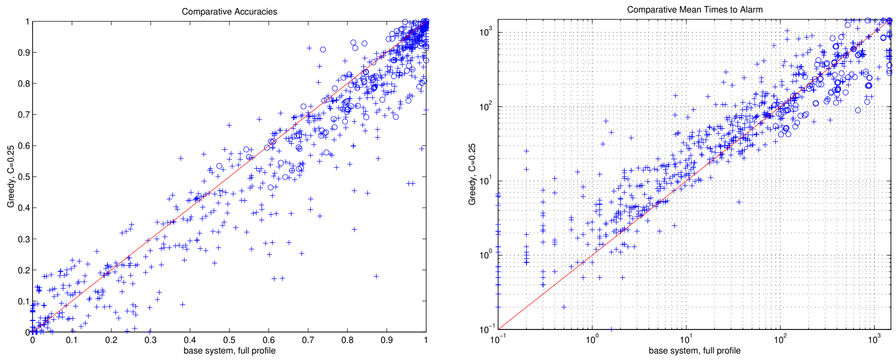
Decreasing C (the parameter controlling the termination of the greedy search for new clusters) would allow the greedy clustering algorithm to search further and locate more clusters, thereby decreasing total profile size. But as $C = 0.25$ already yields about 180 clusters, it would not be possible to reduce the profile to 150 instances solely by decreasing C.

Increasing C, on the other hand, causes premature halting of the clustering process and yields fewer clusters (typically fewer than 5 clusters at $C = 0.75$, for example) and a large number of retained original points (more than 900, or greater than 90% of the original profile, for $C = 0.75$). Doing so does improve accuracy with respect to the K-centers system, but this is to be expected given the much larger profile of the greedy system in this case.

Thus, the best performance of the K-centers system is beaten, in this domain, by a rather conservative setting of the greedy clustering system. We attribute this to the dimensionality of the space — attempting to optimize $K = 150$ clusters simultaneously is combinatorially difficult. While the K-centers algorithm is guaranteed to converge to a locally optimal cluster assignment, there is no guarantee of time bounds on the convergence. In this domain, we have found that the performance of the K-centers based system is bounded largely by the convergence rate. The greedy clustering method does not suffer from the same scaling difficulties, as it is guaranteed to terminate.

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". An example of such an "intuitive" cluster is shown in Figure 13.

4.2.2 *Comparison of Clustering to Base System.* Because the greedy clustering algorithm was shown to have superior performance to the K-centers algorithm for this data, we present only its results here. The relative performance of the greedy



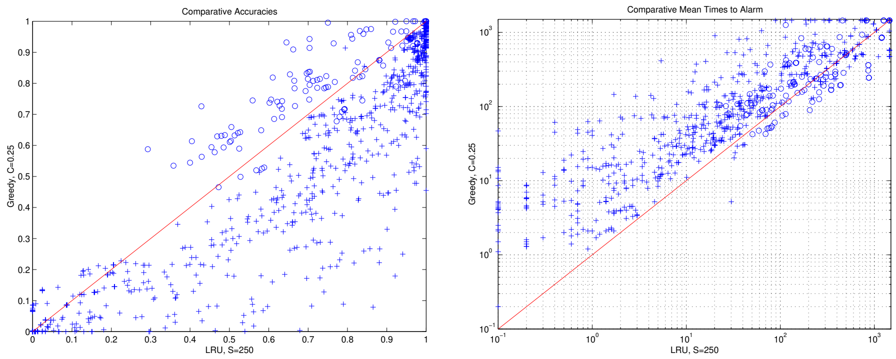
(**a**) (**b**)

Fig. 14. Comparisons of accuracy, (**a**), and time-to-alarm, (**b**), for greedy clustering method with $C = 0.25$ (vertical axis) versus the base system with the full user profile (horizontal axis). The diagonal is the equal performance line. Circles denote true accept rate/time to false alarm and pluses denote true detect rate/time to true detection.

clustering and base systems is displayed in Figure 14. In this figure, the test accuracies of the greedy clustering system with $C = 0.25$ are plotted vertically against the accuracies for the base system (full user profile) on the horizontal. The diagonal line is the iso-performance surface. In the region to the right of this line the base system has superior performance, while in the region above the diagonal the greedy-clustering technique has superior performance. The "o" symbols represent true acceptance rates, while the "+" symbols represent true detection rates. The corresponding times-to-alarm are given in (**b**).

The situation in this comparison is analogous to those of Section 4.1; a reduced user profile (approximately 300 instances for the greedy system with $C = 0.25$) compared to the full user profile (1000 instances) of the base system. Not surprisingly, the base system has better overall performance in both accuracy and time-to-alarm (note that the time to false alarms is longer for the base system while the time to true detection is shorter for the base system). What *is* interesting, is that the greedy clustering technique does not display the same behavior that the instance selection techniques do. Recall that the instance selection systems produced true accept and true detect rates that were well separated by the iso-performance diagonal line. That result is characteristic of a "nobody is the profiled user" problem — enough data had been removed from the profile that the systems were unable to characterize the profiled user well so they marked too many situations as anomalous. The greedy clustering system does not seem to suffer from this fault. It has lower accuracy on both true detect and true accept, indicating that it has lost ground against the full profile, but has done so gracefully. Rather than learning an incorrect generalization ("nobody is the profiled user"), the greedy clustering method has preserved the correct characterization of the user's behavior and has degraded gracefully under the pressure of reduced profile size. The K-centers clustering method displays a similar form of degradation.

4.2.3 *Comparison of Clustering to Instance Selection.* Since the LRU and LFU instance selection techniques have comparable performance and are the most effec-



(**a**) (**b**)

Fig. 15. Comparisons of accuracy, (**a**), and time-to-alarm, (**b**), for greedy clustering method with $C = 0.25$ (vertical axis) versus LRU instance selection with $S = 250$ (horizontal axis). The diagonal is the equal performance line. Circles denote true accept rate/time to false alarm and pluses denote true detect rate/time to true detection.

tive of the instance selection techniques we examined, we compare only LRU to the greedy clustering method. The most nearly comparable settings, in terms of profile size, for the two systems are $C = 0.25$ (yielding approximately 300 instances in the final user profile) and $S = 250$ (or 250 instances retained in the profile after LRU instance selection). Comparisons of the two system are displayed in Figure 15. In this figure, the test accuracies of the greedy clustering system with $C = 0.25$ are plotted vertically against the accuracies for the LRU instance selection system with $S = 250$ on the horizontal. The diagonal line is the iso-performance surface. In the region to the right of this line the LRU system has superior performance, while in the region above the diagonal the greedy-clustering technique has superior performance. The "o" symbols represent true acceptance rates, while the "+" symbols represent true detection rates. The corresponding times-to-alarm are given in (**b**).

The difference between the systems is substantial. The LRU system generally has much higher true detection rates, but much lower true acceptance rates. This difference is reflected in the TTA plot, where we find that LRU has much shorter time to detection of an anomaly (by an average of 61.3 tokens) at the cost of a shorter time to generation of false alarms (38.9 tokens on average). This type of result is reminiscent of those presented in Section 4.1 and, indeed, the cause is similar. The profiles produced by the LRU algorithm often do not preserve important behavior exemplars that are needed to provide high similarity matches to the test data. The result is improperly set decision thresholds that reject a large proportion of test data, yielding high detection rates for opponents, but low acceptance rates for the profiled user. The characteristic separation between accept rate and detect rate points is not as dramatic as those in Section 4.1.1 because the comparison is not against the full profile, but against a profile selected by the greedy clustering algorithm. The existence of a clear separation is an additional indication that the greedy clustering system is performing in the same fashion (albeit at lower overall accuracies) as is the base system. Overall, the greedy algorithm appears to be producing superior profiles to those generated by the LRU selection process in most cases.

4.3 Summary of Data Reduction Techniques

Because user behavioral profiles can, potentially, grow without bound, it is necessary to examine methods for reducing the size of the profile. Profile reduction saves not only space, but time because the running time of the classification operation in an instance-based learning system is related to the number of instances in the instance dictionary. We have examined two classes of techniques for reducing the profile size: instance selection techniques which are based on examining the interactions of profile instances with external parameter selection data; and clustering techniques which are based on examining the relations of instances *within* the profile.

Within the class of instance selection techniques, we found that the LRU and LFU methods were comparable and both soundly beat other contenders. FIFO, the other "intelligent" instance selection technique, performed poorly, losing even to random element selection. All of the instance selection techniques, however, display the same class of performance degradation. They all generalize to an incorrect concept of user behavior in which they tend to classify novel data as anomalous, resulting in an improved ability to recognize impostors at the cost of a decreased ability to accept the profiled user. The major difference between these methods is in the degree to which they succumb to this incorrect generalization.

The clustering methods exhibit a different class of behavior. Rather than learning to incorrectly label too many query instances as anomalous, they degrade in a more uniform manner, losing both true accept and true detect accuracy as the profile is reduced. This is a preferable form of degradation, as it preserves the concept of user behavior rather than replacing with a "nobody is the profiled user" concept. In the limit, the correct concept is likely to scale better, while the "nobody is the profiled user" concept could reduce to marking all behaviors as anomalous.

The greedy clustering algorithm outperforms the K-centers method in this domain. The difference appears to result from K-centers's slow convergence for large K. As K (the number of clusters to locate) grows, K-centers takes longer to converge to a solution. By the time K is as large as the number of clusters located by the greedy method (approximately 180), K-centers does not locate an acceptable cluster solution in an acceptable time (10,000 search iterations).

5. PRACTICAL IMPLEMENTATION ISSUES

Although it is likely that the system presented here does not possess the level of performance necessary for a fielded anomaly detection system, it is nonetheless educational to examine the results given above in terms of practical utility and to consider some ways of rectifying weaknesses in the present system.

5.1 Impersonation Rates

One measure of the practical effectiveness of an anomaly detection system is the chance that an imposter can find a valid user to successfully impersonate. "Successful impersonation" can be interpreted in a number of ways, and we shall examine two of them. First, we may consider that an impersonation is successful if an imposter appears indistinguishable from a valid user. For example, if the impostor displays times-to-alarm (TTA's) that are at least as long as those of a valid user, the imposter may be considered to be successful. A summary of this measure appears in Table 1.

	USER0	USER1	USER2	USER3	USER4	USER5	USER6	USER7
USER0	I	-	-	-	-	-	-	-
USER1	-	I	-	-	-	-	-	-
USER2	-	-	I	-	-	I	-	-
USER3	-	-	-	I	-	-	-	-
USER4	I	I	I	I	I	I	I	I
USER5	I	-	-	-	-	I	-	-
USER6	-	-	-	-	-	-	I	-
USER7	-	-	-	-	-	-	-	I

Table 1. Impersonation incidence for the greedy clustered system at an acceptable false alarm level of 0.005. The valid users are listed on the left and impostors across the top. The character "I" represents an instance of a successful impersonation.

This table displays incidences of successful impersonations for the greedy clustering system at a halting threshold of $C = 0.25$ (see Section 4.2) and an acceptable false alarm rate threshold of 0.005 (see Section 2.2.5). Whenever an impostor's mean time-to-alarm (averaged across all test folds) meets or exceeds that of the valid user, the impersonation is considered successful and is denoted with an "I". The valid user is always considered to be successful at impersonating him- or herself. We see that under this measure, successful impersonations are relatively difficult to achieve, the notable exception being impersonations of USER4. Overall, there are only nine impersonations out of 56 possible, or about a one in six chance. USER4 is particularly bad, as observed in Section 3.4, because of the way in which his behaviors change over time. As we discuss in the next section, online learning methods can probably compensate for most or all of that difficulty. If we disregard USER4, there are only two impersonations out of 49 possible or about a one in 25 chance. If we relax the acceptable false alarm threshold to 0.1 (allowing more false alarms in exchange for fewer false accepts) the figures improve to $6/56 = 10.7\%$ with USER4, and no successful impersonations without USER4.

A more conservative way to estimate impersonation rate, however, is to examine the chance of an imposter going undetected for some fixed period of time. The results of such an analysis appear in Table 2. The three sub-tables display incidence of successful impersonations at TTA cutoffs of 10, 100, and 500 tokens (recall, however, that all TTA's are measured *after* the smoothing window, Section 2.2.4, has been applied so an additional $W = 100$ tokens have been observed in addition to the TTA cutoff value). If an alarm has not, on average, been issued within the specified cutoff, the impersonation is considered to be successful and is denoted with a "I". Note that under this definition, if a false alarm occurs within the cutoff period then the valid user is considered to have been unsuccessful at impersonating him- or herself.

We see from Table 2 that impersonation is relatively easy at short time frames; at a cutoff of 10 tokens, impersonations are successful 50 out of 56 possible times (or 89%). This is, however, not terribly surprising as the explicit goal of the techniques

	USER0	USER1	USER2	USER3	USER4	USER5	USER6	USER7
10 tokens								
USER0	I	I	I	I	I	I	I	I
USER1	I	I	I	I	I	I	I	I
USER2	I	I	I	I	I	I	I	I
USER3	I	I	I	I	I	I	-	I
USER4	I	I	I	I	I	I	I	I
USER5	I	-	I	I	I	I	-	-
USER6	I	I	I	I	I	I	I	I
USER7	-	I	I	I	I	-	I	I
100 tokens								
USER0	I	I	I	I	-	I	I	I
USER1	-	I	-	I	I	I	-	-
USER2	I	-	I	-	-	I	-	-
USER3	-	-	-	I	-	I	-	-
USER4	I	I	I	I	I	I	I	I
USER5	I	-	-	-	-	I	-	-
USER6	I	I	I	I	-	I	I	I
USER7	-	-	-	-	-	-	-	I
500 tokens								
USER0	I	-	I	-	-	I	-	-
USER1	-	I	-	-	-	-	-	-
USER2	-	-	-	-	-	I	-	-
USER3	-	-	-	-	-	-	-	-
USER4	I	I	I	I	-	I	I	I
USER5	-	-	-	-	-	-	-	-
USER6	I	-	I	-	-	-	I	-
USER7	-	-	-	-	-	-	-	-

Table 2. Impersonation incidences for various fixed time frames under the greedy clustering system with a acceptable false alarm rate of 0.005. The valid user appears on the left and the impostors across the top. "-" characters denote successful detections and "I" characters denote successful impersonations.

employed here is to detect long-term attacks rather than hit-and-run scenarios (Section 1). When we extend the TTA cutoff to 100 tokens (or just twice the minimum possible detection time of the smoothing window length), the situation improves. At this level, there are 30 out of 56 successful impersonations or 61%, and by 500 tokens it has dropped to 15 incidents or about 27%. When we relax the acceptable false alarm threshold to 0.1, the corresponding rates are 29% and 11%. As above, the presence of USER4 is an important factor. When we omit USER4 from consideration, the rates are 55% at a cutoff of 100 tokens and a acceptable false alarm rate of 0.005 and only 16% at 500 tokens and a 0.005 acceptable false alarm rate.

5.2 Performance Improvements

Although the system described in this paper has a number of demonstrated weaknesses, there are many possible modifications which would be likely to improve performance substantially. Here we discuss some approaches that could be taken to improving performance with the goal of reaching a fieldable system. We are currently investigating a number of these issues.

Online training. As we have seen in the empirical results, the detection system displays dramatically poor detection ability for some profiles (either ability to detect impostors or ability to correctly identify the valid user, Section 3.4). To some extent, this is a result of actual behavioral similarities between the users (as illustrated, for example, in Figure 6). But a second source of such errors stems from an interaction between the normal behavioral changes of the users and the batch mode training employed in these experiments. USER4, for example, displays a very narrow set of behaviors across the training data but a different set of behaviors in the parameter selection data. Thus, the profile does not reflect observed behaviors well and the classification thresholds are set very permissively, resulting in poor ability to detect impostors. The batch mode training framework is simply not responsive enough to adapt to such behavioral changes. In other work, [Lane and Brodley 1998], we have examined an online training extension to this system in which the user profile and classification thresholds are modified after every observed token. We found that such a framework dramatically improves performance and greatly reduces the impacts of behavioral changes. We did not employ the online training framework in this work because we wished to examine the effects of the various data reduction strategies in isolation, without the possibility of interaction with another factor, but online training techniques should be employed in a fielded application.

Feature set engineering. Although in this work we employ only command line data (suppressing even file names), many other data sources are available and could provide valuable information for discrimination. Features such as file names, extensions, or types; login times and time of day; network activity; working directory; and even GUI events are all potential sources of important user characteristics. Careful analysis and engineering of such data sources is likely to provide greater discrimination between users and reduce the degree of overlap between users that is illustrated in Figure 6.

Alternative similarity measures. In addition to adding more information sources, it may be possible to take better advantage of the information we are already employing. The similarity measure we employ here (see Section 2.2.2) is relatively coarse and captures only one notion of temporal interactions. Using a different measure, or multiple together, may allow finer discrimination within the data framework already available. In other work, [Lane 1999], we have investigated the use of hidden Markov models as user profiles and similarity measures. Other possible measures that we are or will investigate include edit distance, [Cormen et al. 1992], stochastic context-free grammars, [Charniak 1997], or frequent episode analysis, [Srikant and Agrawal 1996].

Meta-learning and hierarchical classifiers. It is likely that no single sensor will possess the breadth of capability to handle the full anomaly detection problem. But multiple sensors can be combined in a meta-learning framework such as voting, boosting [Freund and Schapire 1997], or stacking [Schaffer 1994] to yield an overall more accurate classifier (assuming that the base-level classifiers are at least minimally competent and make statistically independent errors). It is notable that recent intrusion detection frameworks such as AAFID, [Balasubramaniyan et al. 1998], or EMERALD, [Porras and Neumann 1997], take advantage of such possibilities by employing hierarchical classification schemes.

Domain knowledge. The system presented here employs relatively little domain knowledge, assuming only that the data is represented as a stream of nominal events. By including *a priori* domain knowledge available from a security expert (for example, "Pay special attention to activity on critical files such as `/etc/passwd` or the Windows NT registry") performance could be improved at the cost of decreasing generality of the techniques (a tradeoff worth making for a fielded system).

Automatic parameter selection. The system presented here possesses a number of parameters that must be set: sequence length, l, 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 performance can likely be improved. We are currently investigating methods for automatically adapting system parameters to the profiled user.

6. CONCLUSIONS

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 demonstrated two classes of data reduction methods for limiting profile size. We found that instance selection techniques, based on relations between instances in the user profile and external, parameter selection data, were susceptible to improper generalizations of the normalcy concept. Clustering based methods, based on examining relations solely among instances within the profile, were found not to suffer from this misgeneralization. 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.

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 machine learning 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.

ACKNOWLEDGMENTS

We would like to thank Gene Spafford and the members of the Purdue University Center for Education and Research in Information Assurance and Security, the members of the Purdue machine learning lab, and our reviewers for their helpful comments on this work. We would also like to thank our data donors and, especially, USER4 whose data forced us to examine this domain more closely than we might

otherwise have done.

REFERENCES

Aha, D., Kibler, D., and Albert, M. 1991. Instance-based learning algorithms. *Machine Learning 6*, 1, 37–66.

Angulin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation 75*, 87–106.

Aslam, J. A. and Rivest, R. L. 1990. Inferring graphs from walks. In *Proceedings COLT-90* (1990), pp. 359–370.

Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., and Zamboni, D. 1998. An architecture for intrusion detection using autonomous agents. Technical Report COAST TR 98/05 (June), COAST Laboratory, Purdue University, W. Lafayette, IN 47907-1398.

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

Casella, G. and Berger, R. L. 1990. *Statistical Inference*. Brooks/Cole, Pacific Grove, CA.

Charniak, E. 1997. Statistical techniques for natural language parsing. *AI Magazine 18*, 4, 33–43.

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

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1992. *Introduction to algorithms*. The MIT Press, Cambridge, MA.

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

Denning, D. E. 1987. An intrusion-detection model. *IEEE Transactions on Software Engineering 13*, 2, 222–232.

Domingos, P. 1995. Rule induction and instance-based learning: A unified approach. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (Montreal, Canada, 1995), pp. 1226–1232. Morgan Kaufmann.

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

Freund, Y. and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*, 1, 119–139.

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

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

Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., and Wolber, D. 1990. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy* (May 1990), pp. 296–304.

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

Kumar, S. 1995. *Classification and detection of computer intrusions*. Ph. D. thesis, Purdue University, W. Lafayette, IN.

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

Lane, T. 1999. Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users* (1999), pp. 35–44.

LANE, T. AND BRODLEY, C. E. 1997a. An application of machine learning to anomaly detection. In *National Information Systems Security Conference*, Volume 1 (Baltimore, MD., 1997), pp. 366–380.

LANE, T. AND BRODLEY, C. E. 1997b. 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), pp. 43–49.

LANE, T. AND BRODLEY, C. E. 1998. Approaches to online learning and concept drift for user identification in computer security. In *Fourth International Conference on Knowledge Discovery and Data Mining* (1998), pp. 259–263.

LUNT, T. F. AND JAGANNATHAN, R. 1988. A prototype real-time intrusion-detection expert system. In *Proceedings of the IEEE Symposium on Security and Privacy* (1988), pp. 59–66.

MOON, T. K. 1996. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 47–59.

NORTON, S. W. 1994. 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* (Seattle, WA, 1994), pp. 657–663.

OPPENHEIM, A. AND SCHAFER, R. 1989. *Discrete-Time Signal Processing.* Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey.

PORRAS, P. AND NEUMANN, P. 1997. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the Twentieth National Information Systems Security Conference* (1997), pp. 353–365. The National Institute of Standards and Technology and the National Computer Security Center.

PROVOST, F. AND FAWCETT, T. 1998. Robust classification systems for imprecise environments. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (Madison, WI, 1998). AAAI Press.

QUINLAN, J. R. 1993. *C4.5: Programs for machine learning.* Morgan Kaufmann, San Mateo, CA.

RABINER, L. AND JUANG, B. H. 1993. *Fundamentals of Speech Recognition.* Prentice Hall, Englewood Cliffs, New Jersey.

RABINER, L. R. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE 77*, 2 (February).

RIPLEY, B. D. 1996. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK.

RIVEST, R. L. AND SCHAPIRE, R. E. 1989. Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theoretical Computing* (1989), pp. 411–420.

SALZBERG, S. 1991. A nearest hyperrectangular learning method. *Machine Learning 6*, 3, 251–276.

SALZBERG, S. 1995. Locating protein coding regions in human DNA using a decision tree algorithm. *Journal of Computational Biology 2*, 3, 473–485.

SCHAFFER, C. 1994. Cross-validation, stacking, and bi-level methods for stacking: Metamethods for classification learning. In P. CHEESEMAN AND W. OLDFORD Eds., *Selecting Models from Data: Artificial Intelligence and Statistics IV*. New York: Springer-Verlag.

SMAHA, S. E. 1988. Haystack: An intrusion detection system. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference* (1988), pp. 37–44.

SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements,. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)* (Avignon, France, 1996).

STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., LEVITT, K., WEE, C., YIP, R., AND ZERKLE, D. 1996. GrIDS – a graph-based intrusion detection system for large networks. In *Proceedings of The 19th National Information Systems Security Conference* (Oct 1996). The National Institute of Standards and Technology and the National Computer Security Center.

Wilson, D. R. and Martinez, T. R. 1999. Reduction techniques for exemplar-based learning algorithms. *Machine Learning Journal*. To appear.