Intelligent Information Systems 2006
New Trends in Intelligent Information Processing and Web Mining
Ustron, Poland, June 19-22, 2006

Learning Symbolic User Models for Intrusion Detection: A Method and Initial Results

Ryszard S. Michalski[1,2], Kenneth A. Kaufman[1], Jaroslaw Pietrzykowski[1], Bartlomiej Sniezynski[1,3], and Janusz Wojtusiak[1]

[1] Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA
[2] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland,
[3] AGH University of Science and Technology, Krakow, Poland

Abstract. This paper briefly describes the LUS-MT method for automatically learning user signatures (models of computer users) from datastreams capturing users' interactions with computers. The signatures are in the form of collections of multistate templates (MTs), each characterizing a pattern in the user's behavior. By applying the models to new user activities, the system can detect an imposter or verify legitimate user activity. Advantages of the method include the high expressive power of the models (a single template can characterize a large number of different user behaviors) and the ease of their interpretation, which makes possible their editing or enhancement by an expert. Initial results are very promising and show the potential of the method for user modeling.

1 Introduction

The widespread availability of information technology and rising computer use have increased interest in *intrusion detection*, the detection of illegitimate access to computer systems (e.g., [2]). This paper describes an approach to intrusion detection based on user profiling – creating models of user behavior and matching them with observed activities. Differences between a user model and observed activity indicate potential illegitimate use [7]. Although the presented initial research concentrated on modeling a small set of users, the method can be applied to large numbers of users. It can capture characteristics of users' behavior that differentiate them from each other, while ignoring irrelevant variations.

The method, called LUS-MT (*Learning User Signatures-Multistate Template* model), applies symbolic machine learning to induce patterns in users' computer interactions. Given records measuring various characteristics of user activity, LUS-MT learns models in the form of *multistate templates* that relate the measured characteristics to individual users. These models can be easily matched in parallel against datastreams, and incrementally updated as needed. They can be inspected, verified and hand-modified by experts. The following sections briefly describe LUS-MT, and present initial results from its application to a collection of datastreams available from the New Jersey Institute of Technology website.

2 Objectives

The objectives of this research are to develop methods for creating effective and efficient user models that resemble human recognition capabilities in terms of:

- *Idiosyncrasy:* User models should be able to capture patterns that are most characteristic of the given user, but not of other users, making it possible to identify the given user from short periods of observation (episodes).
- *Satisfiability:* If at some stage of observing a datastream, the observed behavior strongly matches exactly one user model, no further observation is conducted, and a decision is reported.
- *Understandability:* User models should be easily understandable and potentially modifiable by an expert managing the intrusion detection system.
- *Incrementality:* User models should be incrementally updatable to capture changes in behavior over time, without having to completely relearn the model.
- *Efficiency:* User models should be in a form that can be efficiently applied to datastreams from a large number of users to ensure their practical applicability.

It is not the goal of this research to build a complete Intrusion Detection System (IDS); rather, we are developing building blocks for such a system, with sufficient flexibility to be applied in many real world scenarios, possibly in conjunction with other components. Evidence indicates that the presented method is able to achieve high performance with respect to very important IDS evaluation measures such as false alarm rate [9, 10, 22] or "time to detect." LUS-MT was developed and tested on real-world data, thereby avoiding the pitfalls of using simulation data [13].

Intrusion detection has been heavily investigated using a number of different methods. Most methods are statistically oriented, such as Bayesian methods (e.g., [6, 7, 20, 25]), statistical modeling (e.g., [21]), pattern matching (e.g., [8, 23]), and various data analysis methods (e.g., [16, 17]). A method that learns from observation (unsupervised) rather than from examples [5] applies multiple strategies to recorded events in order to identify anomalous behavior. These methods all examine overall system/network behavior in search of signs of misuse.

A variety of methods of user profiling also been developed. Traditionally, user profiling relies on statistical approaches. Valdes [24] describes a multi-approach system that combines sequence matching and statistical analysis to identify user behaviors, and Goldring [6] applies probabilistic scoring to match episodes with user profiles. Billsus and Pazzani [3] present a system that uses a combination of k-Nearest-Neighbor and Naive Bayes techniques to build adaptive models.

Another approach is described by Adomavicius and Tuzhilin [1] in which customer profiles are created based on the selection of attributes representing less variant user information and on a set of rules discovered using an association rule mining program. These rules are further validated by experts. A method that extracts user signatures from transactional data by applying C programs automatically generated based on input specifications is described in [4].

As does LUS-MT, the work of Schonlau and Theus [19] also bases its anomaly detection on observing the various processes run. Their approach, however, generates a statistical plot of user commands' popularity, both for individual users, and for all users together. During the application phase, observed episodes are compared to these profiles, and high levels of mismatch set off alarms.

Another approach [12] employs an n-gram-based representation of activity, but rather than using processes as the basic n-gram units, this method uses command line tokens. This approach applies a similarity-based measure between known legitimate behavior and new events, but does not articulate learned user patterns.

It is worth noting that statistical systems quite easily pick up changes in a user's behavior and adapt the model to them, which can have undesirable consequences when the new data stream is generated by an imposter [22]. The presented symbolic learning approach provides an advantages in this respect, because an expert can edit a model to eliminate or modify conditions leading to false alarms.

3 Terminology

We start by defining basic concepts and terms used. An *event* is a description of an entity under consideration. In the context of LUS-MT, it is a vector of attribute values that characterizes the use of computer by a user during a specific time period. One representation is an n-gram, a list of values of an attribute characterizing user behavior at n consecutive time instances. An extension of the n-gram that we developed in this work is an $n \times k$-gram, or, briefly, a *multigram*, which is a list of values of k attributes characterizing user states over n consecutive time instances. In LUS-MT, datastreams are represented by multigrams.

A *session* is a sequence of events characterizing a user's interaction with the computer from logon to logoff. An *episode* is a sequence of events extracted from one or more sessions; it may contain just a few events, or all of the events in the sessions. A *pattern* is a frequently occurring regularity in data, characterized by a *pattern description*, which is an expression in a knowledge representation system. Such an expression can be in the form of a set of decision rules, a decision tree, a neural network, a Bayesian net, etc. In this paper, pattern descriptions are in the form of *multistate templates* that are derived from attributional rules [14] learned from datastreams characterizing the interaction between users and the computer.

4 LUS Methodology

The presented LUS-MT method is a special case of the general LUS methodology for intrusion detection. Different LUS methods use different model representations, and thus require different programs for model learning and testing, but the basic steps outlined below are common for all LUS methods. As do other intellectual processes, the process described below typically requires several iterations, rather than a single sequential execution of steps. Depending on the results of a particular step, one may go back to a previous step, for example, to re-discretize attributes or relearn rules when the current models are not satisfactory.

In order to understand the LUS methodology, assume that raw datastreams characterizing interactions of a given set of users with the computer have been extracted. The basic steps of the LUS methodology are:

1. Choose the user model. In this paper we assume that the multistate template model has been selected.

2. Preprocess the datastream and transform it into a master target dataset. In this step, a *preprocessed dataset* is created from the raw datastreams that are to be used for model learning and testing. This initial step involves selecting the set of users for whom models will be built, selecting sufficient subsets of data for model learning and testing, selecting the attributes to be used, and possibly constructing new attributes from those in the raw datastreams and metadata.

The preprocessed datastreams are transformed into a *master target dataset* appropriate for learning the user models. In LUS-MT, this is a set of $n \times k$-grams labeled by user identifier and episode number. The subsequent steps will modify this dataset in order to generate a final target dataset for learning and testing.

3. Discretize numeric attributes. All numeric attributes are discretized into a small number of intervals. Such a discretization is done in two steps: 1) selecting a set of candidate discretization schemas, and 2) selecting the best among the candidate schemas. The first step defines discretization thresholds both manually and automatically. The second step evaluates these discretizations and selects the best for each attribute.

4. Select the most relevant attributes. The relevance of attributes in the target data is evaluated according to some measure (e.g., information gain), and the most relevant ones are retained for training. Attributes may be selected for all users, or separately for each user.

5. Determine the target data size. This step helps to suggest the amount of data to be used, utilizing a *similarity measure* that roughly characterizes the similarity between two sets of events. The consistency of each user's behavior in the master target dataset is determined by splitting each user's master data into subsets and determining the similarity between them. If the similarity is low, it may not be possible to learn a predictive model that will reliably classify behavior. Similarity between different users' data is also

measured, since such high similarity may lead to poor performance of some user models. By incrementally expanding the subsets used for measuring the similarity, it can be determined (by seeing how the performance changes) how large the target dataset needs to be in order to achieve a satisfactory user model.
6. Select training and testing episodes. In this step, training and testing episodes are defined and extracted from the target dataset. This step should be performed so as to ensure that there is a sufficient amount of data for each model to be learned and tested, but not more than is necessary to achieve satisfactory results.
7. Select the most relevant training data. This step seeks events from the training episodes that are most characteristic of each user, and thus most relevant to learning reliable models, using measures of *event significance* that are functions of the frequency of an event in the datastream of the user being modeled, and its infrequency in the datastreams of other users.
8. Learn models from training dataset. This step applies a learning program to the training dataset in order to induce user models. The best model for each user is sought by modifying learning program parameters. Model quality is characterized by complexity, consistency on the training data, and performance on the testing set in the next step.
9. Test and apply learned models. This step tests user models by matching testing data against the learned models. In LUS-MT, such matching is done by two interconnected modules, one that computes degrees of match between each model and the individual multigrams in the testing data [18, 26], and one that combines these degrees over the events in each testing episode (typically, each such episode corresponds to one session).

Below we describe how these steps were applied in LUS-MT, and show exemplary experiments and results. Further details are presented in [15].

5 Data Description and Preparation

Datastreams recording user behaviors were collected from computers running Windows, based on information stored in the system process table. To generate the datastream records, the process table was polled for changes roughly every 0.5 second. Records of processes primarily controlled by the operating system were filtered out. The entire set of datastreams represents 1292 sessions of 26 users. The experiments described in this paper were concerned with learning models for the ten users with the highest numbers of sessions recorded.

Each session's data consists of two types of records: (1) *window records* indicating a new or killed active program window, or a change to the active program window's title, and (2) *activity records* reporting process activity. The raw data consisted of eight attributes, three of which (name of current process, time since start of session, and name of active window) were used, and two of which (process status, process CPU time) were activity record attributes that were incorporated into derived attributes. Only events corresponding to window records were used in the presented experiments. Activity records were not used directly for model learning because their granularity was found to be too fine to yield useful patterns. Derived attributes included: the number of words in the active window title, the amount of time that had elapsed in the active window, the number of windows opened, the number of activity records logged from the current window, etc.

The input datastreams were transformed into a form needed for model learning and testing by a software tool that uses the Linux *awk* utility [15]. The output was a list of multigrams corresponding to both window and activity input records. In the master target dataset, n was usually set to 4 and k was 31. Continuous attributes were discretized using two manual (based on human assesment of distributions of values of attributes) and up to seven automatic (generated by the *ChiMerge* algorithm [11]) discretization schemes. For each attribute, the discretization scheme with the fewest intervals and with the highest value of the *Promise* [26] evaluation measure was selected. Attribute selection used both *Gain Ratio* and *Promise* criteria. For each criterion, a list of the top attributes was created, and the attributes that appeared in the most lists were selected. For each modeled user, the first ten user sessions were used for model learning and the next five for model testing, a choice of experiments made to satisfy our sponsor's guidelines. We conducted a number of data similarity experiments in order to determine a minimum target data size. For example, we measured similarity between the chronologically first and last segments of each user's master dataset, for segment sizes of 10%, 30%, 50%, 70% and 90% of the master dataset, and found a monotonically increasing similarity as the segments increased in size, came temporally closer to each other, and ultimately overlapped [15]. These results suggested necessary training data sizes in order to create relatively stable models.

To characterize consistency of a user's behavior and differences in the behavior of two users, we use a similarity measure called *Combined Similarity* (COS), which is the product of *Forward* (FS) and *Backward Similarity* (BS) measures. Let D1 and D2 be two datasets characterizing behavior of a single user during non-overlapping time intervals, or behavior of two users during any two time intervals. The forward similarity, FS(D1,D2) is the fraction of the events in D1 that match (over specified attributes) events in D2, and the backward similarity, BS(D1,D2), is the fraction of events in D2 that match events in D1. If D1 and D2 are used as training and testing sets, respectively, for a given user, a low FS indicates that D1 contains events that should be filtered out, and a low BS indicates that models built from D1 will likely perform poorly on D2, because its behavior is very different from the model's basis. COS determined for data from one user is called *self-similarity*, and from datastreams of different users is called *cross-similarity.*

For example, Figure 1 shows self- and cross- similarities for Users 19 (left) and 25 (right). User 19 had the lowest, and user 25 had the highest, combined self-similarity among all users. In this figure, three types of bars represent the cross-similarities if the x-axis label differs from the target user name, and self-similarities otherwise. The results indicate that for User 19, more data may be required to build a satisfactory user model [15], and that a significant number of User 19's events that appear also in other users' data should be filtered out.

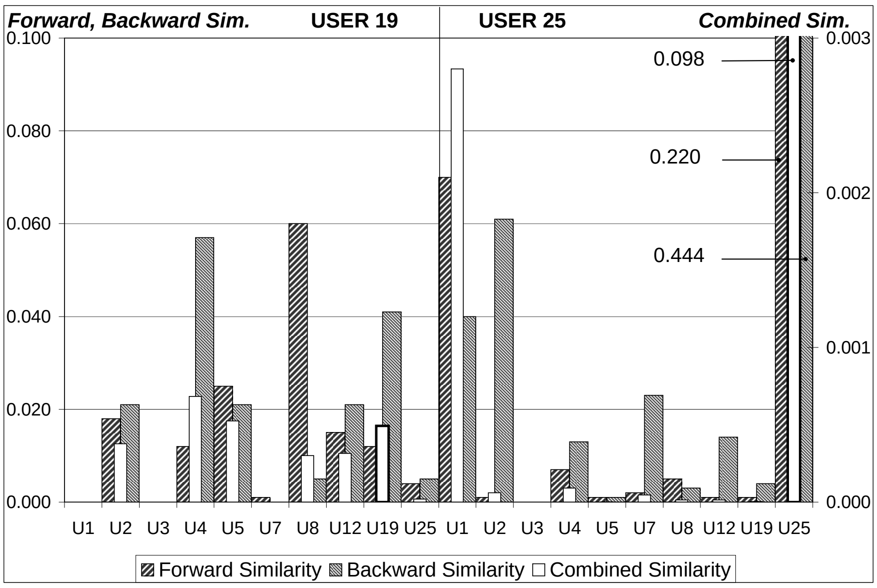

Fig. 1. Self- and cross-similarities for Users 19 and 25

6 Learning Multistate Template Models

To learn user models from sets of multigrams, we employed the rule learning program, AQ21 [26]. Given positive and negative examples of a concept to be learned, the program creates general and optimized attributional rules that describe (completely or approximately) positive examples of the concept and none of the negative examples. Positive examples characterize the behavior of the user whose model is being learned, and negative examples characterize the behavior of other users. The negative examples thus provide a contrast set, acting as constraints on the scope of generalization of the description of a user's behavior.

The program optimizes rules according to a user-defined multi-criterion rule quality measure tailored to the given learning problem. In our experiments, the measure minimized the number of rules and conditions in the models. Figure 2 shows an example of a template generated from a learned rule. A template is a list of sequences of n value sets associated with a single attribute, where the value sets represent the attribute at n consecutive time instances. In this experiment n was set to 4, and k to 6, that is, 4×6-grams were used to represent datastreams.

The first sequence in the template consists of four conditions specifying values of the *process_name* attribute in consecutive time instances. The first one states legal processes in the first time instance. Each value set is annotated by two numbers, representing the absolute positive and negative support for the given condition. The number 1339, called *absolute positive support*, is the number of multigrams in the User 2 training data that satisfy this time period constraint, and the number 5192, called *absolute negative support*, is the number of negative events that satisfy it. The second condition states possible processes in the second time instance. The interpretation of the remaining conditions follows the same pattern. If the value set is defined only for one time instance, as in the case of attributes such as "prot_word_chars", only that value set is shown in double angle brackets, and a number in parentheses denotes the position of this value set in the attribute's n-gram. In this case, the condition is in the second time instance.

The numbers p and n at the end of the template respectively denote the absolute positive and negative support of the whole pattern. Thus, this one template describes p=39 *positive events* and n=0 *negative events* in the training set for User 2. As shown in Figure 2, this template sets conditions on the values of 6 input attributes out of 31; thus, other attributes do not have to be measured to match it.

```
[ process_name=<csrss,explorer,msimn,netscape,outlook,powerpnt,rundll32,wordpad:1339,5192;
        explorer, msimn, netscape, notepad, rundll32,services, wordpad : 893, 2342;
        explorer, msimn, perfmon, rundll32, wordpad : 373, 657;
        explorer, msimn, netscape, perfmon, rundll32, wordpad : 876, 2392 > ]
[ delta_time_new_window = << 0..10500 : 1578, 7410 >> (1) ]
[ prot_words_chars = << 0..24 : 1487, 6003 >> (2) ]
[ proc_count_in_win_logfun = < -inf..0 : 325, 1322;
                               0..2.01267 : 1213, 5092;
                               0..2.01267 : 1214, 5117;
                               -inf..0 : 339, 1425 > ]
[ win_opened = << 16..inf : 276, 306 >> (4) ]
[ win_title_prot_words = << 0..3 : 1520, 6894  >> (1) ]
p = 39, n = 0
```

Fig. 2. A multistate template describing User 2

7 Testing Multistate Template Models

To test the developed user models, we employed the testing module of the learning program to match learned rulesets with episodes of user activity. Given an episode, the testing module generates a classification of the episode with associated degrees of match for each user profile. To generate those degrees of match, the module applies a three-step process:

1) Generate a degree of match between each event and each template in the user profiles. A user's model's conditions can be matched strictly, or flexibly, in which case a degree of match is computed. To calculate the aggregate degree of match, possible operators include minimum, product, average, etc. [26].

2) Generate a degree of match between each event in the episode and each user profile as a whole by aggregating the degrees of match generated in (1) between the event and the profile's individual templates. Methods such as maximum, probabilistic sum, average etc. can be used for the aggregation [26].

3) Generate a degree of match between the episode and each profile by averaging the degrees of match generated in (2).

Once a degree of match between the episode and each user profile is calculated, we classify based on *threshold* and *tolerance* parameters. All profiles returning degrees of match both above the threshold, and within the tolerance of the highest degree of match attained are returned as possible classifications.

8 Initial Experimental Results

In the target data, 94 episodes were used for learning user models, and 47 episodes were used for testing the models. In the experiment design, 100 and 50 training and testing episodes, respectively, were initially specified, but nine of those episodes were too short to generate multigrams. In these experiments, the size of the training sessions varied between 1 and 495 events, while the number of testing events per session ranged from 1 to 393. The total number of events in individual users' data varied between 51 and 1992 for training data, and between 6 and 703 for testing data. In total there were 9041 training, and 4139 testing events.

Experiments learned different types of rules by our changing the program parameters. For instance, some experiments involved learning user models using *characteristic rules* (highly specific descriptions of user activities) and others involved learning user models using maximally simple rules. In both cases, we experienced high predictive accuracy for users who showed higher self-similarity than cross-similarity, and erratic predictive accuracy for other users. Details are presented in [15]. Limited space does not allow us to discuss in greater detail the issue of training and testing dataset similarity, and its influence on the performance of user models, but these will be addressed in a subsequent report.

These experiments brought some surprises. For example, varying the window size (n) of the $n \times k$ multigrams derived from the data had little effect on predictive accuracy, as if users could be modeled nearly as well by characterizing single state multigrams (n=1) as by characterizing multistate multigrams. These results may have been due to an inappropriate temporal granularity of the datastreams. If this is the case, it opens the possibility of improving models by optimizing the time granularity of the datastreams. Further research will investigate this phenomenon.

In addition to predictive accuracy, another very important IDS evaluation criterion is false alarm rate. To deal with this concern, our testing method allows for flexibility in specifying conditions under which an alarm should be issued. For example, if the target user is not the user with the highest degree of match, it does not necessarily indicate an alarm. An alarm can be issued based on specified parameters, such as if the target user is not in the group of the users with the top 4 degrees of match, or his/her degree of match is too far from the highest degree. By setting a high threshold, we can forego issuing an alarm if there is no clear winner among candidate models. In this case the system outputs "I don't know", which is not an alarm, but indicates the need for further observation.

9 Conclusion and Future Research

This paper presented the LUS-MT method and preliminary results of its testing on user data. The most significant results are the multistate templates that concisely represent user activity patterns, and a method for efficiently applying them to user datastreams. To enhance the representation of user states at specific time instances, we introduced multigrams ($n \times k$-grams) for characterizing each state.

During the course of the experiments, we found that the behavior of some users was radically different in training and testing sessions. This situation precluded the system from learning highly predictive models from these users' training data. In cases where users' behavior was relatively consistent, predictive accuracy was 100%. Taking into consideration this limitation of the training data, the results show that LUS-MT is capable of learning high quality and efficient user models. The models are expressed in forms that make them relatively easy to interpret and understand. The latter feature makes it possible for experts to manually adjust the models by modifying or removing spurious conditions, or adding new ones.

In addition, user models can be applied to datastreams in parallel, and thus the model matching time does not depend on the number of user models. LUS-MT can therefore be potentially employed for developing an intrusion detection system for a large number of users. Achieving high recognition of individual users in an intrusion detection system is predicated upon sufficient consistency in the given user's behavior, and its sufficient difference from the behavior of other users. What constitutes sufficient consistency and sufficient difference depends on the required degree of certainty of recognition. If the user's behavior changes frequently, user models must be updated frequently as well.

Obtained experimental results have opened several topics for further research. Studies of MT models using different data, possibly with information about usage of input devices such as keyboard or mouse, are needed to comprehensively evaluate this model. Another topic is to investigate other types of user models within LUS, and compare them with the multistate template model. Such models could include, for instance, a Bayesian model, and a combination of these models.

Other important topics for research are to study different methods for filtering. In a large group of users, there will likely be users who during some periods will behave very similarly to others. Consequently, events extracted from the datastreams of these users during such a period will be very similar, and discrimination among them will not be possible. Adequate training event filtering may allow user models that have a higher predictive accuracy and are also simpler.

Related problems concern methods of representation space optimization. One method includes in training datastreams only the most relevant attributes for discriminating among users. Another method optimizes discretization levels of continuous attributes, in search of optimal precision. The third and most difficult method searches for new attributes that better discriminate among the users. An important topic is to determine the needed sizes of the training datastreams for different users, and of the application datastreams needed to confirm legitimate use or detect possible illegitimate use. Other major open topics include the development of methods for incremental learning of the user models, and for learning and applying ensembles of user models.

Acknowledgments

This research was supported in part by the UMCB/LUCITE #32 grant, and in part by the National Science Foundation under Grants No. IIS-0097476 and IIS-9906858. The opinions expressed in this report are those of the authors, not necessarily the opinion of the UMBC/LUCITE program or the National Science Foundation.

References

1. Adomavicius, G. and Tuzhilin, A., "Building Customer Profiles in Personalization Applications Using Data Mining Methods," IEEE Computer, 34(2), 2001.
2. Bace, R.G.,. Intrusion Detection, Indianapolis: Macmillan Technical Publishing, 2000.

3. Billsus, D. and Pazzani, M., "User Modeling For Adaptive News Access," User Modeling and User-Adapted Interaction, 10(2-3):147-180, 2000.
4. Cortes, C., Fisher, K., Pregibon, D., Rogers, A. and Smith, F., "Hancock: A Language For Extracting Signatures From Data Streams," Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.
5. Eskin, E., Arnold, A., Prerau, M., Portnoy, L. and Stolfo, S., "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data," in D. Barbara, & S. Jajodia (Eds.), Applications of Data Mining in Computer Security, Kluwer, 2002, pp. 77-102.
6. Goldring, T., "Recent Experiences with User Profiling for Windows NT," Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, 2002.
7. Goldring, T., Shostak, J., Tessman, B. and Degenhardt, S., "User Profiling (Extended Abstract)," NSA unclassified internal report, 2000.
8. Hofmeyr, S., Forrest, S. and Somayaji, A., "Intrusion Detection using Sequences of System Calls," Journal of Computer Security, 6, 1998, pp. 151-180.
9. Javitz H. S. and Valdes, A., "The SRI IDES Statistical Anomaly Detector," Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1991.
10. Julisch, K. and Dacier M., "Mining Intrusion Detection Alarms for Actionable Knowledge," Proc. 8th Intl. Conf. on Knowledge Discovery and Data Mining, July 2002.
11. Kerber, R., "Chimerge: Discretization for Numeric Attributes," Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), 1992, pp. 123-128.
12. Lane, T. and Brodley, C.E., "Temporal Sequence Learning and Data Reduction for Anomaly Detection," ACM Trans. on Information and Syst. Security, 2, 1999, pp. 295-331.
13. McHugh, J., "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," ACM Trans. on Information & Systems Security, 3, November 2000, pp. 262-294.
14. Michalski, R.S., "Attributional Calculus: A Logic and Representation Language for Natural Induction," Reports of the Machine Learning and Inference Laboratory, MLI 04-2, George Mason University, 2004.
15. Michalski, R.S., Kaufman K., Pietrzykowski, J., Sniezynski, B. and Wojtusiak, J., "Learning User Models for Computer Intrusion Detection: Preliminary Results from Natural Induction Approach," Reports of the Machine Learning and Inference Laboratory, MLI 05-3, George Mason University, 2005.
16. 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, 2002.
17. Novak, J., Stark, V. and Heinbuch, D., "Zombie Scan," Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, 2002.
18. Reinke, R., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System," M.S. Thesis, Reports of the Intelligent Systems Group, ISG 84-5, UIUCDCS-F-84-921, University of Illinois Dept. of Computer Science, Urbana, 1984.

19. Schonlau, M. and Theus, M., "Detecting Masquerades in Intrusion Detection based on Unpopular Commands," Information Processing Letters, 76, 2000, pp. 33-38.
20. Scott, S., "A Bayesian Paradigm for Designing Intrusion Detection Systems," Computational Statistics and Data Analysis, 45, 2004, pp. 69-83.
21. Shah, K., Jonckheere, E. and Bohacek, S., "Detecting Network Attacks through Traffic Modeling," Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, 2002.
22. Shavlik, J. and Shavlik, M., "Selection, Combination, and Evaluation of Effective Software Sensors for Detecting Abnormal Computer Usage," Proc. of the 10th Intl. Conference on Knowledge Discovery and Data Mining, Seattle, WA, 2004, pp. 276-285.
23. 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, 2002.
24. Valdes, A., "Profile Based Intrusion Detection: Lessons Learned, New Directions," Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, 2002.
25. Valdes, A. and Skinner, K., "Adaptive, Model-based Monitoring for Cyber Attack Detection," in H. Debar, L. Me and F. Wu (Eds.), Lecture Notes in Computer Science #1907 (from Recent Advances in Intrusion Detection, RAID-2000), Springer-Verlag, 2000.
26. Wojtusiak, J., "AQ21 User's Guide," Reports of the Machine Learning and Inference Laboratory, MLI 04-3, George Mason University, 2004.