An Application of Machine Learning to Anomaly Detection

Terran Lane and Carla E. Brodley
School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1287

February 14, 1997

Abstract

The anomaly detection problem has been widely studied in the computer security literature. In this paper we present a machine learning approach to anomaly detection. Our system builds user profiles based on command sequences and compares current input sequences to the profile using a similarity measure. The system must learn to classify current behavior as consistent or anomalous with past behavior using only positive examples of the account's valid user. Our empirical results demonstrate that this is a promising approach to distinguishing the legitamate user from an intruder.

Keywords: Computer security, Anomaly detection, Machine learning.

Email: terran@ecn.purdue.edu
Phone: +1 317 494-0635
Fax: +1 317 494-6440

An Application of Machine Learning to Anomaly Detection

Abstract

The anomaly detection problem has been widely studied in the computer security literature. In this paper we present a machine learning approach to anomaly detection. Our system builds user profiles based on command sequences and compares current input sequences to the profile using a similarity measure. The system must learn to classify current behavior as consistent or anomalous with past behavior using only positive examples of the account's valid user. Our empirical results demonstrate that this is a promising approach to distinguishing the legitamate user from an intruder.

Keywords: Computer security, Anomaly detection, Machine learning.

1 Introduction

A long-standing problem in the field of computer security is that of *intrusion detection*. The goal is to automatically detect violations of security policy for a computer site by an outsider. Of the many possible approaches to intrusion detection, one that has recieved considerable attention is *anomaly detection* [Anderson, 1980, Lunt, 1990, Heberlein, Dias, Levitt, Mukherjee, Wood & Wolber, 1990]. According to Kumar (1995) "Anomaly detection attempts to quantify the usual or acceptable behavior and flags other irregular behavior as potentially intrusive." Under this definition, the scope of anomaly detection encompasses not only violations by an outsider but also anomalies arising from violations on the part of an authorized user. It is important to note that anomaly detection omits the class of security policy violations which occur within the bounds of normal behavior for a system or site. Detecting anomalous behavior can be viewed as a binary valued classification problem in which measurements of system activity such as system log files, resource usage, command traces, and audit trails are used to produce a classification of the state of the system as normal or abnormal.

In this paper we present a machine learning approach to anomaly detection designed to handle these two problems. Following [Denning, 1987], our system learns a *user profile* and subsequently employs it to detect anomalous behavior. Based on sequences of actions (UNIX commands) of the current user's input stream, the system classifies current behavior as consistent or anomalous with past behavior.

Traditionally, in computer security, user profiles have been built based on characteristics such as resources consumed, typing rate, login location, and counts of particular commands employed [Denning, 1987, Smaha, 1988, Lunt, 1990, Frank, 1994]. These approaches do not use the observation that human/computer interaction is essentially a *causal process*. Typically, a user has a goal to achieve when using the computer, which causes the person to issue certain commands, causing the computer to act in a certain manner. The computer's response, in turn, keys further actions on the part of the human.

To form a user profile our approach learns *characteristic sequences* of actions generated by users. The underlying hypothesis is that a user responds in a similar manner to similar situations, leading to repeated sequences of actions. Indeed, the existence of command alias mechanisms in many UNIX command interpreters supports the idea that users tend to perform many repeated sets of actions, and that *these sequences differ on a per-user basis*. It is the differences in characteristic sequences that we attempt to use to differentiate a valid user from an intruder masquerading as that user. Note that the detection of anomalous behavior is made more difficult because a malicious intruder may attempt to emulate the valid user's behavior, including alias and command usage.

In the remainder of this paper, we discuss our approach to anomaly detection and examine the malicious intruder problem. Our empirical results demonstrate that an approach based on profiling a user through characteristic sequences of commands yields high detection accuracy when not considering malicious users.

2 Learning a User Profile

In order for the detection system to recognize anomalous behavior, it must first form a user profile to characterize normal behavior. In this section we describe the model underlying our approach to user profiling, and discuss implementation details of how profiles are formed from command data.

2.1 Collecting Training Data to Form a User Profile

To learn characteristic patterns of actions, our system uses the *sequence* (an ordered, fixed-length set of temporally adjacent actions) as the fundamental unit of comparison. For this research, actions were taken to be UNIX shell commands with their arguments, although the approach developed here is general and can be extended to any stream of discrete events such as operating system calls or graphical user interface events. Forrest et al. (1996) have developed a similar, sequence based, model by employing analogies to human immunology. Their work focuses on monitoring of system call sequences generated by privileged system processes (daemons) using exact matching to known behaviors.

For ease of data collection, the temporal order of commands was maintained only within the context of a single command interpreter (a shell). Currently, we preserve command names and argument switches but omit the specific file names associated with each command execution. This decision was based on the intuition that the significant facet of the user's command history for this work was *behavior* rather than *content*. Thus, it should be more useful to note that the user invoked the command `emacs` (a text editor) with the behavioral switch `-nw` (run in text-mode rather than initialize the X-windows interface) and two file names, than it would be to take note of the actual file names used. Clearly, for some applications of misuse detection, important information could be extracted from the filenames (for example, directories in which the user typically works).

We envision our anomaly detection system as a personal software assistant (an agent) that helps monitor a user's account for penetrations. Because of privacy issues, and the fact that it is impossible to characterize the full space of user behaviors, only positive examples of the account owner's behavior are available for training.

In traditional classification tasks one has access to both positive and negative examples of the target concept for training. Here we must characterize user behavior purely from positive examples. The simplest classifier possible when presented with only positive examples is the classifier that labels *all* inputs as positive (normal, in this case). This is obviously inadequate, but the space of alternative possible classifiers is large. To resolve this difficulty we invoked the *closed world* assumption — that anything not seen in the historical data represents a different user. Intuitively, it seems likely that this is a reasonable assumption — the very terms anomaly, abnormal, and unusual imply that divergence from past behavior is an important indication of trouble.

2.2 The Data Collection System

To collect user action data, we created a parser for the UNIX csh family of languages (including tcsh) which translates the raw data stream of the shell command trace into a token stream suitable for storage and comparison. This translation suppresses filenames, as described above, but preserves command names, argument switches[1], and other syntactically important symbols such as |, ;, and >&!. For example, the command stream:

```
> ls -laF
> cd /tmp
> gunzip -c foo.tar.gz | (cd \  ; tar xf -)
```

would be translated by the parser into the token stream:

```
ls -laF cd <1> gunzip -c <1> | ( cd <1> ; tar - <1> )
```

where the token <1> denotes the occurrence of a single filename argument[2]. The parser also introduces the tokens **SOF** and **EOF** indicating start and end of a command interpreter session, respectively.

During training, the processed token stream is stored verbatim in the *dictionary*. The dictionary is an instance (sequence) database that, together with a similarity measure and a set of system parameters (described below), constitutes a user's profile.

3 Detecting Anamolous Behavior

Once a user profile is formed, the basic action of the detection system is to compare incoming input sequences to the historical data and form an opinion as to whether or not they both represent the same user. The fundamental unit of comparison in the anomaly detector system is the command sequence. To classify sequences of new actions as consistent or inconsistent with sequence history, all input token streams are segmented into overlapping sequences of tokens (where the *length* of each sequence is a parameter to the system, but is fixed for a single run). Two fixed-length sequences can then be compared using a similarity measure.

[1] Strictly speaking, the parser depends upon the UNIX convention that argument switches are prefixed with a dash, so the `-laF' switch in the command ls -laF ${HOME} would be correctly recognized, but the switch tvf in the command tar tvf /tmp/foo.tar would not be. This is not taken to be a serious weakness, however, as the dash convention is widely used.

[2] Multiple filenames are replaced by an appropriately numbered token. For example, the parser would emit a set of five filename arguments as <5>

3.1 Computing Sequence Similarity

A number of possible methods exist for measuring the similarity of two sequences. The most straightforward is the equality function, which yields `TRUE` when both sequences match in every position and `FALSE` otherwise. This is the similarity function employed by string matching algorithms and has the advantage of being widely studied and highly optimizable. Forrest, et al. (1996) employed equality matching between sequences in their anamoly detection system. In our domain, the difficulty is that, because of human variability, for long sequences the probability of locating exact matches in historical command data becomes exceedingly low. Thus, the equality function is not a viable choice for this particular domain.

Our system computes a numerical *similarity measure* that returns a high value for pairs of sequences that it believes to have close resemblance, and a low value to pairs of sequences that it believes largely differ. The similarity measure is based on the intuition that token matches separated by interleaving tokens are more likely to have occurred by chance, while adjacent matches are more likely to have been brought about by a causal process. Therefore, if sequence Seq_1 has k tokens in common with each of Seq_2 and Seq_3, but the common tokens are adjacent in Seq_1 and Seq_2 then we would like the similarity measure to have the property that $\mathrm{Sim}(Seq_1, Seq_2) > \mathrm{Sim}(Seq_1, Seq_3)$. To this end our similarity measure assigns similarity scores, $\mathrm{Sim}(Seq_1, Seq_2)$ as follows:

- Set an adjacency counter, $c := 1$ and the value of the measure, $\mathrm{Sim} := 0$.
- For each position, i, in the sequence length:
  - If $Seq_1(i) = Seq_2(i)$ then $\mathrm{Sim} := \mathrm{Sim} + c$ and increment c by 1.
  - Otherwise, $c := 1$.
- After all positions are examined, return the measure value.

This measure yields a higher score for more similar sequences, bounded between 0 and $n(n+1)/2$ (where n is the sequence length) and biased toward adjacent identical tokens rather than identical tokens separated by some non-matching intermediate tokens. We chose a polynomial upper-bound for our sequence measure based on the observation that the elements in a command sequence are not independent. (If they were independent then a similarity based on a function that grows exponentially with the number of matching tokens would be a better choice.) Thus, the pair of sequences shown below on the left would have a higher similarity value than would the pair on the right.

```
ls <1> ; vi                 ls -l <1> ;
ls <1> cat <3>              ls -a <1> cat
```

We define the similarity of a single sequence Seq_i to a set of sequences L as:

$$\mathrm{Sim}(Seq_i, L) = \max_{Seq_j \in L} \{\mathrm{Sim}(Seq_i, Seq_j)\}$$

Thus, the similarity of a sequence to the user dictionary is the measure of that sequence compared to the *most* similar sequence in the dictionary.

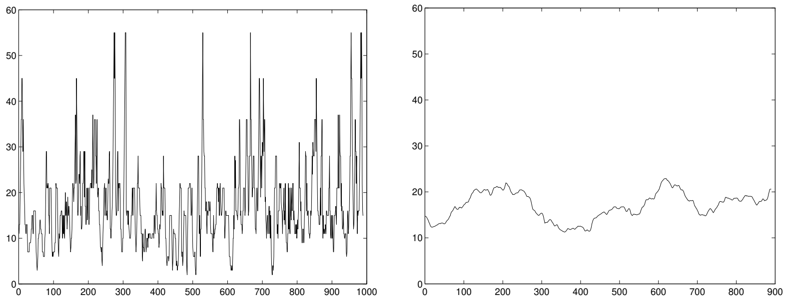

Figure 1: Similarity measure stream. **(a)** Raw. **(b)** Smoothed.

3.2 Classifying User Behavior

Given an input stream of command tokens parsed by the data collection module, the detection module classifies the current user as normal or anomalous after each token. The output of the detection module is a stream of binary decisions indicating, at each point in the input command data, whether or not it believes that the input stream at that point was generated by the profiled user.

To make these decisions, the detection module first calculates the similarity of each input sequence to the user's dictionary, yielding a stream of similarity measures. In an intuitive sense, this stream represents the familiarity of the input commands at each time step, given knowledge about the previous behavior of the user. In preliminary experiments, we discovered that the similarity value stream produced by comparison of test data to a user profile was noisy and erratic (see Figure 1, **(a)**). The noisiness of the raw similarity measure stream can be attributed to normal deviations in actions on the parts of the users, as well as to random elements (pre-empting work to deal with urgent e-mail, for example). Although explainable, this variance in the similarity measure makes it impossible to detect anomalous behavior from a single sequence.

Based on the hypothesis that, while individual sequences may deviate from historical precedent, aggregate behavior should largely conform to historical behavior for valid users but should still noticeably deviate for intruders, we applied a smoothing filter to the data (see Figure 1, **(b)**). The smoothing filter we applied was a windowed mean-value filter, which at sequence i of the input stream is defined by:

$$m_w(i, L) = \frac{1}{w} \sum_{j=i-w}^{i} \mathrm{Sim}(Seq_j, L)$$

where L is the user profile dictionary and w is the window length.

After smoothing the similarity measure stream, the detection module makes a classification of the input stream as being normal or abnormal at the point occurring at the end of the current window. In the current implementation, the classification is made with a threshold decision: if the mean-value of the current window is greater than the threshold, classify the current window as normal, otherwise classify it as abnormal. This threshold is a parameter of the system and the choice of its value is discussed in the next section.

4 An Empirical Analysis

To evaluate our approach to anomaly detection, we performed an empirical evaluation to determine if the type II (false positive) and type I (false negative) rates of our system were acceptable. The requirement that a security system not be intrusive dictates that the accuracy of an anomaly-detection system should be high (or, more specifically, the false negative rate — the occurrence of misclassifications of harmless or normal actions as anomalous — should be low). Depending on site and security policy, false alarms can disrupt the work of system administrators or users. Furthermore, a high false alarm rate can train users or administrators to ignore the security measure. The necessary accuracy depends on the security policies of the site in question; a more security-conscious site may be willing to accept a higher false alarm rate in order to gain a higher rate of detections of actual system abuses. Therefore, the detection threshold of the classification system should also be a configurable parameter.

4.1 The Data

The data examined in this research were a set of UNIX[3] shell command histories from four members of the Purdue MILLENNIUM lab, spanning a time period of approximately four months (a little more than an academic semester). The users varied in experience and academic histories, but all were graduate students with considerable computer experience. Additionally, all users used `tcsh` and worked extensively in the X-windows environment, which may well have influenced the type of data patterns produced. Over the course of data collection, we accumulated 7,769 tokens from USER0, 23,293 from USER1, 12,585 from USER2, and 22,530 from USER3. For these experiments, we assume that the data represents appropriate and normal behavior, and does not include, for example, the actions of an real intruder masquerading as one of the users.

4.2 System Parameters

As discussed in Section 3, there are several parameters in our approach to anomaly detection. This section describes each and gives the rationale for our choices in the experiments that follow. It is important to remember that the parameter settings discussed below are based upon a limited sample size and a particular software configuration — more study is necessary to arrive at methods of selecting parameters for general usage.

Sequence length: We discovered that the number of tokens per sequence had a dramatic impact on performance. Early investigations revealed that lengths of 5 and 15 yielded generally poor accuracy, while a length of 10 resulted in much higher accuracy. These values suggested an experimental range for sequence lengths of 8 to 12 tokens.

[3]Sun Microsystem's Solaris 2.5 running on Sun Ultra SPARC workstations and Linux 2.0 running on Intel i486-based and DEC Alpha-based workstations.

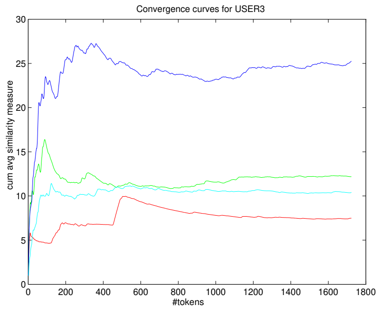

Figure 2: Impact of window length on mean sequence measure

Window length: The number of sequences included in a window of observation for the testing module was set to 80 sequences (80 tokens + n tokens, for sequence length n) based on early examinations of the windowed-mean smoothing filter. Figure 2 displays the average window value for all window sizes in the range $[0..1750]$ for each user (each curve representing the test data for a single user over all window lengths). The top curve shows the similarity measures of test data for the profiled user (USER3) while the lower curves are generated from the other three users' test data. By visual inspection, we determined that separation between the profiled user and other users occurs for window lengths of approximately eighty or more sequences. Because the window size determines the shortest interval in which the system can detect an anomaly, we attempted to select the smallest window that yielded discrimination. From these curves, it is clear that using this similarity measure allows the valid user's behavior to be discriminated from that of other users.

Classification threshold: This value was set based on initial observations at a value of 15 for all experiments. In Figure 2, we see that after eighty tokens thresholding at 15 discriminates that valid user from the invalid users. A single setting is crude because the upper bound of the similarity measure varies according to the sequence length. In future work we will investigate the sensitivity of the results to this parameter.

Dictionary size: Initially, all available training data was allocated to the dictionary, but in order to examine the amount of data required to profile a user, and to examine the possibilities of selectively pruning instances, dictionary sizes of 50, 200, 500, 1000, and 2000 sequences were tested.

4.3 Experimental Method

We examined the performance of the base-line system across the parameters given in Section 4.2. For the purposes of this domain, we define the *detection accuracy* or *true detection rate* to be the number of input windows correctly categorized as normal or abnormal (originating with the profiled user or not). The data sets were divided into train (from which the user dictionary was created) and an independent test set at a split of $2/3$ to $1/3$, respectively. To obtain the desired dictionary sizes (50, 200, 500, 1000, and 2000 sequences), the system truncated the training data to the desired number of sequences, keeping the oldest data (i.e. the earliest records available for each user). In

Profiled User	Tested User			
	USER0	USER1	USER2	USER3
USER0	99.19	35.35	6.11	0.00
USER1	17.84	88.30	23.32	1.25
USER2	3.52	54.86	72.10	8.29
USER3	6.27	15.74	11.52	69.85

Table 1: All users tested against all profiles

addition, we varied the length of sequences examined from 8 to 12 tokens. For each user, dictionary size, and sequence length, we created a user profile sequence dictionary and then measured the detection accuracy for the test data from each of the four users.

4.4 Results

This experiment was designed to test the hypothesis that user behavior could be learned and anomalous behavior detected through the use of characteristic sequences. In addition, we were interested in answering the following questions: 1) What is the effect of sequence length on detection accuracy? and 2) Is the optimal dictionary size dependent on user?

Differentiation of users: A central hypothesis of our anomaly detection system is that user patterns are sufficiently consistent, for a single user, yet sufficiently disparate, when measured between users, that differentiation is possible. In this experiment we show that differentiation is, in fact, possible within the scope of our test data. Table 1 displays the detection results for all pairwise tests of user profiles and users using sequence length of 12 and a dictionary size of 2000 sequences. Recall that the test data sets are $1/3$ of the total user data, as given in Section 4.1. The profiled user is listed in the leftmost column, while the user from whom the test data was generated is listed across the topmost row. The numbers in the table are percentages of windows that the detection system identified as the profiled user. Ideally, the diagonal elements of the table (true positive rates) should be 100% and the off-diagonal elements (false positive rates) should be 0%.

The results demonstrate that the recognition system has higher true detection than type II error rates. Furthermore, in some cases (USER1 tested against USER0 and USER2 tested against USER1, for example), the type I error rate is lower than the type II error rate. This is a desirable characteristic as false negatives make the system less usable (because of the annoyance of false alarms). We take these results as evidence that sequence-based behavioral profiling is beneficial for at least some users in this domain, although we note that the users involved in this study are all fairly to extremely experienced computer users. The question of whether or not the techniques presented here would apply equally well to novice users is still open.

The effect of sequence length on detection accuracy: Our experiments were designed to examine the impact of sequence length on detection rate, as well as the question of whether optimal sequence

	50	200	500	1000	2000
8	2.9	4.8	62.2	91.7	96.0
9	5.2	6.5	78.1	97.2	97.7
10	5.7	11.1	85.4	97.8	98.5
11	8.0	15.2	88.9	98.5	98.9
12	9.7	19.2	92.1	98.8	99.2

Table 2: USER0's test data tested against USER0's profile

length is user dependent. For brevity, the full set of experimental results is omitted here, but Table 2 displays some typical trends. The numbers in this table are percentages of the input stream identified as USER0 input (equivalent to detection accuracy for this case). The column headings are dictionary size and the row headings are sequence length.

A positive relation between sequence length and detection rate is seen over the range of sequences examined in this experiment. As mentioned earlier, this trend reverses at longer sequence lengths. We also noted that the false positive rate increases (erroneously classifying normal behavior as anomalous) when the sequence length increases.

Note that the sequence length has the most dramatic impact for a dictionary size of 500 sequences for this user, and, simultaneously, the dictionary size of 500 sequences represents a dramatic accuracy improvement over a dictionary size of 200 sequences. When we examined the usage patterns for the dictionary elements, we found that, even when the dictionary size is unrestricted, only 850 dictionary sequences are ever selected as `most similar' to an input sequence. It is probable that, for this user, most of the behavioral information is contained in a few characteristic sequences. This led us to explore the effect of dictionary size on accuracy.

Optimal dictionary size: The observation that most of the information is contained in relatively few instances for USER0 leads to the question of whether optimal dictionary size is invariant of the user profiled. Tables 3 **(a)-(d)** suggest that ideal dictionary size is user specific. In these tables, the column headings indicate the size of the dictionary used in the user profile, while the row headings indicate the test set under examination. `SELF' denotes a test of the user's data against that same user's profile. The numbers in the table are percentages of the input stream classified as normal (belonging to the profiled user). Thus, the ideal values for the `SELF' row are 100% and the ideal values for other rows are 0%.

It appears that, while for USER0 most of the important behavioral data is extracted within 500 sequences, for USER3 significant information is still being acquired by the 1000 and 2000 sequence points. USER2 and USER3's accuracies do not appear to asymptote on this range of sequences. There are a two possible explanations for this behavior. The first is that users 1, 2, and 3 are characterized by a small number of sequences, but that those sequences occur infrequently, requiring a larger sample to acquire. Under this hypothesis, it is possible that a single dictionary size is ap-

	200	500	1000	2000
SELF	19.1	92.1	98.8	99.2
USER1	9.0	12.90	27.6	35.4
USER2	0.0	0.0	2.0	6.1
USER3	0.0	0.0	0.0	0.0

(a)

	200	500	1000	2000
SELF	46.0	74.4	82.6	88.3
USER0	0.0	7.4	12.0	17.8
USER2	4.7	10.1	15.6	23.3
USER3	0.0	1.0	1.0	1.3

(b)

	200	500	1000	2000
SELF	4.6	21.7	55.0	72.1
USER0	0.0	1.5	2.8	3.5
USER1	8.3	22.4	46.1	54.9
USER3	0.0	0.2	1.4	8.3

(c)

	200	500	1000	2000
SELF	14.3	36.9	55.3	69.8
USER0	1.4	1.8	2.9	6.3
USER1	0.5	2.4	6.1	15.7
USER2	0.0	2.1	5.6	11.5

(d)

Table 3: Profiled users (SELF) versus all other users for various dictionary sizes.

plicable to all users, and that the important sequences occurred early for USER0 by chance. Alternatively, it is possible that different dictionary sizes are necessary for superior performance for different users (intuitively, this seems to be the more likely explanation). This issue is complicated by the result that the false positive rate seems to increase with increasing dictionary size; it is desirable to maintain the smallest acceptable dictionary for accuracy as well as resource reasons.

Instance Selection: The amount of data that is available for examination on a per-user basis is potentially staggering. If the anomaly-detection system is to run real-time then it is imperative that the system be both fast and resource conservative. This implies that much of the available data must be discarded with little or no examination. The experiments reported in this paper provided evidence that optimal dictionary size is user dependent. This suggests that a gain in resource efficiency can be made by discarding some historical command data. In other work [Lane & Brodley, 1997], we have investigated sequence selection for this domain. We found an analogy to the operating-systems problem of page replacement selection to be useful, and developed an instance selection algorithm based on the least-recently-used (LRU) page replacement policy. In testing, we determined that this algorithm was able to greatly reduce the dictionary size while retaining most of the classification accuracy for many of the tested users. We are currently investigating other instance-selection methods based on the page replacement policy analogy.

5 Informed Malicious Users

A distinct challenge presented to anomaly detection systems is that of detecting an insider or other well-informed intruder. Such a user is presumed to have full knowledge of the system defenses, including the anomaly detection system and its user profiles. Given such knowledge, the invader can attempt to conform his or her behavior to that recorded in a profile for a valid user, thereby avoiding notice by the anomaly detection system. In principle, with sufficient sophistication and care, any user's profile can be emulated. The counterbalancing factor is that the intruder presumably desires to accomplish different tasks (such as system penetration or corruption) than does the valid user. The goal is for this deviation to appear distinct to the anomaly detection system in spite of efforts on the intruder's part to mask anomalous behaviors.

We argue that profiles based on behavioral sequences are difficult to successfully emulate. Not only must the intruder accomplish the desired goal within a set of commands similar to that employed by the valid user, but the command issue order must also be similar. For many users and known attack patterns (e.g. system browsing or multiple login attempts), those two conditions might be difficult to obtain simultaneously. Furthermore, if the detector is extended to include such factors as command issue times, keystroke rate, or GUI events the emulation task would become yet more difficult. Such extensions are conceivable for the system as it exists. For example, time of day for usage could be encoded by timestamping each token and modifying the similarity function to match two tokens with a strength proportional to the difference in their times of day. For added protection, the adaptive anomaly detection system could also be augmented with a rule-base of known attacks as in [Lunt, 1990, Heberlein, Dias, Levitt, Mukherjee, Wood & Wolber, 1990].

A major goal of an adaptive anomaly detection system is to deal with *concept drift* — the behavioral changes undergone by valid users in the normal course of events. Such adaptability, however, leaves a window of opportunity for a malicious attacker. The malicious user may be able to subvert the detection system by *training* it — initially conforming to expected behavior but gradually changing to malicious behavior in such a way as not to appear suspicious. The introduced problem is one of distinguishing real concept drift (introduced by normal changes in behavior on the part of a valid user) from hostile training.

6 Conclusions and Future Work

This research has demonstrated a number of points. The first is that command sequence learning can be a valuable technique in the domain of anomaly detection for user recognition in computer security. Because of the somewhat specialized nature of the data (graduate students in computer engineering), it is not clear to what extent this result generalizes. The experiments have provided empirical evidence that the optimal dictionary size is a function of the profiled user.

There are a number of directions available at this point for future research. It is possible that greater accuracy can be achieved by replacing the mean-value smoothing operation (Section 3.1) with a different noise-suppression algorithm. Similarly, it might be beneficial to apply a more sophisticated discrimination test than comparison to a constant threshold value (see Section 3.2). In future research we will investigate methods for setting the value of the window length and the detection threshold, on a per-user basis, by examining statistics of the user's smoothed input sequence and setting the threshold in accordance with a pre-selected false negative tolerance level. In addition we plan to investigate alternative instance selection methods.

One potential problem that this research has not addressed is, that as time passes, normal user actions will change – they will use different applications or read the UNIX manual. This means that some of the old sequence data will no longer accurately reflect the user's behavior. To handle this *concept drift* [Schlimmer, 1987] a method is needed to remove out-of-date data sequences from the dictionary. At the same time, allowance must be made for the the threat of hostile training of the system. A direction for future work will be to examine the tradeoff between adapting to legitimate change and protecting against hostile training.

Acknowledgments
We would like to thank T. Stough, G. Spaford, R. Kohavi, P. Utgoff, and C. Codrington for their helpful comments and the members of the Purdue MILLENNIUM Lab for their contributions of data. A portion of this research was funded by the commercial and government sponsors of the COAST Laboratory: Cisco Systems, HP, Schlumberger, MITRE, Sprint, Sun Microsystems, Hughes Research Laboratories, Thompson Consumer Electronics, and the U.S. Department of Defense.

References

[Anderson, 1980] Anderson, J. P. (1980). *Computer security threat monitoring and surveillance*, (Technical Report), Washington, PA, James P. Anderson Co.

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

[Forrest, Hofmeyr & Somayaji, 1996] Forrest, S., Hofmeyr, S., & Somayaji, A. (1996). Computer immunology. *Communications of the ACM*.

[Frank, 1994] Frank, J. (1994). Machine learning and intrusion detection: Current and future directions. *Proc. of the 17th National Computer Security Conference*.

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

[Kumar 1995] Kumar, S. (1995) *Classification and Detection of Computer Intrusions,* Ph.D, Thesis, Department of Computer Sciences, Purdue University, W. Lafayette, IN

[Lane & Brodley, 1997] Lane, T., & Brodley, C. E. (1997). *Detecting the abnormal: Machine learning in computer security*, (TR-ECE 97-1), West Lafayette, IN: Purdue University.

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

[Schlimmer, 1987] Schlimmer, J. C. (1987). *Concept acquisition through representational adjustment*. Doctoral dissertation, University of California, Irvine.

[Smaha, 1988] Smaha, S. E. (1988). Haystack: An intrusion detection system. *Proceedings of the Fourth Aerospace Computer Security Applications Conference* (pp. 37-44).