International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 5, July - 2012

INTRUSION DETECTION USING DATAMINING

[1]R.Sateesh Kumar [2]M.Sunitha Reddy

Abstract

In this paper we discuss our research in developing general and systematic methods for intrusion detection. The key ideas are to use data mining techniques to discover consistent and useful patterns of system features that describe program and user behavior, and use the set of relevant system features to compute (inductively learned) classifiers that can recognize anomalies and known intrusions. Using experiments on the sendmail *system call data , we demonstrate that we can construct concise and accurate classifiers to detect anomalies. We provide an overview on two general data mining algorithms that we have implemented: the association rules algorithm and the frequent episodes algorithm. These algorithms can be used to compute the intra- and inter- audit record patterns, which are essential in describing program or user behavior. The discovered patterns can guide the audit data gathering process and facilitate feature selection. To meet the challenges of both efficient learning (mining) and real-time detection, we propose an agent-based architecture for intrusion detection systems where the learning agents continuously compute and provide the updated (detection) models to the detection agents.*

1 Introduction

As network-based computer systems play increasingly vital roles in modern society, they have become the targets of our enemies and criminals. Therefore, we need to find the best ways possible to protect our systems.

The security of a computer system is compromised when an intrusion takes place. An intrusion can be defined [8] as ``any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource". Intrusion prevention errors, and information protection (e.g., encryption) have been used to protect computer systems as a first line of defense. Intrusion prevention alone is not sufficient because as systems become ever more complex, there are always exploitable weakness in the systems due to design and programming errors, or various ``socially engineered" penetration techniques. For example, after it was first reported many years ago, exploitable ``buffer overflow" still exists in some recent system software due to programming errors. The policies that balance convenience versus strict control of a system and information access also make it impossible for an operational system to be completely secure.

Intrusion detection is therefore needed as another wall to protect computer systems. The elements central to intrusion detection are: *resources* to be protected in a target system, i.e., user accounts, file systems, system kernels, etc; *models* that characterize the ``normal" or ``legitimate" behavior of these resources; *techniques* that compare the actual system activities with the established models, and identify those that are ``abnormal" or ``intrusive".

Many researchers have proposed and implemented different models which define different measures of system behavior, with an ad hoc presumption that normalcy and anomaly (or illegitimacy) will be accurately manifested in the chosen set of system features that are modeled and measured. Intrusion detection techniques can be categorized into *misuse detection*, which uses patterns of well-known attacks or weak spots of the system to identify intrusions; and *anomaly detection*, which tries to determine whether deviation from the established normal usage patterns can be flagged as intrusions.

Misuse detection systems, for example [12] and STAT [9], encode and match the sequence of ``signature actions" (e.g., change the ownership of a file) of known intrusion scenarios. The main shortcomings of such systems are: known intrusion patterns have to be hand-coded into the system; they are unable to detect any future (unknown) intrusions that have no matched patterns stored in the system.

Anomaly detection (sub)systems, such as IDES , establish normal usage patterns (profiles) using statistical measures on system features, for example, the CPU and I/O activities by a particular user or program. The main difficulties of these systems are: intuition and experience is relied upon in selecting the system features, which can vary greatly among

different computing environments; some intrusions can only be detected by studying the sequential interrelation between events because each event alone may fit the profiles.

Our research aims to eliminate, as much as possible, the manual and ad-hoc elements from the process of building an intrusion detection system. We take a data-centric point of view and consider intrusion detection as a data analysis process. Anomaly detection is about finding the normal usage patterns from the audit data, whereas misuse detection is about encoding and matching the intrusion patterns using the audit data. The central theme of our approach is to apply data mining techniques to intrusion detection. Data mining generally refers to the process of (automatically) extracting models from large stores of data [6]. The recent rapid development in data mining has made available a wide variety of algorithms, drawn from the fields of statistics, pattern recognition, machine learning, and database. Several types of algorithms are particularly relevant to our research:

Classification:
maps a data item into one of several pre-defined categories. These algorithms normally output ``classifiers", for example, in the form of decision trees or rules. An ideal application in intrusion detection will be to gather sufficient ``normal" and ``abnormal" audit data for a user or a program, then apply a classification algorithm to learn a classifier that will determine (future) audit data as belonging to the normal class or the abnormal class;

Link analysis:
determines relations between fields in the database. Finding out the correlations in audit data will provide insight for selecting the right set of system features for intrusion detection;

Sequence analysis:
models sequential patterns. These algorithms can help us understand what (time-based) sequence of audit events are frequently encountered together. These frequent event patterns are important elements of the behavior profile of a user or program.

We are developing a systematic framework for designing, developing and evaluating intrusion detection systems. Specifically, the framework consists of a set of environment-independent guidelines and programs that can assist a system administrator or security officer to

- select appropriate system features from audit data to build models for intrusion detection;
- architect a hierarchical detector system from component detectors;
- update and deploy new detection systems as needed.

The key advantage of our approach is that it can automatically generate concise and accurate detection models from large amount of audit data. The methodology itself is general and mechanical, and therefore can be used to build intrusion detection systems for a wide variety of computing environments.

The rest of the paper is organized as follows: Section 2 describes our experiments in building classification models for *sendmail* and network traffic. Section 3 presents the association rules and frequent episodes algorithms that can be used to compute a set of patterns from audit data. Section 4 briefly highlights the architecture of our proposed intrusion detection system. Section 5 outlines our future research plans.

2 Building Classification Models

In this section we describe in detail our experiments in constructing classification models for anomaly detection. The first set of experiments, first reported in [15], is on the *sendmail* system call data, and the second is on the network *tcpdump* data.

2.1 Experiments on *sendmail* Data

There have been a lot of attacks on computer systems that are carried out as exploitations of the design and programming errors in privileged programs, those that can run as root. For example, a flaw in the *finger* daemon allows the attacker to use ``buffer overflow" to trick the program to execute his malicious code. Recent research efforts by Ko et al. [11] and Forrest et al. [5] attempted to build intrusion detection systems that monitor the execution of privileged programs and detect the attacks on their vulnerabilities. Forrest et al. discovered that the short sequences of system calls made by a program during its normal executions are very consistent, yet different from the sequences of its abnormal (exploited) executions as well as the executions of other programs. Therefore a database containing these normal sequences can be used as the ``self" definition of the normal behavior of a program, and as the basis to detect anomalies. Their findings motivated us to search for simple and accurate intrusion detection models.

Stephanie Forrest provided us with a set of traces of the *sendmail* program used in her experiments [5].

We applied machine learning techniques to produce classifiers that can distinguish the exploits from the normal runs.

2.1.1 The *sendmail* System Call Traces

The procedure of generating the *sendmail* traces were detailed in [5]. Briefly, each file of the trace data has two columns of integers, the first is the process ids and the second is the system call ``numbers''. These numbers are indices into a lookup table of system call names. For example, the number ``5'' represents system call *open*. The set of traces include:

Normal traces:

a trace of the *sendmail* daemon and a concatenation of several invocations of the *sendmail* program;

Abnormal traces:

3 traces of the *sscp* (*sunsendmailcp*) attacks, 2 traces of the *syslog-remote* attacks, 2 traces of the *syslog-local* attacks, 2 traces of the *decode* attacks, 1 trace of the *sm5x* attack and 1 trace of the *sm565a* attack. These are the traces of (various kinds of) abnormal runs of the *sendmail* program.

2.1.2 Learning to Classify System Call Sequences

In order for a machine learning program to learn the classification models of the ``normal'' and ``abnormal'' system call sequences, we need to supply it with a set of training data containing pre-labeled ``normal'' and ``abnormal'' sequences. We use a sliding window to scan the normal traces and create a list of unique sequences of system calls. We call this list the ``normal'' list. Next, we scan each of the intrusion traces. For each sequence of system calls, we first look it up in the normal list. If an exact match can be found then the sequence is labeled as ``normal''. Otherwise it is labeled as ``abnormal'' (note that the data gathering process described in [5] ensured that the normal traces include nearly all possible ``normal'' short sequences of system calls, as new runs of sendmail failed to generate new sequences). Needless to say all sequences in the normal traces are labeled as ``normal''. See Table 1 for an example of the labeled sequences. It should be noted that an intrusion trace contains many normal sequences in addition to the abnormal sequences since the illegal activities only occur in some places within a trace.

Table 1: Pre-labeled System Call Sequences of Length 7

System Call Sequences (length 7)	Class Labels
4 2 66 66 4 138 66	``normal''
...	...
5 5 5 4 59 105 104	``abnormal''
...	...

We applied RIPPER [3], a rule learning program, to our training data. The following learning tasks were formulated to induce the rule sets for normal and abnormal system call sequences:

- Each record has n positional attributes, p1, p2, ..., pn, one for each of the system calls in a sequence of length n; plus a class label, ``normal'' or ``abnormal''
- The training data is composed of normal sequences taken from 80% of the normal traces, plus the abnormal sequences from 2 traces of the sscp attacks, 1 trace of the *syslog-local* attack, and 1 trace of the *syslog-remote* attack
- The testing data includes both normal and abnormal traces not used in the training data.

RIPPER outputs a set of if-then rules for the ``minority'' classes, and a default ``true'' rule for the remaining class. The following exemplar RIPPER rules were generated from the system call data:

> normal:- p2=104, p7=112. [meaning: if p2 is 104 (vtimes) and p7 is 112 (vtrace) then the sequence is ``normal'']
>
> normal:- p6=19, p7=105. [meaning: if p6 is 19 (lseek) and p7 is 105 (sigvec) then the sequence is ``normal'']
>
> abnormal:- true. [meaning: if none of the above, the sequence is ``abnormal'']

These RIPPER rules can be used to predict whether a sequence is ``abnormal'' or ``normal''. But what the intrusion detection system needs to know is whether the trace being analyzed is an intrusion or not. We

use the following post-processing scheme to detect whether a given trace is an intrusion based on the RIPPER predictions of its constituent sequences:

1. Use a sliding window of length 2L+1, e.g., 7, 9, 11, 13, etc., and a sliding (shift) step of L, to scan the predictions made by the RIPPER rules on system call sequences.
2. For each of the (length 2L+1) regions of RIPPER predictions generated in Step 1, if more than L predictions are ``abnormal'' then the current region of predictions is an ``abnormal'' region. (Note that L is an input parameter).
3. If the percentage of abnormal regions is above a threshold value, say 2%, then the trace is an intrusion.

This scheme is an attempt to filter out the spurious prediction errors. The intuition behind this scheme is that when an intrusion actually occurs, the majority of adjacent system call sequences are abnormal; whereas the prediction errors tend to be isolated and sparse. In [5], the percentage of the mismatched sequences (out of the total number of matches (lookups) performed for the trace) is used to distinguish normal from abnormal. The ``mismatched'' sequences are the abnormal sequences in our context. Our scheme is different in that we look for abnormal regions that contain more abnormal sequences than the normal ones, and calculate the percentage of abnormal regions (out of the total number of regions). Our scheme is more sensitive to the temporal information, and is less sensitive to noise (errors).

RIPPER only outputs rules for the ``minority'' class. For example, in our experiments, if the training data has fewer abnormal sequences than the normal ones, the output RIPPER rules can be used to identify abnormal sequences, and the default (everything else) prediction is normal. We conjectured that a set of specific rules for normal sequences can be used as the ``identity'' of a program, and thus can be used to detect any known and unknown intrusions (anomaly intrusion detection). Whereas having only the rules for abnormal sequences only gives us the capability to identify known intrusions (misuse intrusion detection).

Table: Comparing Detection of Anomalies. The column [5] is the percentage of the abnormal sequences of the traces. Columns A, B, C, and D are the percentages of abnormal regions (as measured by the post-processing scheme) of the traces. sendmail is the 20% normal traces not used in the training data. Traces in bold were included in the training data, the other traces were used as testing data only.

	% abn.	% abn. in experiment			
Traces	[5]	A	B	C	D
sscp-1	5.2	41.9	32.2	40.0	33.1
sscp-2	5.2	40.4	30.4	37.6	33.3
sscp-3	5.2	40.4	30.4	37.6	33.3
syslog-r-1	5.1	30.8	21.2	30.3	21.9
syslog-r-2	1.7	27.1	15.6	26.8	16.5
syslog-l-1	4.0	16.7	11.1	17.0	13.0
syslog-l-2	5.3	19.9	15.9	19.8	15.9
decode-1	0.3	4.7	2.1	3.1	2.1
decode-2	0.3	4.4	2.0	2.5	2.2
sm565a	0.6	11.7	8.0	1.1	1.0
sm5x	2.7	17.7	6.5	5.0	3.0
sendmail	0	1.0	0.1	0.2	0.3

We compare the results of the following experiments that have different distributions of abnormal versus normal sequences in the training data:

Experiment A:
46% normal and 54% abnormal, sequence length is 11;
Experiment B:
46% normal and 54% abnormal, sequence length is 7;
Experiment C:
46% abnormal and 54% normal, sequence length is 11;
Experiment D:
46% abnormal and 54% normal, sequence length is 7.

Table 2 shows the results of using the classifiers from these experiments to analyze the traces. We report

here the percentage of abnormal regions (as measured by our post-processing scheme) of each trace, and compare our results with Forrest et al., as reported in [5]. From Table 2, we can see that in general, intrusion traces generate much larger percentages of abnormal regions than the normal traces. We call these measured percentages the ``scores'' of the traces. In order to establish a threshold score for identifying intrusion traces, it is desirable that there is a sufficiently large gap between the scores of the normal sendmail traces and the low-end scores of the intrusion traces. Comparing experiments that used the same sequence length, we observe that such a gap in A, 3.4, is larger than the gap in C, 0.9; and 1.9 in B is larger than 0.7 in D. The RIPPER rules from experiments A and B describe the patterns of the normal sequences. Here the results show that these rules can be used to identify the intrusion traces, including those not seen in the training data, namely, the *decode* traces, the *sm565a* and *sm5x* traces. This confirms our conjecture that rules for normal patterns can be used for anomaly detection. The RIPPER rules from experiments C and D specify the patterns of abnormal sequences in the intrusion traces included in the training data. The results indicate that these rules are very capable of detecting the intrusion traces of the ``known'' types (those seen in the training data), namely, the *sscp-3* trace, the *syslog-remote-2* trace and the *syslog-local-2* trace. But comparing with the rules from A and B, the rules in C and D perform poorly on intrusion traces of ``unknown'' types. This confirms our conjecture that rules for abnormal patterns are good for misuse intrusion detection, but may not be as effective in detecting future (``unknown'') intrusions.

The results from Forrest et al. showed that their method required a very low threshold in order to correctly detect the decode and sm565a intrusions. While the results here show that our approach generated much stronger ``signals'' of anomalies from the intrusion traces, it should be noted that their method used all of the normal traces but not any of the intrusion traces in training.

2.1.3 Learning to Predict System Calls

Unlike the experiments in Section 2.1.2 which required abnormal traces in the training data, here we wanted to study how to compute an anomaly detector given just the normal traces. We conducted experiments to learn the (normal) correlation among system calls: the nth system calls or the middle system calls in (normal) sequences of length n.

The learning tasks were formulated as follows:

- Each record has n-1 positional attributes, p1, p2, ..., pn-1, each being a system call; plus a class label, the system call of the nth position or the middle position
- The training data is composed of (normal) sequences taken from 80% of the normal sendmail traces
- The testing data is the traces not included in the training data, namely, the remaining 20% of the normal sendmail traces and all the intrusion traces.

RIPPER outputs rules in the following form:

> 38 :- p3=40, p4=4. [meaning: if p3 is 40 (lstat) and p4 is 4 (write), then the 7th system call is 38 (stat).]
>
> ...
>
> 5:- true. [meaning: if none of the above, then the 7th system calls is 5 (open).]

Each of these RIPPER rules has some ``confidence'' information: the number of matched examples (records that conform to the rule) and the number of unmatched examples (records that are in conflict with the rule) in the training data. For example, the rule for ``38 (stat)'' covers 12 matched examples and 0 unmatched examples. We measure the confidence value of a rule as the number of matched examples divided by the sum of matched and unmatched examples. These rules can be used to analyze a trace by examining each sequence of the trace. If a violation occurs (the actual system call is not the same as predicted by the rule), the ``score'' of the trace is incremented by 100 times the confidence of the violated rule. For example, if a sequence in the trace has p3=40 and p4=4, but p7=44 instead of 38, the total score of the trace is incremented by 100 since the confidence value of this violated rule is 1. The averaged score (by the total number of sequences) of the trace is then used to decide whether an intrusion has occurred.

Table 3: Detecting Anomalies using Predicted System Calls. Columns A, B, C, and D are the averaged scores of violations of the traces. sendmail is the 20% normal traces not used in the training data.

None of the intrusion traces was used in training.

	averaged score of violations			
Traces	Exp. A	Exp. B	Exp. C	Exp. D
sscp-1	24.1	13.5	14.3	24.7
sscp-2	23.5	13.6	13.9	24.4
sscp-3	23.5	13.6	13.9	24.4
syslog-r-1	19.3	11.5	13.9	24.0
syslog-r-2	15.9	8.4	10.9	23.0
syslog-l-1	13.4	6.1	7.2	19.0
syslog-l-2	15.2	8.0	9.0	20.2
decode-1	9.4	3.9	2.4	11.3
decode-2	9.6	4.2	2.8	11.5
sm565a	14.4	8.1	9.4	20.6
sm5x	17.2	8.2	10.1	18.0
sendmail	5.7	0.6	1.2	12.6

Table 3 shows the results of the following experiments:

Experiment A:
predict the 11th system call;
Experiment B:
predict the middle system call in a sequence of length 7;
Experiment C:
predict the middle system call in a sequence of length 11;
Experiment D:
predict the 7th system call.

We can see from Table 3 that the RIPPER rules from experiments A and B are effective because the gap between the score of normal sendmail and the low-end scores of intrusion traces, 3.9, and 3.3 respectively, are large enough. However, the rules from C and D perform poorly. Since C predicts the middle system call of a sequence of length 11 and D predicts the 7th system call, we reason that the training data (the normal traces) has no stable patterns for the 6th or 7th position in system call sequences.

2.1.4 Discussion

Our experiments showed that the normal behavior of a program execution can be established and used to detect its anomalous usage. This confirms the results of other related work in anomaly detection. The weakness of the model in [5] may be that the recorded (rote learned) normal sequence database may be too specific as it contains about ~1,500 entries. Here we show that a machine learning program, RIPPER, was able to generalize the system call sequence information, from 80% of the normal sequences, to a set of concise and accurate rules (the rule sets have 200 to 280 rules, and each rule has 2 or 3 attribute tests). We demonstrated that these rules were able to identify unseen intrusion traces as well as normal traces.

We need to search for a more predictive classification model so that the anomaly detector has higher confidence in flagging intrusions. Improvement in accuracy can come from adding more features, rather than just the system calls, into the models of program execution. For example, the directories and the names of the files touched by a program can be used. In [7], it is reported that as the number of features increases from 1 to 3, the classification error rate of their network intrusion detection system decreases dramatically. Furthermore, the error rate stabilizes after the size of the feature set reaches 4, the optimal size in their experiments. Many operating systems provide auditing utilities, such as the BSM audit of Solaris, that can be configured to collect abundant information (with many features) of the activities in a host system. From the audit trails, information about a process (program) or a user can then be extracted. The challenge now is to efficiently compute accurate patterns of programs and users from the audit data.

A key assumption in using a learning algorithm for anomaly detection (and to some degree, misuse detection) is that the training data is nearly ``complete" with regard to all possible ``normal" behavior of a program or user. Otherwise, the learned detection model can not confidently classify or label an unmatched data as ``abnormal" since it can just be an unseen ``normal" data. For example, the experiments in Section 2.1.3 used 80% of ``normal" system call sequences; whereas the experiments in Section 2.1.2 actually required all ``normal" sequences in order to pre-label the ``abnormal" sequences to create the training data. During the audit data gathering process, we want to ensure that as much different normal behavior as possible is captured. We first need to have a simple and incremental (continuously learning) summary measure of an audit trail so that we can update this

measure as each new audit trail is processed, and can stop the audit process when the measure stabilizes. In Section 3, we propose to use the frequent intra- and inter- audit record patterns as the summary measure of an audit trail, and describe the algorithms to compute these patterns.

3 Mining Patterns from Audit Data

In order to construct an accurate (effective) base classifier, we need to gather a sufficient amount of training data and identify a set of meaningful features. Both of these tasks require insight into the nature of the audit data, and can be very difficult without proper tools and guidelines. In this section we describe some algorithms that can address these needs. Here we use the term ``audit data'' to refer to general data streams that have been properly processed for detection purposes. An example of such data streams is the connection record data extracted from the raw *tcpdump* output.

3.1 Association Rules

The goal of mining association rules is to derive multi-feature (attribute) correlations from a database table. A simple yet interesting commercial application of the association rules algorithm is to determine what items are often purchased together by customers, and use that information to arrange store layout. Formally, given a set of records, where each record is a set of items, an association rule is an expression X --> Y, confidence, support. X and Y are subsets of the items in a record, *support* is the percentage of records that contain X+Y, whereas *confidence* is *support(X+Y)/support(X)*. For example, an association rule from the shell command history file (which is a stream of commands and their arguments) of a user is

trn --> rec.humor; [0.3, 0.1],

which indicates that 30% of the time when the user invokes trn, he or she is reading the news in rec.humor, and reading this newsgroup accounts for 10% of the activities recorded in his or her command history file. Here 0.3 is the *confidence* and 0.1 is the support.

The motivation for applying the association rules algorithm to audit data are:

- Audit data can be formatted into a database table where each row is an audit record and each column is a field (system feature) of the audit records;
- There is evidence that program executions and user activities exhibit frequent correlations among system features. For example, one of the reasons that ``program policies'', which codify the access rights of privileged programs, are concise and capable to detect known attacks [11] is that the intended behavior of a program, e.g., read and write files from certain directories with specific permissions, is very consistent. These consistent behaviors can be captured in association rules;
- We can continuously merge the rules from a new run to the aggregate rule set (of all previous runs).

Our implementation follows the general association rules algorithm.

3.2 Frequent Episodes

While the association rules algorithm seeks to find intra- audit record patterns, the frequent episodes algorithm, can be used to discover inter- audit record patterns. A frequent episode is a set of events that occur frequently within a time window (of a specified length). The events must occur (together) in at least a specified minimum frequency, *min_fr*, sliding time window. Events in a serial episode must occur in partial order in time; whereas for a parallel episode there is no such constraint. For X and Y where X+Y is a frequent episode, X --> Y with *confidence=frequency(X+Y)/frequency(X)* and support=frequency(X+Y) is called a frequent episode rule. An example frequent serial episode rule from the log file of a department's Web site is

home, research --> theory; [0.2, 0.05], [30s]

which indicates that when the home page and the research guide are visited (in that order), in 20% of the cases the theory group's page is visited subsequently within the same 30s time window, and this sequence of visits occurs 5% of the total (the 30s) time windows in the log file (that is, approximately 5% of all the records).

We seek to apply the frequent episodes algorithm to analyze audit trails since there is evidence that the sequence information in program executions and user commands can be used to build profiles for anomaly detection [5,14].

3.3 Using the Discovered Patterns

The association rules and frequent episodes can be used to guide the audit process. We run a program many times and under different settings. For each new run, we compute its rule set (that consists of both the association rules and the frequent episodes) from the audit trail, and update the (existing) aggregate rule sets using the following *merge* process:

- For each rule in the new rule set: find a match in the aggregate rule set. A match is defined as the exact matches on both the LHS and RHS of the rules, plus *epsilon* matches (using ranges), on the support (or frequency) and confidence values
- If a match is found, increment the *match_count* of the matched rule in the aggregate rule set. Otherwise, add the new rule and initialize its *match_count* to be 1.

When the rule set stabilizes (there are no new rules added), we can stop the data gathering process since we have produced a near complete set of audit data for the normal runs. We then *prune* the rule set by eliminating the rules with low *match_count*, according to a user-defined threshold on the ratio of *match_count* over the total number of audit trails. The system builders can then use the correlation information in this final *profile* rule set to select a subset of the relevant features for the classification tasks. We plan to build a support environment to integrate the process of user selection of features, computing a classifier (according to the feature set), and presenting the performance of the classifier. Such a support system can speed up the iterative feature selection process, and help ensure the accuracy of a detection model.

We believe that the discovered patterns from (the extensively gathered) audit data can be used directly for anomaly detection. We compute a set of association rules and frequent episodes from a new audit trail, and compare it with the established profile rule set. Scoring functions can be used to evaluate the deviation scores for: missing rules with high support, violation (same antecedent but different consequent) of rules with high support and confidence, new (unseen) rules, and significant changes in support of rules.

4 Conclusion and Future Work

In this paper we proposed a systemic framework that employs data mining techniques for intrusion detection. This framework consists of classification, association rules, and frequence episodes programs, that can be used to (automatically) construct detection models. The experiments on *sendmail* system call data and network *tcpdump* data demonstrated the effectiveness of classification models in detecting anomalies. The accuracy of the detection models depends on sufficient training data and the right feature set. We suggested that the association rules and frequent episodes algorithms can be used to compute the consistent patterns from audit data. These frequent patterns form an abstract summary of an audit trail, and therefore can be used to: guide the audit data gathering process; provide help for feature selection; and discover patterns of intrusions. Preliminary experiments of using these algorithms on the *tcpdump* data showed promising results.

We are in the initial stages of our research, much remains to be done including the following tasks:

- Implement a support environment for system builders to iteratively drive the integrated process of pattern discovering, system feature selection, and construction and evaluation of detection models;
- Investigate the methods and benefits of combining multiple simple detection models. We need to use multiple audit data streams for experiments;
- Implement a prototype agent-based intrusion detection system. JAM already provides a base infrastructure;
- Evaluate our approach using extensive audit data sets, some of which is presently under construction at Rome Labs.

References

[1] D. Atkins, P. Buis, C. Hare, R. Kelley, C. Nachenberg, A. B. Nelson, P. Phillips, T. Ritchey, and W. Steen. *Internet Security Professional Reference*. New Riders Publishing, 1996.

[2] S.M.Bellovin. Security problems in the tcp/ip protocol suite. *Computer Communication Review*, 19(2):32-48, April 1989.

[3]

W.W.Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[4]

P.K.Chan and S.J.Stolfo.Toward parallel and distributed learning by meta-learning. In *AAAI Workshop in Knowledge Discovery in Databases*, pages 227-240, 1993.

[5]

S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A.Longstaff.A sense of self for unix processes.
In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120-128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[6]

U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth.
The KDD process of extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11):27-34, November 1996.

[7]

J.Frank.Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference*, October 1994.

[8]

R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical report, Computer Science Department, University of New Mexico, August 1990.

[9]

K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach.*IEEE Transactions on Software Engineering*, 21(3):181-199, March 1995.

[10]

V. Jacobson, C. Leres, and S. McCanne. tcpdump.
available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

[11]

C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring.
In *Proceedings of the 10th Annual Computer Security Applications Conference*, pages 134-144, December 1994.

[12]

S. Kumar and E. H. Spafford. A software architecture to support misuse intrusion detection.
In *Proceedings of the 18th National Information Security Conference*, pages 194-204, 1995.

[13]

J. O. Kephart, G. B. Sorkin, M. Swimmer, and S. R.White. Blueprint for a computer immune system. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1997.

[14]

T.Lane and C.E.Brodley.Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43-49. AAAI Press, July 1997.

[15]

W. Lee, S. J. Stolfo, and P. K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 50-56. AAAI Press, July 1997.

[16]

T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey.
A real-time intrusion detection expert system (IDES) - final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.