Detecting the Abnormal: Machine Learning in Computer Security

Terran Lane and Carla E. Brodley
School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

January 31, 1997

Keywords: Application, Learning from positive examples, Sequence learning, Classification, Recognition, Computer security, Anomaly detection.

Contents

ABSTRACT

Two problems of importance in computer security are to 1) detect the presence of an intruder masquerading as the valid user and 2) detect the perpetration of abusive actions on the part of an otherwise innocuous user. In this paper we present a machine learning approach to anomaly detection, designed to handle these two problems. Our system learns a user profile for each user account and subsequently employs it to detect anomalous behavior in that account. Based on sequences of actions (UNIX commands) of the current user's input stream, the system compares each fixed-length input sequence with a historical library of the account's command sequences 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 in most cases it is possible to distinguish the legitimate user from an intruder and, furthermore, that an instance selection technique based on a memory page-replacement algorithm is capable of drastically reducing library size without hindering detection accuracy.

Detecting the Abnormal: Machine Learning in Computer Security

1 Introduction

A long-standing problem in the field of computer security is that of *intrusion detection* [Anderson, 1980]. According to Mukherjee et al. [1994], the problem is to identify "individuals who are using a computer system without authorization (crackers) and those who have legitimate access to the system but are abusing their privileges (the insider threat)." This problem is a subcase of the *anomaly detection* problem, in which the goal is to identify anomalous situations that may cause impairment in system usability through loss of data, denial of service, or invasion of privacy. 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. 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. Creating such a system presents four challenging problems:

- The definition of the class abnormal is often site or user dependent.
- If we consider the problem of learning user profiles as a concept learning task, we face the difficulty that the profile must be formed from positive examples only.
- There is potentially an unlimited amount of data available for user profiling.
- Changes in user behavior lead to concept drift, which must also be incorporated into the user profile.

This paper explores how to learn a user profile and how to employ the profile for anomaly detection. Our empirical results demonstrate that an approach based on profiling a user through characteristic sequences of commands yields high detection accuracy.

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 then discuss implementation details of how user profiles are formed from command data.

2.1 Capturing the Casual Nature of User Actions

Traditionally, in computer security, user profiles have been built based on characteristics such as resources consumed, typing rate, command issue rate, and counts of particular commands employed [Denning, 1987, Smaha, 1988, Frank, 1994]. It is unclear how successful these approaches have been because, although a number of applications have been fielded and are in use, to our knowledge rigorous comparative testing has yet to be performed[1]. 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.

[1] As reported by members of the COAST security research lab. COAST is a computer security laboratory in the Computer Science Department at Purdue University.

2.2 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. 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 (directories in which the user typically works, for example).

We envision our anomaly detection system as a personal software assistant 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.

Norton has explored sequence learning for DNA sequences [Norton, 1994], but his data had both positive and negative training examples. The anomaly detection domain differs from traditional concept formation tasks in that one must characterize user behavior from "positive" examples only. To resolve this difficulty we invoked the *closed world* assumption — that anything not seen in the historical data represents a different user. Indeed, one goal of this research was to examine the appropriateness of the closed world assumption for the anomaly detection domain. 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.

2.3 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[2], 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[3]. 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 *library*. The library is an instance database that, together with a similarity measure and a set of system parameters (described below), constitutes a user's profile. Similar to instance based learning, a design criterion is whether one collects all available data or performs some type of instance selection [Aha, et al., 1991, Lewis & Catlett, 1994]. We explore a strategy for instance selection in Section 5.

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

[2]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.

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

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. Dietterich and Michalski [1996] have studied the problem of learning to predict sequences by fitting sequence data to a model from a space of possible models. Their goal was to create a system that could predict subsequent actions in the sequence, whereas our goal is to classify sequences of new actions as consistent or inconsistent with sequence history. To this end, 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 sequences can be compared using a similarity measure.

3.1 Computing Sequence Similarity

One approach to learning from sequence data is to convert the data into feature vectors by accumulating measures of the individual sequences [Hirsh & Japkowicz, 1994, Salzberg, 1995]. Then one can apply any off the shelf classifier construction algorithm such as a neural network or a decision tree to the feature vectors that describe the sequence data. By contrast, our approach uses a measure of similarity between sequences to compare current input to historical data.

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. For example, the UNIX `diff` program employs this form of matching. Srikant and Agrawal [1996] use a modified equality matching function to detect frequently occuring sequences in large data sets; they allow gaps, or intervening non-matching elements, in their sequence detection. In our domain, the difficulty is that 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, therefore, 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 individual elements of the sequences are from an unordered set, which creates a matching problem identical to that of symbolic features for IBL. However, unlike IBL, our similarity measure is judging the similarity between two *se-*

quences rather than two feature vectors. 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 occurred due to 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 make more sense.) 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 library is the measure of that sequence compared to the *most* similar sequence in the library.

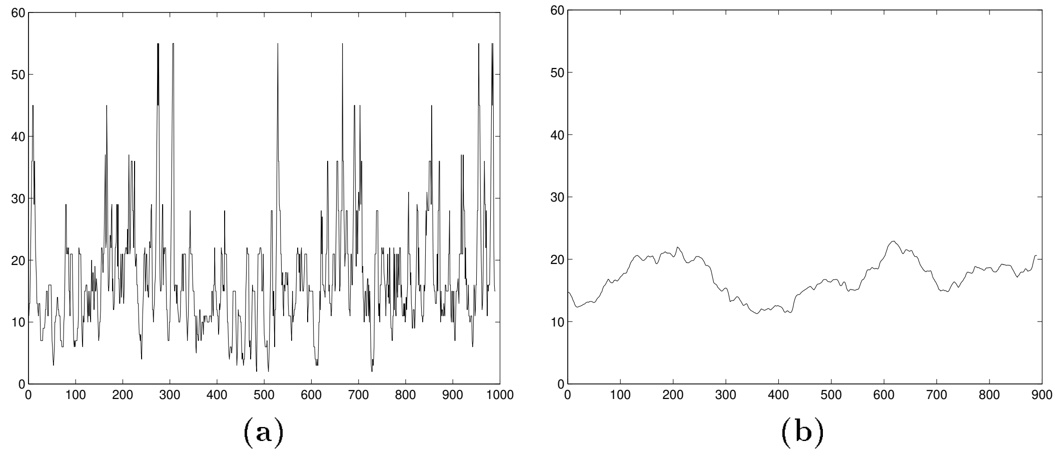

Figure 1: Similarity measure stream. **(a)** Raw. **(b)** Smoothed.

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 library, 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. (The profiled user sporadically has very low similarity with their own past behavior.)

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 library 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 Experiment 1: Proof of Concept

To evaluate our approach to anomaly detection, we performed an empirical evaluation to determine if the *false positive* and *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 very 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. If the system is constantly flagging valid users as abnormal it will quickly gain the distrust of both users and system administrators, much in the same way that the 'boy who cried wolf' lost the trust of his townspeople, and that car alarms are often ignored. The level of 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.

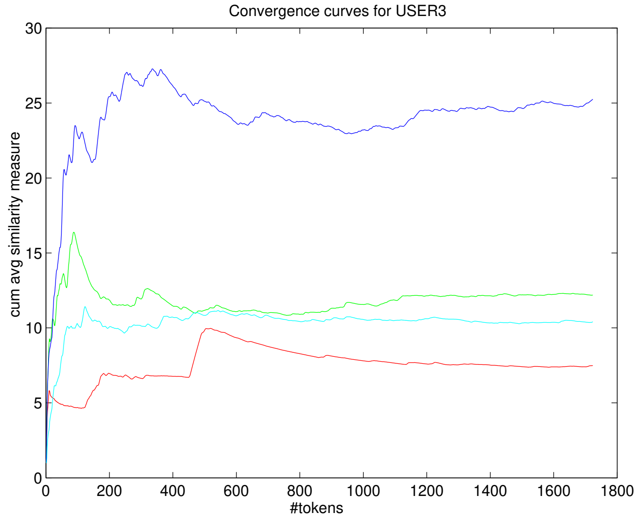

Figure 2: Impact of window length on mean sequence measure

4.1 The Data

The data examined in this research were a set of UNIX[4] 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.

4.2 System Parameters

As discussed in Section 3, there are several parameters in our approach to anomaly detection. This section describes each in detail and gives the rational for our choices in the experiments that follow.

Sequence length: We discovered that the number of tokens per sequence had

[4]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.

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.

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. By visual inspection we determined that separation occurs for window lengths of approximately 80 or more sequences. Since the window size determines the shortest interval in which the system can detect an intruder, 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. Unfortunately, limited data precluded independent verification of this value (we used all four users' data to select this window length). Since this is likely to be a user specific parameter, future work will address how to customize this choice to the particular user.

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 80 tokens thresholding at 15 discriminates that valid user from the invalid users. USER3 is the user profiled and the top curve shows the similarity measure for their actions recorded in the test data. 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.

Library size: Initially, all available training data was allocated to the library, but in order to examine the amount of data required to profile a user, and to examine the possibilities of selectively pruning instances, library 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 library was created) and test at a split of 2/3 to 1/3, respectively. To obtain the desired library 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 addition, we varied the length of sequences examined from 8 to 12 tokens. For each user, library size, and sequence length, we created a user profile sequence library and then measured the detection accuracy for the test data from each of the four users.

4.4 Results

This experiment was intended largely as a proof-of-concept system, and as such we were interested in answering the following questions: 1) Is the closed world assumption appropriate for this domain? 2) What is the effect of sequence length on detection accuracy? and 3) Is the optimal library size dependent on user?

Closed world assumption: 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 with sequence length of 12 and a library 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 user from whom the profile was generated 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%.

These results demonstrate that the recognition system has higher true positive than false positive rates. Furthermore, in some cases (USER1 tested against USER0 and USER2 tested against USER1, for example), the false negative rate is lower than the false positive rate. This is a desirable characteristic as false negatives make the system less usable (due to the annoyance of false alarms). We take these results as evidence that the closed world assumption is appropriate for at least some users in this domain, although we note

Profiled User	Tested User			
	USER0	USER1	USER2	USER3
USER0	99.19	35.35	6.113	0.000
USER1	17.84	88.30	23.32	1.251
USER2	3.519	54.86	72.10	8.292
USER3	6.270	15.74	11.52	69.85

Table 1: All users tested against all profiles

	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

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 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 library 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 library

	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 library sizes.

size of 500 sequences for this user, and, simultaneously, the library size of 500 sequences represents a dramatic accuracy improvement over a library size of 200 sequences. When we examined the usage patterns for the library elements, we found that, even when the library size is unrestricted, only 850 library 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 library size on accuracy.

Optimal library size: The observation that much information is contained in relatively few instances for USER0 leads to the question of whether optimal library size is invariant of the user profiled. Tables 3 **(a)**-**(d)** suggest that ideal library size is user specific. In these tables, the column headings indicate the size of the library 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 detected as the same as the profile. 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 also characterized by a small number of sequences, but that those sequences occur infrequently, and thus require a larger sample to acquire. Under this hypothesis, it is possible that a single library size is applicable to all users, and that the important sequences occurred early for USER0 more-or-less by chance. Alternatively, it is possible that different library sizes are necessary for superior performance for different users. This issue is complicated by the result that the false positive rate seems to increase with increasing library size; it is desirable to maintain the smallest acceptable library for accuracy as well as resource reasons.

5 Experiment 2: Instance Selection

The amount of data that is available for examination on a per-user basis is potentially staggering. If the granularity of examination is reduced to the level of individual operating system calls, the data stream could well amount to thousands, or even millions, of data per second. If the anomaly-detection system is to run real-time then it is imperative that the system be both fast and resource conservative (history has shown that a security measure that is sufficiently obtrusive will not actually be used, and will, therefore, be useless.) This implies that much of the available data must be discarded with little or no examination.

5.1 The LRU Instance Selection Algorithm

The experiments reported in Section 4 provided evidence that optimal library size is user dependent. This suggests that a gain in resource efficiency can be made by discarding some historical command data. The hypothesis that some sequences are more characteristic of a user's behavior than others suggests a possible strategy for deleting unnecessary sequences. We note, first, that the algorithm employed in the first experiment 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 library 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 discarded in favor of others. To examine this analogy, we modified the anomaly detection system from Experiment 1 to employ the least-recently-used (LRU) discard strategy. Under this strategy, a set of *pruning sequences* (separate from both train and test sequences) are selected and used to mark which sequences in the library are used for detection. As each pruning sequence is examined, the library instance selected as most similar is time-stamped. After all pruning data is processed, the library is reduced to the desired size by removing the least-recently-used sequences. The resulting pruned library is then employed as the user profile to classify input streams. Strictly speaking, LRU is an iterative algorithm, while our implementation for these experiments was batch mode.

5.2 Experimental Verification

The experimental runs described in Section 4 were repeated, with the exceptions that 1) library size was achieved with the LRU instance selection algorithm rather than library truncation and that 2) of the 1/3 of user data reserved for testing, only 1000 sequences were used as test data for each user, the rest being employed as pruning data. Characteristic results are given in Tables 4 **(a)**-**(d)**. The format of these tables is identical to the format of Tables 3 **(a)**-**(d)**.

For USER2 and USER3, these results display a dramatic increase in true detection rate, accompanied by a decrease in false positive rate. USER0, on the other hand, experiences a slight drop in true positive rate for large library sizes, along with increases in false negative rate in some cases. There are two noteworthy features of USER0's results. The first is that the true detection rate (the SELF rate) for a library size of 200 elements is much greater than for the equivalent entry in Table 3 (improved by 50 percentage points). This suggests that the LRU pruning algorithm can be useful even for this user. It also suggests, however, that the LRU selection technique might asymptote at lower accuracies for the other users as well, if the experiments were extended to cover larger library sizes. The second interesting factor in Table 4 for USER0 is that, while false positive rates increased with respect to USER1, they declined or remained constant with respect to USER2 and USER3. This raises the possibility that the optimal instance selection scheme is not merely a function of the user being profiled, but also of the intruder.

	200	500	1000	2000
SELF	69.4	86.8	90.7	93.4
USER1	13.3	37.6	46.7	50.9
USER2	0.0	0.0	0.0	0.0
USER3	0.0	0.0	0.0	0.0

(a)

	200	500	1000	2000
SELF	28.3	89.8	100.0	100.0
USER0	2.1	15.4	25.8	36.7
USER2	0.0	1.3	8.1	11.4
USER3	0.0	0.0	0.0	0.0

(b)

	200	500	1000	2000
SELF	10.7	47.1	70.1	95.0
USER0	2.2	3.1	3.4	10.6
USER1	1.8	16.0	42.8	60.1
USER3	0.0	0.0	0.0	0.9

(c)

	200	500	1000	2000
SELF	32.3	69.9	80.4	89.8
USER0	0.0	1.2	2.9	5.9
USER1	0.0	0.0	0.0	0.2
USER2	0.0	0.0	3.2	3.6

(d)

Table 4: Results of LRU instance selection.

6 Conclusions and Future Work

This research has demonstrated a number of points. The first is that sequence learning can be a valuable technique in the domain of anomaly detection for user recognition in computer security. The experiments have provided empirical evidence that the optimal library size is a function of the profiled user and further that an instance selection system can lead to increased performance. We found that the least-recently-used instance selection technique yields substantial performance benefits for some users in our test sets, but not universally. This points the way toward investigation of other instance selection techniques for the security anomaly detection domain.

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 library similar to removing instances from instance-based learning systems [Moore, 1990]. Fortunately, these sequences can be detected as they are older and will not have been recently used in matching (new behavior looks different). A focus of future work will be to explore the LRU method for adapting user-profiles to concept drift.

References

[Aha, et al., 1991] Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37-66.

[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.

[Dietterich & Michalski, 1986] Dietterich, T. G. , & Michalski, R. S. (1986). Learning to predict sequences. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

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

[Hirsh & Japkowicz, 1994] Hirsh, H., & Japkowicz, N. (1994). Bootstrapping training-data representations for inductive learning: A case study in molecular biology. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 639-644). Seattle, WA.

[Lewis & Catlett, 1994] Lewis, D., & Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 148-156). New Brunswick, NJ: Morgan Kaufmann.

[Moore, 1990] Moore, A. W. (1990). Aquisition of dynamic control knowledge for a robot manipulator. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 244-252). Austin, TX: Morgan Kaufmann.

[Mukherjee, et al., 1994.] Mukherjee, B., Heberlein, L. T. , & Levitt, K. N. (1994.). Network intrusion detection. *IEEE Network, 8*, 26-41.

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

[Salzberg, 1995] Salzbery, S. (1995) Locating Protein Coding Regions in Human DNA using a Decision Tree Algorithm. *Journal of Computational Biology, 2(3)*, 473-485.

[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).

[Srikant & Agrawal, 1996] Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements, . *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT).* Avignon, France.

Acknowledgments:

We would like to thank Tim Stough, Gene Spaford, Ronny Kohavi, Paul Utgoff, and Craig Codrington for their helpful comments. We are also grateful to the members of the Purdue MILLENNIUM Lab for their contributions of data and insight to this work. A portion of this research was funded by the commercial and government sponsors and supporters 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.