An Application of Symbolic Learning to Intrusion Detection: Preliminary Results from the LUS Methodology

Kenneth A. Kaufman
Guido Cervone
Ryszard S. Michalski

MLI 03-2
P 03-3
June, 2003

AN APPLICATION OF SYMBOLIC LEARNING TO INTRUSION DETECTION: PRELIMINARY RESULTS FROM THE LUS METHODOLOGY

Kenneth A. Kaufman, Guido Cervone and Ryszard S. Michalski

Machine Learning and Inference Laboratory

George Mason University

Fairfax, VA 22030-4444

{ kaufman, cervone, michalski}@gmu.edu

http://www.mli.gmu.edu

Abstract

This paper describes briefly a method for applying AQ symbolic learning to problems of computer user modeling and intrusion detection. The method, called *LUS* (Learning User Signatures), learns models of users' interaction in the form of sets of rules in *attributional calculus*, and signals a possible intrusion when a user interaction with a computer violates the model. An important characteristic of LUS is that the generated user signatures are easy to interpret and understand. We describe briefly the LUS method, the machine learning and inference tools developed to support it, and selected initial experimental results from its application to real-world data.

Keywords: Intrusion Detection, Symbolic Learning, Attributional Calculus, Episode Classification, Multistate Conjunctive Patterns

Acknowledgments

The authors express their gratitude to Dr. Tom Goldring for his invaluable assistance and collaboration in conducting this project, and for providing us with the data sets used in this research. They thank Professor James Gentle for suggestions that led to the development of classification bars, and Janusz Wojtusiak for his helpful comments and suggestions. This research was performed in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research activities are supported in part by the UMBC/LUCITE #32 grant, and in part by the National Science Foundation under Grants No. IIS-9906858 and IIS-0097476. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the UMBC/LUCITE program or the National Science Foundation.

1 INTRODUCTION

The widespread availability of information technology coupled with rapidly rising numbers of computer users has created an increasing interest in *intrusion detection*, a new discipline concerned with the automatic detection of illegitimate access to computing systems (e.g., Bace, 2000).

This paper discusses an approach to *user profiling*, a discipline of intrusion detection research that focuses on creating models of the behavior of individual users in an effort to recognize illegitimate activity. Following Goldring et al (2000), user profiling can be defined as *a) observing what someone is doing on a computer, b) modeling this activity for the purpose of representing the user's "normal" behavior, and c) scoring new activity against the model.* Sufficiently high degrees of match confirm legitimate use; otherwise, illegitimate use may be indicated. A successful application of this approach requires a user modeling methodology that is able to ignore significant variations that typically occur in users' activities but capture essential features of their behavior that differentiate each user from other users.

Given a set of *U* known users, each of whom is represented by a set of temporal sequences of events, the goal is to classify a new temporal sequence as belonging to one of those users, or to recognize it as belonging to a new, unseen user. Typically, classifying the new sequence correctly suggests legitimate behavior by that user. Conversely, classifying the sequence as belonging to another user may suggest misuse, and classifying the sequence as belonging to no known user may be an indicator of an intrusion.

The presented method, called *LUS* (derived from *Learning User Signatures*), applies AQ symbolic machine learning to the induction of general and consistent patterns in the interactions between users and computers. Given records measuring various characteristics of the interaction between users and computers (as manifested, in the presented case, in a process table), LUS automatically creates models of user behavior in the form of sets of rules relating the measured characteristics to the individual users. These sets of rules, called *symbolic user signatures*, are expressions in *attributional calculus*---a logic-based language that can concisely and simply represent complex relationships based on multi-type attributes (Michalski, 2003). This feature is due to the ability of AQ learning to build descriptions with internal disjunctions (see Section 3.2), which contributes to the central objective of LUS: to generate user signatures that are not only accurate, but also compact, and easy to interpret and understand. Thus, they can be inspected and verified by experts, and hand-modified or extended, if desired.

The following sections present the application of LUS to a large set of proprietary data collected from various computer users.

2 PROBLEM DESCRIPTION

In a user profiling application, the system is provided with data streams regarding the behavior of different users. The data may represent higher level concepts, such as commands executed, or lower level ones, such as individual processes, I/O accesses, or system calls for which the user was responsible. These data are typically associated with temporal information, so that it is known which events preceded which, by how much, and whether any intervening events were recorded.

An individual user's behavior may have been recorded at several different times. A data stream consisting of data collected during all or a contiguous part of one of these collection periods is called an *episode*. Within an episode, one can say that an event immediately followed another, and thus can construct unbroken sequences of events.

The computational goal of the user profiling task is then very straightforward. During the training phase, the learner views episodes representing the behavior of known users, and builds models of those users' behavior. During the testing or application phase, episodes representing unidentified users are matched against those models, and are either identified as one of the known users, or as an unknown user (some matching methods are able to return imprecise identifications, narrowing the identity to a small number of users, without providing a single identification if it is determined not to be able to sufficiently differentiate). If in reality, the identity of the users responsible for these episodes is known, this phase serves as validation of the models, or as confirmation of the users' legitimate behavior. Misclassification or classification as unknown could suggest misuse by the purported user or another user masquerading as that user.

Given such a framework, the ultimate goal is to develop a system reliable enough both to not let intruders through undetected, and to minimize the number of false alarms. It should be able to detect misuse rapidly, in order to minimize the damage that may be inflicted by illegitimate use.

3 METHOD AND EXPERIMENTS

3.1 Input Data

Our initial experiments were applied to three separate sets of data. The first set reflects the activities of 4 users using UNIX workstations. The second set covers 24 users using Windows. The third set describes the activities of 17 users using Windows, and is filtered so that only activities in the user's active window are included. The data were collected by polling system process tables at fixed time intervals and recording changes since the last polling. Table 1 describes the three different data sets

	System	# Users	# Events
Data set 1	Unix	4	825,023
Data set 2	Windows	24	4,809,752
Data set 3	Windows	17	1,052,960

Table 1. Summary of the three experimental data sets.

Specifically, the three sets of data were acquired by scripts that would poll the system process tables every half second, and record every process that had been created, had been destroyed, or had had CPU activity since the last check of the table. For each such process, a record was created identifying the process, the process that spawned it, its status, the program it invoked, the amount of CPU time used by the process, and a timestamp of the observation. (Note: To some degree this is a simplification, but for the purposes of this paper, it will suffice.)

In the presented experiments, we extracted for learning the attributes directly representing the user's activity. In the first two data sets, this attribute, called "mode," encompassed the user's activity that caused the process in question to be invoked. Thus, even if the process was some system utility, its mode might indicate that the user was in an email program, running a compiler, using a calendar utility, etc. In the third data set, prior filtering of the data had already removed records of such lower level processes, and thus, the mode attribute was replaced directly by the process name.

In these experiments, the modes or processes were organized into *n*-grams – vectors of values from *n* consecutive events. Thus, given a training or testing episode of length N ($N > n$), it was transformed into ($N - n + 1$) overlapping *n*-grams, each starting at the next consecutive event.

Finally, the attribute domains were structured into hierarchies of related activities. For example, "netscape" and "msie" would both be placed in the higher level category of web browsers, allowing rules to, if appropriate, express concepts at that higher level of abstraction. The ability to reason with such hierarchies is one of the features of the AQ20 learning program employed in this study (Section 3.2).

3.2 Learning Engine and Knowledge Representation

To generate user signatures, the LUS experiments have employed the AQ-type learning systems AQ19 (Michalski and Kaufman, 2000) and AQ20, currently under development (Cervone, Panait and Michalski, 2001). One advantage of AQ-type learning is that it can create highly expressive and compact user signatures through the use of a rule representation language based on *attributional calculus* (Michalski, 2003). This representation combines elements of propositional calculus, predicate calculus, and many-valued logic in such a way that user signatures are directly related to equivalent natural language expressions, and easy to interpret and understand. In this domain, this representation allows rules to be reported in the form of *multistate conjunctive patterns*.

AQ-learning involves a *separate-and-conquer* algorithm, in which a positive example (in this case, an *n*-gram of the target user) is generalized against the negative examples (here, *n*-grams of other users). The generalization that best satisfies a given multi-criterion preference measure is selected for the output rule set. If any positive examples remain uncovered by this generalization, a new such example is chosen, and a new rule is generated. The process repeats until there are no more examples to be covered.

To illustrate the kind of signatures learned by AQ, Figure 1 shows an example of a multistate conjunctive pattern obtained from an attributional rule generated by the AQ20 learning program. This pattern characterizes a strong regularity in the behavior of User 23 from data set 2. Terms in {} denote alternative modes of user operation. The condition part of the pattern is an ordered Cartesian product of four sets of activities. If a user is recorded in one of the activities listed in the first set, and then moves to one of the activities in the second set, and so forth, the condition is satisfied, and User 23 is suggested.

The pattern annotations say that in the input data, there were 1,337,548 4-grams describing User 23's behavior and 2,063,808 describing users other than User 23. In terms of distinct 4-grams, User 23 had 9,712 different 4-grams in the data, and the other users had 40,602 different 4-grams. Given the high pd and pt ratios to the number of input events, and the low nd and nt values, this rule describes a strong pattern in User 23's behavior; it covers a large portion of events characterizing the user's behavior (1,296,647 of the 1,337,548 observed).

[user=23] ←
<{ControlPanel, activesync id, mail, multimedia, netscape, network,
  spreadsheet, system, wordprocess},
 {ControlPanel, activesync, web, id, logon, mail, msie, multimedia, netscape,
  network, print, spreadsheet, wordprocess},
 {ControlPanel, activesync, mail, multimedia, netscape, print, spreadsheet,
  wordprocess}
 {ControlPanel, activesync ,mail, multimedia, netscape, spreadsheet, web,
  wordprocess}>

pd=4685, *nd*=878, *pt*=1296647, *nt*=1166

```
# -- Distinct training events in target class:   9,712
# -- Distinct training events in other classes: 40,602
# -- Total training events in target class:1,337,548
# -- Total training events in other classes:       2,063,808
```

Figure 1. A multistate conjunctive pattern characterizing a user's behavior.

In earlier LUS development, learned rules would consist of conditions of the form [x_i = vs], where x_i represented the value at the *i*th position in the *n*-gram, and vs was a set of legal values for the attribute being used. Such rules are cumbersome and lead to redundancy. For instance, if a characteristic pattern of a user involves an email process preceding a print process with exactly one intervening process, and 5-grams are the representation in use, three rules will be needed to express this relationship: "x1 = email and x3 = print," "x2 = email and x4 = print," and "x3 = email and x5 = print." In 5-gram notation, the three templates would be <email, *, print, *, *>, <*, email, *, print, *> and <*, *, email, *, print>

In order to cope with this, a general *n*-gram template was built into the AQ programs. In this mode of operation, AQ removes leading and trailing wildcards in order to use a single template, <email, *, print>, which can be applied anywhere within larger *n*-grams when matching the rule to an event.

3.3 Rule Matching: User Identification from Episodes

During the testing (or execution) phase, LUS applies the user signatures to testing (or incoming) episodes from different users, and computes degrees of match between episodes and the corresponding models. High matches are indicative of legitimate users, and low matches suggest a possible misuse.

In traditional machine learning applications, the quality of the generated knowledge is estimated by matching the knowledge against a set of events, and comparing its performance with ground truth. This is not an appropriate approach in user profiling, where an event can represent one simple interaction between the user and the machine. Rather, a new way of matching needs to be applied in which the knowledge is matched against entire episodes. We call such an approach EPIC (from "Episode Classification"). This section briefly describes the EPIC program developed for LUS.

Given an episode, EPIC generates a classification of the episode with associated degrees of match for each user profile by applying a three-step process:

1) Generate a degree of match between each **event** in the episode and each **rule** in the user profiles. Three matching strategies are available in EPIC: *Strict Match*, in which an event-rule pair scores 1 if the event satisfies all of the rule's conditions, and 0 otherwise; *Selector ratio*, in which an event-rule pair scores a number between 0 and 1, inclusive, representing the ratio of the number of conditions in the rule satisfied by the event to the total number of conditions in the rule; or *Coverage ratio*, in which an event-rule pair scores 0 if the event does not satisfy all of the rule's conditions, and a number between 0 and 1, inclusive, otherwise, equal to the ratio of the number of training events of the rule's consequent class (in LUS, of the target user) satisfying the rule's conditions to the total number of training events of the consequent class.

2) Generate a degree of match between each **event** in the episode and each entire **user profile** by aggregating the degrees of match generated in (1) between the event and the profile's individual rules. Two methods of aggregation were used in these experiments: *maximum*, and *probabilistic sum*.

3) Generate a degree of match between the full **episode** and each **user profile** by aggregating the degrees of match generated in (2) between the user profile and the episode's individual events. Currently, only one method of aggregation is available in EPIC: *average*.

Once degrees of match between the episode and each user profile are calculated, EPIC makes its classifications based on *threshold* and *tolerance* parameters. All profiles that return degrees of match both above the threshold and within the tolerance of the highest degree of match attained by the episode are returned as possible classifications.

EPIC is designed to go beyond the typically used predictive accuracy measure in which for each testing event or episode, the system assigns a *single* classification that is either correct or incorrect. This single-decision schema makes the evaluation of a learning method simple, but is not adequate for many real-world problems. We argue that it is better to give a few alternative answers ("multiple decisions") that include the correct decision, or to give no definite answer ("no decision"), than to be forced to give just one answer and be incorrect. Consequently, an EPIC evaluation of the performance of a knowledge set consists of not just one number, but several numbers, each with a simple cognitive interpretation. Specifically, the output includes the following measures:

- Predictive Accuracy-Single Choice Correct
- Predictive Accuracy-Multiple Correct
- Precision-Single Choice
- Precision-Multiple Choice

- Prediction Gain
- Decision Rate
- Classification Bars

Predictive Accuracy-Single Choice Correct is the percentage of the testing events that match the correct decision with the highest degree of match. If there is more than one top match decision, one of which is correct, the answer is counted as correct. This measure is thus equivalent when there is only one top match to a standard measure of predictive accuracy.

Predictive Accuracy-Multiple Correct is the percentage of testing events that match the correct decision with a degree of match within the tolerance range of the best degree of match, and greater than the *acceptability threshold*. The tolerance range, specified by a user-defined parameter, is the percentage by which the degree of match can fall below the top degree of match and be selected as an alternative decision. If a testing event matches more than one decision with a degree of match within the tolerance range, corresponding decisions are presented as alternative decisions. If all degrees of match fall below the acceptability threshold, "no decision" (or "don't know") is returned. Thus, the two Predictive Accuracy measures are equivalent when the tolerance and acceptability threshold are both zero.

To account for cases of multiple decisions, two Precision measures are introduced, and defined as: $(e * c - d) / d * (c - 1)$ where e is the number of events that matched at least one candidate model with a degree of match equal to or above the acceptability threshold, c is the number of decision classes, and d is the number of specific decisions (unique or alternative). Thus, the precision is 1 when for each testing event only one specific decision is returned, and 0 when for each testing event all possible decisions were returned as alternatives.

Prediction Gain is the ratio of Predictive Accuracy-Single Choice Correct to the expected percentage of correct decisions if decisions are drawn randomly from a uniform distribution over the set of all decisions. This latter number is thus equal to 1/D, where D is the number of possible specific decisions.

Decision Rate is the percentage of definite decisions among all decisions (that is, the percentage of testing events that matched some decision with a degree of match above the acceptability threshold).

The classification bars allow visualization of both the accuracy and the precision for a given test set. The x-axis indicates the number of alternative decisions, with an extra column in order to represent "unknown class" decisions. The y-axis indicates how many classifications were made. The bars indicate how often the algorithm gave one answer, two answers etc., and are divided into two parts, representing correct and incorrect classifications. Figure 2 shows an abstract example of classification bars.

In the example depicted in this figure, there are four classes, and each testing event or episode can be classified as belonging to one or more of them, or as being in an unknown class if its degrees of match are all below the acceptance threshold.

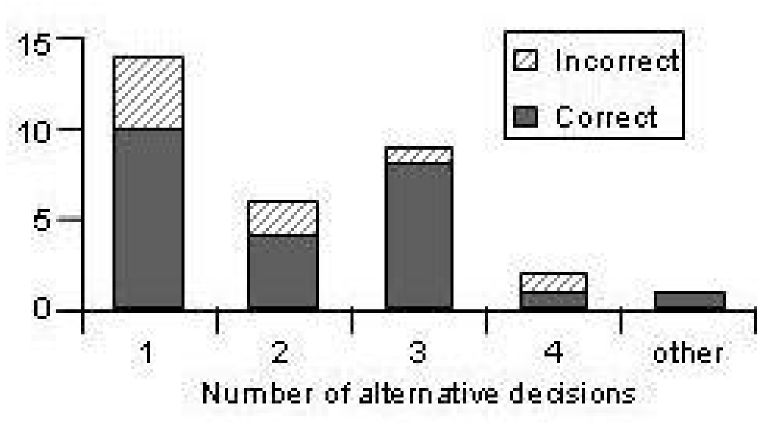


Figure 2. Classification bars.

The leftmost bar indicates that 14 times the algorithm gave one classification only. Ten of those 14 answers were correct, meaning that if an event/episode was classified as class A, it was really from class A, and 4 were incorrect. The second bar indicates how many times the algorithm classified a testing event as being of two classes, for example, in either class A or class B. Four of the six double classifications included the correct classification; two did not. Moving to the right, the correct classifications become less important, and the incorrect classifications become more important. For example, the fourth column indicates that the algorithm recognized two events as belonging to one of four classes, once with the correct classification, but once without it. Thus, that particular testing event was of an unknown class, and should have been classified as "other."

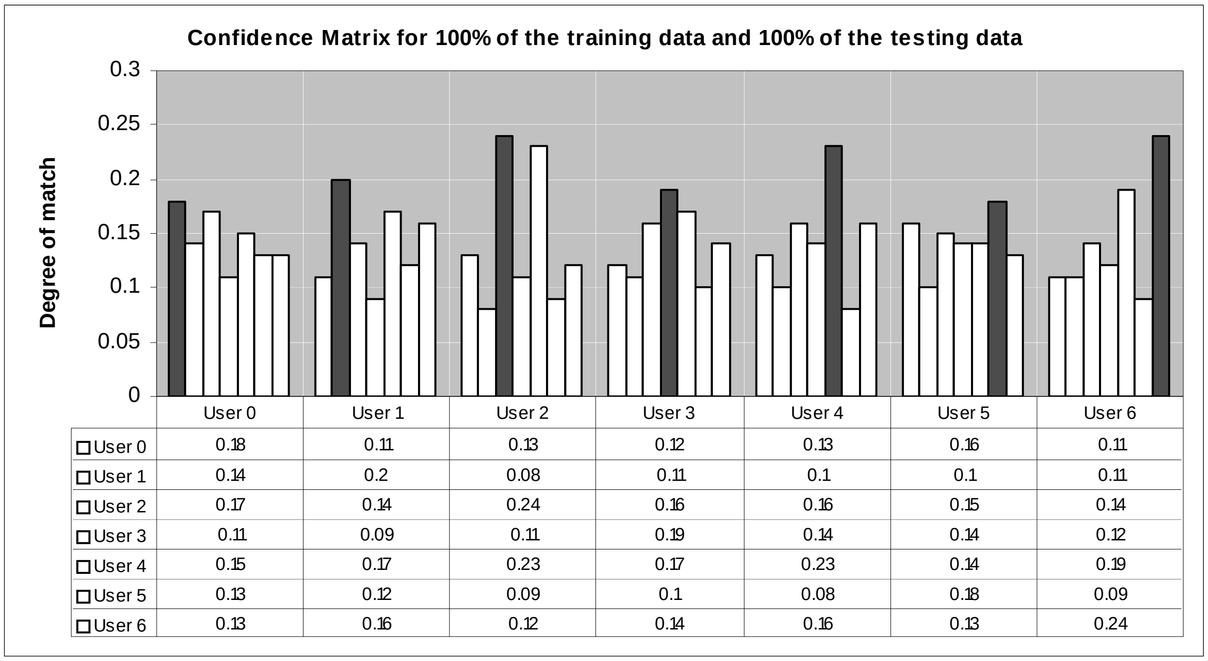
By using a parametric function that takes into account the numbers of correct and incorrect classifications, it is possible to determine a new measure that takes into consideration both predictive accuracy and precision. The function is parametric because answers that are less precise should carry less weight than those that are more precise. For example if there are four classes, and a test classifies an event as being of any of the classes, such an answer is technically correct (since the event is from one of the four classes) but maximally imprecise.

4 INITIAL EXPERIMENTAL RESULTS

On the first testing data set, consisting of only four users on UNIX machines, LUS performed very well, identifying the correct user in all but one of the 240 testing episodes. We subsequently applied LUS to the other data sets, a more challenging problem given the greater number of users represented. For example, using session (logon to logoff) sized testing episodes from the data set representing 24 Windows users, LUS achieved at best 75% performance. However, when multiple sessions were combined into longer testing episodes, 20 of the 24 users were correctly classified, one was incorrectly classified, and three returned ambiguous classifications that included the target user.

Figure 3 shows a graphical representation of a relative confidence matrix (normalized, such that all values sum to 1). Because space limitations prevent the display of a similar graph for all 24 users, this figure instead shows the full results of a smaller scale experiment involving 7 of the 24 users. In this case all seven users were correctly classified. In each case, the shaded bar represents the normalized degree of match generated by EPIC for the target user, and the other bars represent in order each of the other users. The figure indicates that these users were all classified correctly.

The *n*-gram model worked very well for the first two sets of data, but generated inferior results with the third set. The best results achieved were 66% correct with 80% precision, corresponding to the correct unique classification of 9 users out of 17 and the correct multiple classification of 2 users. Six users were classified incorrectly. An analysis of the data showed that the incorrect classifications were due largely to a completely different behavior of the user in the training and testing sets. Coping effectively with such variations is an ongoing issue for further research in the intrusion detection community.



	User 0	User 1	User 2	User 3	User 4	User 5	User 6
□User 0	0.18	0.11	0.13	0.12	0.13	0.16	0.11
□User 1	0.14	0.2	0.08	0.11	0.1	0.1	0.11
□User 2	0.17	0.14	0.24	0.16	0.16	0.15	0.14
□User 3	0.11	0.09	0.11	0.19	0.14	0.14	0.12
□User 4	0.15	0.17	0.23	0.17	0.23	0.14	0.19
□User 5	0.13	0.12	0.09	0.1	0.08	0.18	0.09
□User 6	0.13	0.16	0.12	0.14	0.16	0.13	0.24

Figure 3. Confidence matrices and normalized score table based on applying LUS to 4-grams representing 7 Windows users.

Experiments were performed in response to determine the degree of similarity between episodes belonging to the same users and those belonging to different users, computed as the ratio of the number of identical *n*-grams to the total number of *n*-grams of a particular episode. It was discovered that in this third data set, there was a high similarity between the *n*-grams of different users, and that very many events were common to any 14 users.

Figure 4 shows the number of events that were common to each number of users in this data set. The leftmost bar indicates 'unique' events, or events that are common to only one user, the next bar the number of events common to any two users, and so on.

The data was subsequently filtered by removing events common to more than 10, 7, 5, 3 and 1 user in successive experiments. Experiments were performed using 4- and 5-grams, and in each case the predictive accuracy increased with filtering. Overall the best results were obtained when events common to more than any three users were filtered, yielding a higher predictive accuracy, precision, and much faster training speed due to the very large reduction of the search space.

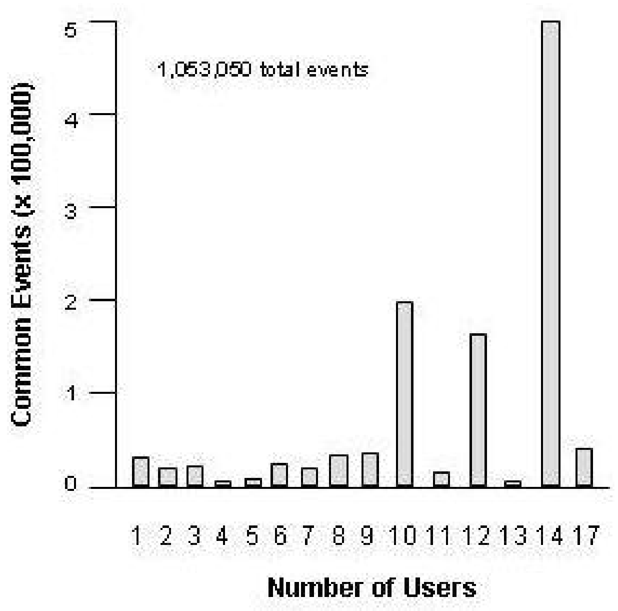


Figure 4. Number of events common to different numbers of users.

5 RELATED RESEARCH

Among the common approaches to intrusion detection are statistical methods, in particular, Bayesian methods (e.g., Goldring et al., 2000, Goldring, 2002, Valdes and Skinner, 2000; Scott, 2003), statistical modeling (e.g., Shah, Jonckheere and Bohacek, 2002), pattern matching (e.g., Hofmeyr, Forrest and Somayaji, 1998; Streilein, Cunningham and Webster, 2002), and various data analysis methods (e.g., Novak, Stark and Heinbuch, 2002; Mukkamala and Sung, 2002). A technique that learns from observation rather than from known examples, described by Eskin et al. (2002), applies multiple strategies to recorded system events mapped onto a feature space, in order to identify anomalous behavior These methods all examine overall system or network behavior in search of indicators of misuse.

A variety of methods of user profiling, in which the activities of individual users or purported users are compared to their known patterns of legitimate use, have been developed. Traditionally, user profiling has relied heavily on the use of statistical approaches. For example, Valdes (2002) describes a multi-approach system that combines sequence matching and statistical analysis to identify user behaviors, and Goldring (2002) applies a probabilistic scoring system to match episodes with user profiles.

As does the method presented here, the work of Schonlau and Theus (2000) also bases its anomaly detection on observing the various processes run. Their approach is to compile a list of commands (invoked on a Unix machine), and generate a statistical plot of commands' popularity, both for individual users, and for all users together. During the application phase, episodes are then compared to these profiles, and high levels of uniqueness may set off alarms.

An approach somewhat similar to LUS is described by Lane and Brodley (1999). They also employ an *n*-gram-based representation of sequences, but rather than using processes as the basic units of the *n*-grams, their method uses command line tokens. Thus, a single command can result in several instances in the input data stream. Their approach is to apply a similarity-based measure between known legitimate behavior and new events, but it does not incorporate a simple way of articulating the learned user patterns.

6 CONCLUSION

LUS represents a novel approach to user profiling and intrusion detection based on learning symbolic user models. Among the most important and novel features of LUS is the ability to learn multistate conjunctive patterns, which can represent compactly large numbers of cases (*n*-grams). This ability is due to the use of internal disjunction in descriptions learned by the AQ method. Another important feature is that learned models can be easily interpreted and understood by humans This feature makes it possible for users to hand-modify user models if desired. Other aspects of LUS are the use of variable length *n*-gram representation of data streams, and the use of episodes for user classification, rather than individual events in the data streams.

Initial results from applying LUS to data generated from actual computer usage have been encouraging, and indicate a potential practical viability of this approach. Because attributional rules can be executed in parallel, the recognition/monitoring phase can potentially be done very fast.

Important practical issues to be investigated in the future are the scalability of the proposed method to a large number of users, an experimental comparison of LUS with other methods, and a determination of the most appropriate settings and parameters for the AQ20 learning method for the LUS application.

AQ20 offers users a wide variety of options and settings for learning. In response to the early results, we have developed a wide-ranging schema for further experimentation in order to determine the optimal combination of models of user behavior and parameter settings for this problem. This will allow for testing of the LUS scalability to large numbers of users.

Future research will also concern several new tasks of great practical importance. Up to this point, the LUS research has focused on user identification. The described method is likely to recognize an impostor. A more difficult problem is to detect the “insider threat,” i.e., cases in which an authorized user may engage in unauthorized activity. Another important problem is *concept drift*, that is, the problem of automatically modifying the user model to reflect changes in the user behavior that typically occur over time. Methods of incremental learning might be applicable to this issue.

REFERENCES

Bace, R.G., *Intrusion Detection*, Indianapolis: Macmillan Technical Publishing, 2000.

Cervone, G., Panait, L.A. and Michalski, R.S., “The Development of the AQ20 Learning System and Initial Experiments,” *Tenth International Symposium on Intelligent Information Systems (IIS'01)*, Zakopane, Poland, 2001.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L. and Stolfo, S., “A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data,” in Barbara, D. and Jajodia, S. (eds.), *Applications of Data Mining in Computer Security*, Kluwer, pp. 77-102, 2002.

Goldring, T., “Recent Experiences with User Profiling for Windows NT,” *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Goldring, T., Shostak, J., Tessman, B. and Degenhardt, S., “User Profiling (Extended Abstract),” NSA unclassified internal report, 2000.

Hofmeyr, S., Forrest, S. and Somayaji, A., “Intrusion Detection using Sequences of System Calls,” *Journal of Computer* Security 6, pp. 151-180, 1998

Lane, T. and Brodley, C.E., “Temporal Sequence Learning and Data Reduction for Anomaly Detection,” *ACM Transactions on Information and System Security* 2, pp. 295-331, 1999.

Michalski, R.S., “Attributional Calculus: A Logic for Natural Induction,” *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 2003.

Michalski, R.S. and Kaufman, K.A., “The AQ19 System for Machine Learning and Pattern Discovery: A Description and User's Guide,” *Reports of the Machine Learning and Inference Laboratory*, MLI 00-4, George Mason University, Fairfax, VA, 2000.

Mukkamala, S. and Sung, A., “Comparison of Neural Networks and Support Vector Machines in Intrusion Detection,” *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Novak, J., Stark, V. and Heinbuch, D., “Zombie Scan,” *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Schonlau, M. and Theus, M., “Detecting Masquerades in Intrusion Detection Based on Unpopular Commands,” *Information Processing Letters* 76, pp. 33-38, 2000.

Scott, S., “A Bayesian Paradigm for Designing Network Intrusion Detection Systems,” *Computational Statistics and Data Analysis*, 2003 (to appear).

Shah,K., Jonckheere, E. and Bohacek, S., “Detecting Network Attacks through Traffic Modeling,” *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Streilein, W.W., Cunningham, R.K. and Webster, S.E., “Improved Detection of Low-Profile Probe and Novel Denial-of Service Attacks,” *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Valdes, A., “Profile Based Intrusion Detection: Lessons Learned, New Directions, *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Baltimore, MD, 2002.

Valdes, A. and Skinner, K., “Adaptive, Model-based Monitoring for Cyber Attack Detection,” in Debar, H., Me, L. and Wu, F. (eds.), *Lecture Notes in Computer Science #1907 (from Recent Advances in Intrusion Detection, RAID-2000)*, Springer-Verlag, 2000.

A publication of the *Machine Learning and Inference Laboratory*
School of Computational Sciences
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: K. A. Kaufman