# Adaptive Intrusion Detection: a Data Mining Approach *

Wenke Lee (wenke@cs.columbia.edu) and Salvatore J. Stolfo (sal@cs.columbia.edu)

*Computer Science Department, Columbia University, 500 West 120th Street, New York, NY 10027*

Kui W. Mok† (mokk@ms.com)

*Morgan Stanley Dean Witter & Co., 750 7th Avenue, New York, NY 10019*

**Abstract.** In this paper we describe a data mining framework for constructing intrusion detection models. The first key idea is to mine system audit data for consistent and useful patterns of program and user behavior. The other is to use the set of relevant system features presented in the patterns to compute inductively learned classifiers that can recognize anomalies and known intrusions. In order for the classifiers to be effective intrusion detection models, we need to have sufficient audit data for training and also select a set of predictive system features. We propose to use the association rules and frequent episodes computed from audit data as the basis for guiding the audit data gathering and feature selection processes. We modify these two basic algorithms to use *axis attribute(s)* and *reference attribute(s)* as forms of item constraints to compute only the relevant patterns. In addition, we use an iterative *level-wise* approximate mining procedure to uncover the low frequency but important patterns. We use meta-learning as a mechanism to make intrusion detection models more effective and adaptive. We report our extensive experiments in using our framework on real-world audit data.

**Keywords:** intrusion detection, audit data, classification, association rules, frequent episodes, feature construction

* This research is supported in part by grants from DARPA (F30602-96-1-0311) and NSF (IRI-96-32225 and CDA-96-25374).
† This author's contributions were made while at Columbia University.

## 1. Introduction

As network-based computer systems play increasingly vital roles in modern society, they have become the target 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 (Heady et al., 1990) as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource", for example, illegally gaining superuser privileges, attacking and rendering a system out of service (i.e., denial-of-service), etc.

Intrusion prevention techniques, such as user authentication (e.g. using passwords or biometrics), avoiding programming 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 weaknesses in the systems due to design and programming errors. For example, there are known design flaws in TCP/IP protocols and Unix systems that have led to security holes (Bellovin, 1989; Grampp and Morris, 1984); and after it was first reported many years ago, exploitable "buffer overflow" bugs still exist in some recent system software due to programming errors. The policies that balance convenience versus strict control of a system and information access, as well as critical "insider" problems, also make it impossible for an operational system to be completely secure. Intrusion detection is therefore needed as another wall to protect computer systems.

Many intrusion detection systems have been constructed by manual and *ad hoc* means. These systems have been designed and implemented based on the system builders' knowledge of a computer system and their understanding of known intrusions. As a result, the effectiveness and adaptability of the intrusion detection systems are limited in the face of new computing environments or newly invented attack methods.

This paper discusses a systematic framework for analyzing audit data and constructing intrusion detection models. Under this framework, a large amount of audit data is first analyzed using data mining algorithms in order to obtain the frequent activity patterns. These patterns are then used to guide the selection of system features as well as the construction of additional temporal and statistical features for another phase of automated learning. Classifiers based on these selected features are then inductively learned using the appropriately formatted audit data. These classifiers can be used as intrusion detection models since they can classify (i.e., decide) whether an observed system activity is "legitimate" or "intrusive". This approach is substantially different from the traditional signature-based approaches. We believe inductively learned models may provide generalized detection capabilities that may not be possible with a static set of hand crafted signatures.

This paper is organized as follows. We first give a brief overview of current intrusion detection techniques and their limitations. We then outline the key elements of our framework. Next, we summarize the lessons we learned from our early experiments on building classification models for intrusion detection, namely we find that tools are needed for feature selection and audit data gathering. We then discuss how to exploit the schema level information (i.e., attribute definitions) of connection records so that the association rules (Agrawal et al., 1993) and frequent episodes (Mannila et al., 1995) algorithms can be directed to compute relevant patterns from audit data efficiently. We report in detail our various experiments on using these patterns as a guide for audit data gathering, and as a basis for feature selection. Finally, we present our experimental results on combining multiple classification models for collaborative intrusion detection. And in conclusion, we compare our framework with related research efforts and discuss issues in our future research.

## 2. Intrusion detection techniques

There are mainly two types of intrusion detection techniques: *Anomaly detection* and *Misuse detection*[1]. Most current intrusion detection systems use one or both of these two approaches.

---

[1] Misuse detection is often times associated with detecting "insider attacks", while intrusion detection generally refers to "outsider attacks". We make no such distinction here.

### 2.1. Anomaly detection

Anomaly detection, for example IDES (Lunt et al., 1992), tries to determine whether deviation from an established normal behavior profile can be flagged as an intrusion. A profile typically consists of a number of statistical measures on system activities, for example, the *CPU usage* and the *frequency of system commands* during a user login session. Deviation from a profile can be computed as the weighted sum of the deviations of the constituent statistical measures. Profiles can be aged so that shifts of normal behavior are accounted for. The key advantages of anomaly detection systems is that they can detect unknown intrusion since they require no *a priori* knowledge about specific intrusions. However, defining and maintaining "normal" profiles is a nontrivial and error prone task, leading to sometimes unacceptable levels of false alarms.

### 2.2. Misuse detection

Misuse detection refers to techniques that use patterns of known intrusions (e.g., *more than three consecutive failed logins* within 2 minutes is a penetration attempt) or weak spots of a system (e.g., system utilities that have the "buffer overflow" vulnerabilities) to match and identify intrusions. The sequence of attack actions, the conditions that compromise a system's security, as well as the evidence (e.g., damage) left behind by intrusions can be represented by a number of general pattern matching models. For example, NIDES (Lunt, 1993) uses rules to describe attack actions, STAT (Ilgun et al., 1995) uses state transition diagrams to model general states of the system and access control violations, and IDIOT (Kumar and Spafford, 1995) uses Colored Petri nets to represent intrusion signatures as sequences of events on the target system. The key advantage of misuse detection systems is that once the patterns of known intrusions are stored, future instances of these intrusions can be detected effectively and efficiently. However, newly invented attacks will likely go undetected, leading to unacceptable false negative error rates.

### 2.3. Difficulties in building intrusion detection systems

Currently there is no systematic approach for building intrusion detection systems, nor any generally agreed upon evaluation metric for measuring their effectiveness. System builders' intuition and experience guide the selection of the statistical measures for anomaly detection (Lunt, 1993). Experts need to analyze and categorize attack scenarios and system vulnerabilities, and hand-code the corresponding rules and patterns for misuse detection.

In today's network computing environment, there are multiple "penetration points" for intrusions to take place. For example, at the network level, intruders can crash a host by simply sending "malicious" packets to the victim; at the host level, intruders can first login to a system and then illegally obtain a root (i.e., superuser) shell. System activities occurring at these different "penetration points" are captured in different audit data sources. However, many intrusion detection systems handle only one particular audit data source because of the knowledge and labor-intensive nature of understanding the system audit data, selecting statistical measures, and modeling intrusion patterns.

The large traffic volume in security related mailing lists and Web sites of industry groups as well as "underground societies" suggest that new system security holes and intrusion methods are continuously being discovered. Therefore, it is imperative that intrusion detection systems be updated rapidly and upgraded frequently. However, these maintenance and updates are expensive and difficult because of the current manual, ad hoc approaches generally employed.

### 2.4. A systematic framework

A basic premise for intrusion detection is that when audit mechanisms are enabled to record system events, distinct evidence of legitimate and intrusive user and program activities will be manifested in the audit data. For example, from network traffic audit data, connection failures are normally infrequent. However, certain types of intrusions will result in a large number of consecutive failures that may be easily detected. Taking a data-centric point of view, intrusion detection can therefore be considered as a data analysis process. Anomaly detection is about finding the normal usage patterns from the audit data, and misuse detection is about encoding and matching intrusion patterns using the audit data.

We are developing a framework, first described in (Lee and Stolfo, 1998), of applying data mining techniques to build intrusion detection models. This framework consists of classification and meta-classification (Chan and Stolfo, 1993), association rules and frequent episodes programs, as well as a support environment that enables system builders to interactively and iteratively drive the process of constructing and evaluating detection models. The end products are concise and consist of intuitive classification rules that can detect intrusions, and that can be easily inspected and edited by security experts when needed.

We try to eliminate the manual and ad hoc elements as much as possible. Here inductively learned classification rules replace the manually encoded intrusion patterns and profiles, and system features and measures are selected by considering the statistical patterns computed from the audit data. Further, meta-learning can be used to learn the correlation of intrusion evidence from multiple audit data sources, making it feasible to detect a full range of intrusions.

Since the process of using pattern mining and classification programs is generic and mechanical, it is much easier to update the intrusion detection models using our framework. Migration to new computing environments or dealing with new intrusion methods means applying the data mining and machine learning process to the relevant audit data[2]. Meta-learning can be used to combine the new models (e.g., those that detect new intrusions) with the existing ones (e.g., those that detect "old" intrusions). This meta-learning mechanism facilitates incremental updates of the overall combined models, making adaptive intrusion detection feasible.

[2] However, exemplar attacks need to be executed, observed and labeled in new audit data stream in order to learn new detection models. Naturally, this is also required for hand crafted signatures.

## 3. Classification models for intrusion detection

In this section we briefly summarize our early experiments on constructing classification models for anomaly detection (detailed in (Lee and Stolfo, 1998)). These experiments showed the effectiveness of classification models computed by machine learning programs.

System programs, such as *sendmail*, are frequent targets of intruders. The attacks normally cause the programs to perform malicious activities. The sequence of run-time operating system calls (e.g., *open*, *read*, ...) of a program provides very detailed information about its execution. We obtained, from Stephanie Forrest of the University of New Mexico, system call data of hundreds of normal *sendmail* runs and a dozen different simulated attacks. Based on the findings that the short sequences of system calls of a program are very consistent (Forrest et al., 1996), we pre-processed the data by using a length 11 sliding window to scan the system call sequence and create a list of records, each of which has 11 consecutive system calls. The 11th system call of each record is the class label (i.e., the target of the learning task), and the first 10 system calls are the positional features (attributes)[3]. RIPPER (Cohen, 1995), a classification rule learner that computes the most distinguishing and concise feature value tests for each class label, was then applied to the data. The resultant rule set, consisting of 252 rules each with 2 or 3 feature value tests, is the normal execution profile since it specifies the normal co-occurrences of the system calls. When it is used to analyze an execution trace, a large number of misclassifications (i.e., deviations) suggests an “anomaly”. Our experiments (detailed in (Lee and Stolfo, 1998)) showed that we need to use data from as many as 80% of the hundreds of the normal runs of *sendmail* for training in order for the classifier to clearly identify the attacks from the normal runs. Therefore, in general we need to collect a sufficiently large amount of training data to learn and establish a normal profile for use as an anomaly detector.

Recently, attacking the weaknesses in the TCP/IP protocols has become a very “cool” trend for intruders. These intrusions are aimed at disrupting network services. “Packet sniffing” programs, such as

[3] Features and attributes both refer to the fields in a database record. In this paper, we use attributes in the context of mining frequent patterns, and features in the context of building classifiers.

*tcpdump* (Jacobson et al., 1989), can capture traffic data for analysis and monitoring. We obtained a set of *tcpdump* data on network traffic, available via `http://iris.cs.uml.edu:8080/network.html`, that is part of an Information Exploration Shootout (see `http://iris.cs.uml.edu:8080`). The data set has traces of normal traffic and a number of intrusions. We followed the TCP/IP protocol specification to first process the raw packet-level data into a time series of connection-level records using a manually written preprocessing script. For each record, we used the destination network service (e.g. *http*) as the class label, and included other intrinsic connection information, for example, duration, number of bytes transfered in each direction, and the flag specifying whether there is an error according to the protocol as the features. We then applied RIPPER to the normal data. The resultant rule set, about 20 rules, can be considered as the normal profile of the network since it characterizes the normal traffic patterns for each network service.

The initial results were not good: the misclassification rates on the normal testing data and the intrusions were very close. This is because the temporal nature of network traffic is not captured in the classification model. For each connection we then added several temporal and statistical features calculated from the connections of the prior *n* seconds. These include the average duration and number of bytes, and the total number of errors, etc. The new classification models showed significant improvement in detecting intrusions. Complete details of these experiments and our findings appear in (Lee and Stolfo, 1998).

### 3.1. THE CHALLENGES

The above experiments revealed that we need to solve some very challenging problems for the classification models to be effective.

Formulating the classification tasks, i.e., determining the class labels and the set of features, from audit data is a very difficult and time-consuming task. Since security is usually an after-thought of computer system design, there is no standard auditing mechanism and data format specifically designed for intrusion analysis purposes. A considerable amount of data pre-processing, which involves domain knowledge, is required to extract raw “action-level” audit data into higher-level “session/event” records with the set of intrinsic system features. The temporal nature of event sequences in network-based computer systems suggests that temporal and statistical measures over the features can be very valuable and hence additional features (e.g., *the total number of connection failures in the past 10 seconds*) should be used. Traditional feature selection techniques, as discussed in the machine learning literature, can not be directly applied here since prior work typically does not consider sequential correlation of features across record boundaries. Fawcett and Provost (Fawcett and Provost, 1997) presented some very interesting ideas on automatic selection of features for a cellular fraud detector. Their method is very effective in detecting “superimposition fraud” in which fraudulent activities are conducted using a legitimate account. Many intrusions can not be easily categorized as “superimposition”, for example, those that attack network protocols.

A critical requirement for using classification rules as an anomaly detector is that we need to have “sufficient” training data that covers as much variation of the normal behavior as possible, so that the *false positive* rate is kept low (i.e., we wish to minimize detected “abnormal normal” behavior). It is not always possible to formulate a classification model to learn the anomaly detector with “insufficient” training data, and then incrementally update the classifier using on-line learning algorithms. This is because the limited training data may not have covered all the class labels, and on-line algorithms, for example, ITI (Incremental Tree Induction) (Utgoff et al., 1997), can’t deal with new data with new (i.e., unseen) class labels. For example, in modeling daily network traffic, we use the services, e.g., *http*, *telnet* etc., of the connections as the class labels in training models. We may not have connection records of the infrequently used services with, say, only one week’s traffic data. A formal audit data gathering process therefore needs to take place first. As we collect audit data, we need an indicator that can tell us whether the new audit data exhibits any “new” normal behavior, so that we can stop the process when there is no more variation evident during our collection process. This indicator ideally should be simple to compute and must be incrementally updated.

## 4. Algorithms for mining audit data

We aim to develop general rather than intrusion-specific tools in response to the challenges discussed in the previous section. The idea is to first compute the association rules and frequent episodes from audit data, which capture the intra- and inter- audit record patterns. These patterns are then utilized, with user participation, to guide the data gathering and feature selection processes. Here we use the term "audit data" to refer to general data streams that can be processed for detection purposes. Examples of such data streams are the connection records extracted from the raw *tcpdump* output, and the Web site visit records processed using Web site logs. We assume that audit data records are timestamped and hence ordered.

As described in (Lee et al., 1998), the main challenge in developing these data mining algorithms is to provide support mechanisms for domain knowledge so that "useful" patterns are computed. We next describe these basic data mining algorithms and our proposed extensions that allow the introduction of domain knowledge in a convenient manner.

### 4.1. The basic algorithms

From (Agrawal et al., 1993), let $\mathcal{A}$ be a set of attributes, and $I$ be a set of values on $\mathcal{A}$, called items. Any subset of $I$ is called an itemset. The number of items in an itemset is called its length. Let $\mathcal{D}$ be a database with $n$ attributes (columns). Define $support(X)$ as the percentage of transactions (records) in $\mathcal{D}$ that contain itemset $X$. An association rule is the expression $X \rightarrow Y, [c, s]$. Here $X$ and $Y$ are itemsets, and $X \cap Y = \emptyset$. $s = support(X \cup Y)$ is the support of the rule, and $c = \frac{support(X \cup Y)}{support(X)}$ is the confidence. For example, an association rule from the shell command history file of a user, which is a stream of commands and their arguments, is $trn \rightarrow$ *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.

We implemented the association rules algorithm following the ideas of Apriori (Agrawal and Srikant, 1994). Briefly, an itemset $X$ is frequent if $support(X) \geq min\_support$. The Apriori algorithm starts by finding all length 1 frequent itemsets, then iteratively generates length $k+1$ frequent itemsets from the length $k$ frequent itemsets. At each iteration, the algorithm first uses a *join* step to generate length $k+1$ candidate itemsets, each from 2 length $k$ frequent itemsets; then a *prune* step to filter out a candidate itemset if one of its length $k$ subsets is not frequent; finally the *support* values for the remaining candidate itemsets are counted to form the set of frequent $k+1$ itemsets.

Since we look for correlation among values of different attributes, and the pre-processed audit data usually has multiple columns of attributes, each with a large number of possible values, we do not convert the data into a binary database. In our implementation we trade memory for speed. The data structure for a frequent itemset has a *row vector* that records the transactions in which the itemset is contained. When a length $k+1$ candidate itemset $c_{k+1}$ is generated by joining two length $k$ frequent item sets $l_k^1$ and $l_k^2$, the row vector of $c_{k+1}$ is simply the bitwise AND product of the row vectors of $l_k^1$ and $l_k^2$. The *support* of $c_{k+1}$ can be calculated easily by counting the 1s in its row vector. There is also no need to perform the *prune* step in the candidate generation function. We minimize the memory consumption by freeing up the row vectors of length $k$ itemsets after they are used to generate the length $k$+1 itemsets. Notice that the database needs to be scanned only once to generate the list of frequent itemsets of length 1. Since most pre-processed audit data files are small enough to fit into main memory, and the size of any list of length $k$ item sets can not exceed the size of the database, this implementation works well in this particular application domain.

The problem of finding frequent episodes based on minimal occurrences was introduced in (Mannila and Toivonen, 1996). Briefly, given an event database $\mathcal{D}$ where each transaction is associated with a timestamp, an interval $[t_1, t_2]$ is the sequence of transactions that starts from timestamp $t_1$ and ends at $t_2$. The width of the interval is defined as $t_2 - t_1$. Given an itemset $A$ in $\mathcal{D}$, an interval is a minimal occurrence of $A$ if it contains $A$ and none of its proper sub-intervals contains $A$. Define $support(X)$ as the the ratio between the number of minimum occurrences that contain itemset $X$ and the number of records in $\mathcal{D}$.

A frequent episode rule is the expression $X,Y \rightarrow Z,[c,s,window]$. Here $X$, $Y$ and $Z$ are itemsets in $\mathcal{D}$. $s = support(X \cup Y \cup Z)$ is the support of the rule, and $c = \frac{support(X \cup Y \cup Z)}{support(X \cup Y)}$ is the confidence. Here the width of each of the occurrences must be less than *window*. A *serial* episode rule has the additional constraint that $X$, $Y$ and $Z$ must occur in transactions in partial time order, i.e., $Z$ follows $Y$ and $Y$ follows $X$. The description here differs from (Mannila and Toivonen, 1996) in that we don't consider a separate *window* constraint on the LHS (left hand side) of the rule. The frequent episode algorithm finds patterns in a single sequence of event stream data. The problem of finding frequent sequential patterns that appear in many different data-sequences was introduced in (Agrawal and Srikant, 1995). This related algorithm is not used in our study since the frequent network or system activity patterns can only be found in the single audit data stream from the network or the operating system.

Our implementation of the frequent episodes algorithm utilized the data structures and library functions of the association rules algorithm. Here instead of finding correlations of attribute values across columns, we look for correlations across rows. The *row vector* is now used as the *interval vector* where each pair of adjacent 1s is the pair of boundaries of an interval. A *temporal join* function, that considers minimal occurrences that are non-overlapping, is used to create the interval vector of a candidate length $k$ itemset from the two interval vectors of two length $k$-1 frequent itemsets . The *support* of an itemset can be easily counted as the number of pairs in the interval vector.

### 4.2. USING THE *axis* ATTRIBUTES(S)

These basic algorithms do not consider any domain knowledge and as a result it can generate many "irrelevant" rules. Consider the audit data of network connections shown in Table I. Here we have already discretized the continuous attribute values, except the timestamps, into proper buckets. The basic association rules algorithm may generate rules such as *src_bytes*=200 $\rightarrow$ *flag*=*SF*. These rules are not useful and to

Table I. Network Connection Records

| timestamp | duration | service | src_bytes | dst_bytes | flag | ... |
|---|---|---|---|---|---|---|
| 1.1 | 10 | telnet | 100 | 2000 | SF | ... |
| 2.0 | 2 | ftp | 200 | 300 | SF | ... |
| 2.3 | 1 | smtp | 250 | 300 | SF | ... |
| 3.4 | 60 | telnet | 200 | 12100 | SF | ... |
| 3.7 | 1 | smtp | 200 | 300 | SF | ... |
| 3.8 | 1 | smtp | 200 | 300 | SF | ... |
| 5.2 | 1 | http | 200 | 0 | REJ | ... |
| 5.7 | 2 | smtp | 300 | 200 | SF | ... |
| ... | ... | ... | ... | ... | ... | ... |

some degree misleading. There is no intuition for the association between the number of bytes from the source, *src_bytes*, and the normal status (i.e., $flag$=*SF*) of the connection.

In (Klemettinen et al., 1994), rule templates specifying the allowable attribute values are used to post-process discovered rules. In (Srikant et al., 1997), boolean expressions over the attribute values are used as item constraints during rule discovery. A drawback of these approaches is that one has to know a priori what rules and patterns are interesting. We can't assume such strong prior knowledge on all audit data.

We instead utilize the schema-level information about audit records to direct the pattern mining process. Observe that certain attributes are essential in describing the data, while others provide only "auxiliary information". For example, a network connection can be uniquely identified by the combination of its *timestamp* (start time), *src_host* (source host), *src_port* (source port), *dst_host* (destination host), and *service* (destination port). These are the essential attributes when describing network data. We argue that the "relevant" association rules should describe patterns related to the essential attributes.

We call the essential attribute(s) *axis* attribute(s) when they are used as a form of item constraints in the association rules algorithm. During candidate generation, an item set must contain value(s) of the axis attribute(s). That is, we consider the correlations among non-axis attributes as not interesting. In practice, we need not designate all essential attributes as the axis attributes. For example, some network analysis tasks require statistics about various network services while others may require the patterns related to the hosts.

We can use *service* as the axis attribute to compute the association rules that describe the patterns related to the services of the connections.

It is even more important to use the axis attribute(s) to constrain the item generation for frequent episodes. The basic algorithm can generate serial episode rules that contain "non-essential" attribute values, for example, *src_bytes*=200, *src_bytes*=200 $\rightarrow$ *dst_bytes*=300, *src_bytes*=200. Note that here each attribute value, for example, *src_bytes*=200, is from a separate connection record. Compared with the association rules, the total number of serial rules is large and so is the number of such useless rules. Observe that the number of iterations for growing the frequent itemsets (i.e., the length of an itemset) is bounded here by the number of rows instead of the number of columns as in association rules, which tends to be a large number. Further, if the support of an association rule on non-axis attributes, $A \rightarrow B$, is high then there will be a large number of "useless" serial episode rules of the form $(A|B)(,A|B)^* \rightarrow (A|B)(,A|B)^*$. To see this, assume that there are a total of $m$ records (rows) in the database, the time difference from the last and the first record is $t$ seconds, and the support of $A \cup B$ is $s$. Then the number of minimal and non-overlapping intervals that have $k$ records with $A \cup B$ is $\frac{sm}{k}$. Note that each of these intervals contains a length $k$ serial episode on $(A|B)$. Assume that the records with $A \cup B$ are evenly distributed, then the width of the interval is $w = \frac{kt}{sm}$. There can be a large number of serial patterns on $(A|B)$ if $s$ is high and *window* (i.e., the interval width threshold) is large, since $k_{max}$, the maximal length of the patterns, can be large and $w \leq width$ still holds.

Instead of using the basic algorithm, here we first find the frequent associations using the axis attributes(s) and then generate the frequent serial patterns from these associations. An example of a rule is

$$(service = smtp, src\_bytes = 200, dst\_bytes = 300, flag = SF),$$

$$(service = telnet, flag = SF) \rightarrow$$

$$(service = http, src\_bytes = 200), [0.2, 0.1, 2s]$$

Note that each itemset of the episode rule, e.g., (*service* = *smtp*, *src_bytes* = 200, *dst_bytes* = 300, *flag* = *SF*), is an association. We in effect have combined the associations among attributes and the sequential patterns

Table II. Web Log Records

| timestamp | remote host | action | request | ... |
|---|---|---|---|---|
| 1 | his.moc.kw | GET | /images | ... |
| 1.1 | his.moc.kw | GET | /images | ... |
| 1.3 | his.moc.kw | GET | /shuttle/missions/sts-71 | ... |
| ... | ... | ... | ... | ... |
| 3.1 | taka10.taka.is.uec.ac.jp | GET | /images | ... |
| 3.2 | taka10.taka.is.uec.ac.jp | GET | /images | ... |
| 3.5 | taka10.taka.is.uec.ac.jp | GET | /shuttle/missions/sts-71 | ... |
| ... | ... | ... | ... | ... |
| 8 | rjenkin.hip.cam.org | GET | /images | ... |
| 8.2 | rjenkin.hip.cam.org | GET | /images | ... |
| 9 | rjenkin.hip.cam.org | GET | /shuttle/missions/sts-71 | ... |
| ... | ... | ... | ... | ... |

among the records into a single rule. This rule formalism provides rich and useful information about the audit data.

### 4.3. USING THE *reference* ATTRIBUTE(S)

Another piece of interesting schema-level information about audit records is that some essential attributes can be the *references* of other attributes. These reference attributes normally carry information about some "subject", and other attributes describe the "actions" that refer to the same "subject". Consider the log of visits to a Web site, as shown in Table II. Here *action* and *request* are the "actions" taken by the "subject", *remote host*. We see that for a number of remote hosts, each of them makes the same sequence of requests: "/images", "/images" and "/shuttle/missions/sts-71". It is important to use the "subject" as a reference when finding such frequent sequential "action" patterns because the "actions" from different "subjects" are normally irrelevant. This kind of sequential pattern can be represented as

$$(subject = X, action = a_1), (subject = X, action = a_2) \rightarrow (subject = X, action = a_3), [c, s, window]$$

**Input**: the terminating minimum support $s_0$, the initial minimum support $s_i$, and the axis attribute(s)
**Output**: frequent episode rules *Rules*
**Begin**
(1) $R_{restricted} = \emptyset$;
(2) scan database to form $L = \{$large 1-itemsets that meet $s_0\}$;
(3) $s = s_i$;
(4) **while** $(s \geq s_0)$ **do begin**
(5) find serial episodes from $L$: each pattern must contain at least one axis attribute value that is not in $R_{restricted}$;
(6) append new axis attribute values to $R_{restricted}$;
(7) append episode rules to the output rule set *Rules*;
(8) $s = \frac{s}{2}$;
**end while**
**end**

*Figure 1.* Level-wise Approximate Mining of Frequent Episodes

Note that within each occurrence of the pattern, the *action* values refer to the **same** *subject*, yet the actual *subject* value may not be given in the rule since any particular *subject* value may not be frequent with regard to the entire dataset. In other words, *subject* is simply a *reference* or a *variable*.

The basic frequent episodes algorithm can be extended to consider reference attribute(s). Briefly, when forming an episode, an additional condition is that, within its minimal occurrences, the records covered by its constituent itemsets have the same value(s) of the reference attribute(s).

In summary, axis attributes and reference attributes provide a convenient and easy to use means of specifying domain knowledge to constrain the outcome of the association rules and frequent episodes algorithms to produce "relevant" output.

### 4.4. *Level-wise* APPROXIMATE MINING

Sometimes it is important to discover the low frequency patterns. For example, in daily network traffic some services, for example, *gopher*, account for a very low percentages of activity. Yet we still need to include their patterns into the network traffic profile so that we have representative patterns for each supported service. If we use a very low support value for the data mining algorithms, we will then get unnecessarily a very large number of patterns related to the high frequency services, for example, *smtp*.

We propose a *level-wise* approximate mining procedure, as outlined in Figure 1, for finding the frequent episodes. Here the idea is to first find the episodes related to high frequency axis attribute values, for example,

$$(service = smtp, src\_bytes = 200), (service = smtp, src\_bytes = 200) \rightarrow$$
$$(service = smtp, dst\_bytes = 300), [0.3, 0.3, 2s]$$

We then iteratively lower the support threshold to find the episodes related to the low frequency axis values by *restricting* the participation of the "old" axis values that already have output episodes. More specifically, when a candidate itemset is generated, it must contain at least one "new" (i.e., low frequency) axis value. The episodes from each iteration are those consisting of either all new axis values or a combination of new and old axis values. For example, in the second iteration, where *smtp* now is an old axis value, we get an episode rule with *http* as a new axis value

$$(service = smtp, src\_bytes = 200), (service = http, src\_bytes = 200) \rightarrow$$
$$(service = smtp, src\_bytes = 300), [0.4, 0.1, 2s]$$

The procedure terminates when a very low support value is reached. In practice, this can be the lowest frequency of all axis values.

Note that for a high frequency axis value, we in effect omit its very low frequency episodes generated in the runs with low support value because they are not as interesting and representative. Hence our procedure is "approximate" mining. We still include all the old (i.e., high frequency) axis values to form episodes with the new axis values because it is important to capture the sequential context of the new axis values. For example, although used infrequently, *auth* normally co-occurs with other services such as *smtp* and *login*. It is therefore imperative to include these high frequency services when presenting episode rules about *auth*.

Our approach here is different from the algorithms in (Han and Fu, 1995) since we do not have and can not assume multiple concept levels, rather, we deal with multiple frequency levels of a single concept, e.g., the network service.

## 5. Using the mined patterns

In this section we report upon our experience in mining the audit data and using the discovered patterns both as the indicator for gathering data and as the basis for selecting appropriate temporal and statistical features for learning classification models.

### 5.1. Audit data gathering

We posit that the patterns discovered from the audit data on a protected target (e.g., a network, system program, or user, etc.) corresponds to the target's behavior. When we gather audit data to learn the target's normal profile, we can compute the patterns from each new audit data set, and *merge* the new rules into the existing aggregate rule set. These added new rules represent new variations of the normal behavior. When the aggregate rule set stabilizes, i.e., no new rules from the new audit data can be added, we can stop the data gathering since the aggregate audit data set has covered sufficient variations of the normal behavior[4].

We merge two rules, $r_1$ and $r_2$, into one rule $r$ if 1) their right and left hand sides are exactly the same, or their RHSs can be *combined* and LHSs can also be *combined*; and 2) the $support$ values and the $confidence$ values are *close*, i.e., within an $\varepsilon$. The concept of *combining* here is similar to *clustering* in (Lent et al., 1997) in that we also combine rules that are "similar" or "adjacent".

To simplify our discussion, consider combining the LHSs and assume that the LHS of $r_1$ has just one itemset, $(ax_1 = vx_1, a_1 = v_1)$. Here $ax_1$ is an axis attribute. The LHS of $r_2$ must also have only one itemset, $(ax_2 = vx_2, a_2 = v_2)$. Further, $ax_1 = ax_2$, $vx_1 = vx_2$, and $a_1 = a_2$ must hold (i.e., the LHSs must cover the same set of attributes, and their axis values must be the same). For the LHSs to be combined, $v_1$ and $v_2$ must be adjacent values or adjacent bins of values. The LHS of the merged rule $r$ is $(ax_1 = vx_1, v_1 \leq a_1 \leq v_2)$, assuming that $v_2$ is the larger value. For example, (*service*=*smtp*, *src_bytes*=200) and (*service*=*smtp*,

[4] Presently, we do not know of any formal means of determining when sufficient normal behavior has indeed been observed. Installing new services or programs on a system clearly requires new data gathering to establish new norms.

*src_bytes*=300) can be combined to (*service*=*smtp*, $200 \leq$ *src_bytes* $\leq 300$). To compute the statistically relevant *support* and *confidence* values of the merged rule $r$, we record *support_lhs* and *db_size* of $r_1$ and $r_2$ when mining the rules from the audit data. Here *support_lhs* is the support of a LHS and *db_size* is the number of records in the audit data.

Our approach of merging rules is based on the intuition that even the same behavior will have slight differences across audit data sets. Therefore we should not expect exact matches of the mined patterns. Instead we need to combine similar patterns into more generalized ones.

#### 5.1.1. *Experiments*

Here we test our hypothesis that the merged rule set can indicate whether the audit data has covered sufficient variations of normal behavior. We obtained one month of TCP/IP network traffic data from `http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html`. We hereafter refer to it as the LBL dataset. We segmented the data by day. With the data of each day, we again segmented the data into four partitions: morning, afternoon, evening and night. This partitioning scheme allowed us to cross evaluate anomaly detection models of different time segments that have different traffic patterns. It is often the case that very little and sometimes no intrusion data is available when building an anomaly detector. A common practice is to use audit data of legitimate activities that is known to have different behavior patterns for testing and evaluation. We followed this approach here.

We first describe the experiments and results on the "weekday morning" traffic data on connections originated from LBL to the outside world. We decided to compute the frequent episodes using the network *service* as the axis attribute. Recall from section 4.2 that this formalism captures both association and sequential patterns of network services. We also used the *level-wise* approximate mining procedure so that even the infrequent services are included in the mined patterns. For the first three weeks, we mined the patterns from the audit data of each weekday morning, and merged them into an aggregate rule set. For each rule in the rule set we record *merge_count*, the number of merges on each rule. Note that if two rules $r_1$

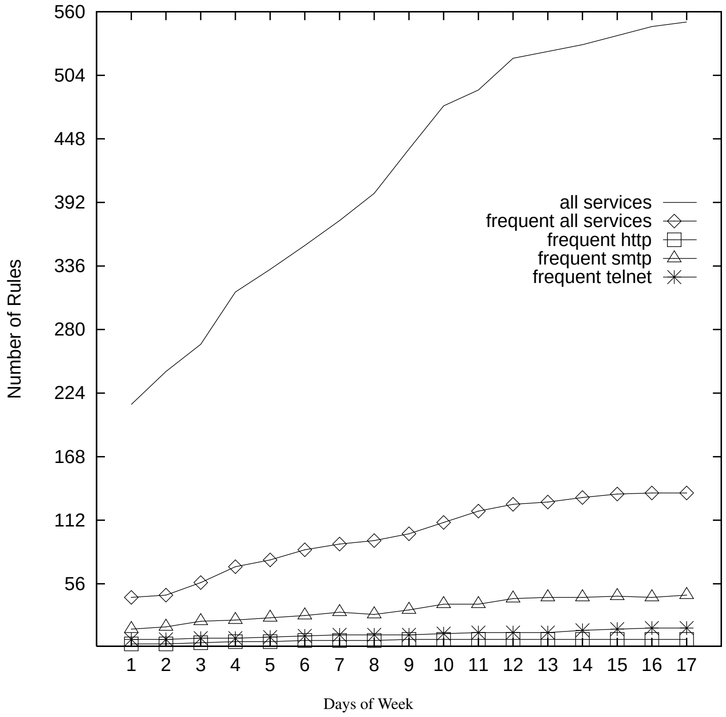
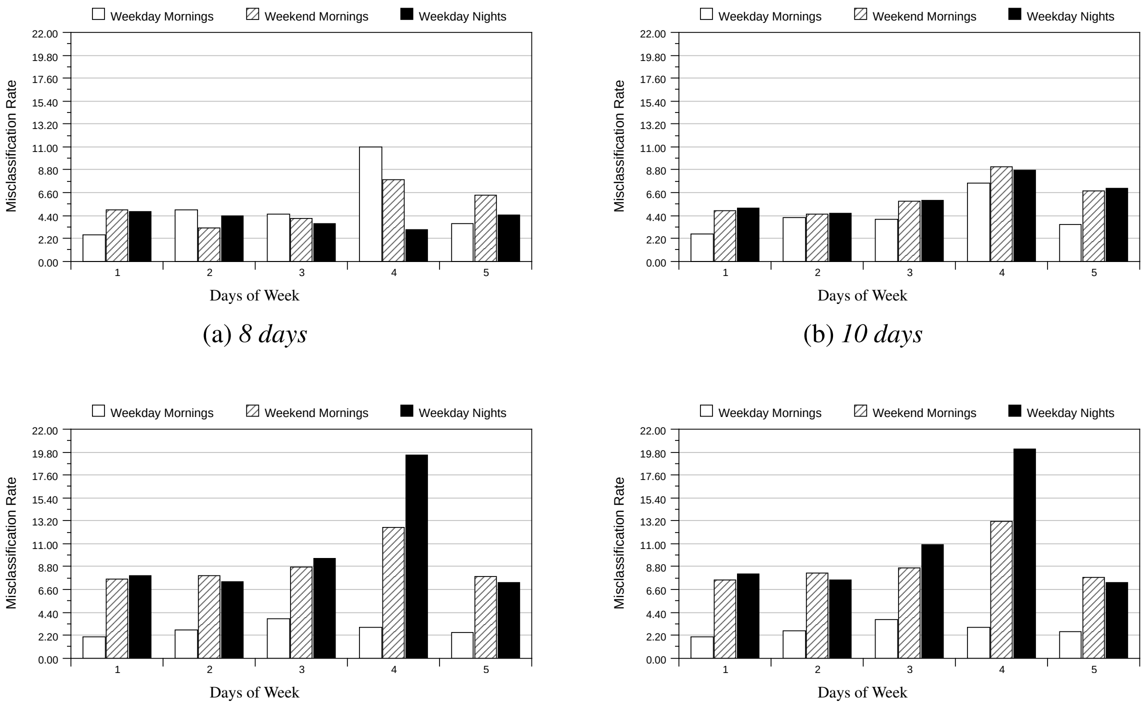



*Figure 2.* The Number of Rules vs. The number of Audit Data Sets

and $r_2$ are merged into $r$, its *merge_count* is the sum from the two parent rules. *merge_count* indicates how frequent the behavior represented by the merged rule is encountered across a period of time. The rules with $merge\_count \geq min\_frequency$ are regarded as frequent rules. Frequent rules are more reliable in describing behavior since they are "verified" by multiple audit data sets. The *freshness* field of a rule indicates the last time the rule was merged. It can be used to age out "old" frequent rules.

Figure 2 plots how the rule set changes as we merge patterns from each new audit data set. We see that the total number of rules monotonically increases. We visually inspected the new patterns from each new data set. In the first two weeks, the majority are related to "new" network services that have no prior patterns in the aggregate rule set, and for the last week, the majority are just new patterns of the existing services. Figure 2 shows that the rate of change indeed slows down during the last week. We used $min\_frequency = 2$ to filter out the "one-time" patterns so that the remaining patterns are considered "frequent". We can see in the plots that the frequent rule sets of all services as well as the individual services grow at a much slower rate and tend to stabilize.




(a) *8 days* (b) *10 days* (c) *15 days* (d) *17 days*

*Figure 7.* Misclassification Rates of Classifier Trained on First 8, 10, 15 and 17 Weekdays

We used the set of frequent rules of all services as the indicator on whether the audit data is sufficient to model normal behavior. We tested the quality of this indicator by constructing four classifiers, using audit data from the first 8, 10, 15, and 17 weekday mornings, respectively, for training anomaly detectors. We used the services of the connections as the class labels, and included a number of temporal and statistical features (we discuss the feature selection and construction process in Session 5.3). The classifiers were tested using the audit data from the mornings and nights of the last 5 weekdays of the month, as well as the last 5 weekend mornings. Figure 5.1.1 shows the performance of these four classifiers respectively. On each plot, we show the misclassification rate on the test data. Since the classifiers model the weekday morning traffic, we wish to see this rate be low on the weekday morning test data, but high on the weekend morning data as well as the weekday night data. The figures show that the classifiers with more training data perform better. Further, the performance of the last two classifiers are very close. This is not surprising at all because from the plots in Figure 2, the set of frequent rules, that is, our indicator on audit data, grows in weekday 8 and 10, but stabilizes from day 15 to 17.

This experiment provides evidence that indeed the rate of growth of the rule set is a plausible indicator on whether sufficient training data has been collected for building useful models for anomaly detection. However, this study involved only one specific externally supplied data set. Further study is required to determine how generally applicable this might be to other contexts.

### 5.2. Off-line analysis

Here we discuss some ideas and experiments on using the mined patterns as a means for understanding and comparing the activity behavior exhibited in the audit data. The fact that the merged rule set can be used to model "new" behavior when gathering audit data suggests that we can use the final rule set directly to detect anomalies. Also, by mining and comparing specific patterns from audit data, we can discover whether certain known intrusion patterns exist. In other words, the patterns might be used for misuse intrusion detection directly.

#### 5.2.1. *Anomaly detection: experiments on LBL dataset*

We used the final set of frequent rules to distinguish the traffic data of the last week's weekday mornings from all the weekends' mornings, and from the last week's weekday nights. We use a *similarity* measure. Assume that the merged rule set has $n$ rules, the size of the new rule set from an audit data set is $m$, and the number of *matches* (i.e., the number of rules that can be merged) between the merged rule set and the new rule set is $p$, then we have $similarity = \frac{p}{n} * \frac{p}{m}$. Here $\frac{p}{n}$ represents the percentage of known behavior (from the merged rule set) exhibited in the new audit data, and $\frac{p}{m}$ represents the proportion of behavior in the new audit data that conforms to the known behavior. Figure 8 shows that in general the *similarity* of the weekday mornings are much larger than the weekend mornings and the weekday nights.

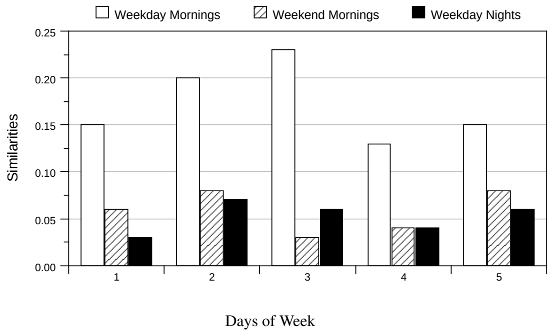



*Figure 8.* Similarity Measures Against the Merged Rule Set of Weekday Mornings

Note that in general these mined patterns cannot be used directly to classify the records. That is, they cannot tell *which* records are an anomaly. However they are very valuable in off-line analysis. We built a graphical tool to visually display the differences of two rule sets. When comparing the weekday morning rule set with a weekend morning, we see the latter rule set has a large number of patterns of *telnet* from 2 or 3 particular machines. This was due to the fact that only a few people showed up to work on the weekend mornings, therefore each machine is relatively frequent. Whereas in the weekdays, many machines used *telnet* and thus each alone is not frequent enough.

5.2.2. *Misuse detection: experiments on InfoWorld IWSS16 dataset*

We report our recent experiments on a set of network intrusion data from InfoWorld, which contains attacks of the "InfoWorld Security Suite 16" that was used to evaluate several leading commercial intrusion detection products (McClure et al., 1998). We hereafter refer this dataset as the IWSS16 dataset.

We were given two traces of *tcpdump* data. One contains 4 hours of normal network traffic, and the other contains 2 hours of network traffic where 16 different types of attacks were simulated. We first summarized the raw packet level *tcpdump* data into connection records using the preprocessing script described in Section 3. For each record, we included only the "intrinsic" connection attributes such as *service*, *duration*, *flag*, etc., that is, no temporal and statistical features were added.

According to their attack methods[5], several intrusions in IWSS16 would leave distinct evidence in the short sequence of time ordered connection records. Below we demonstrate how our frequent episodes program can be used to find or test the patterns of these attacks. The main idea is to mine frequent episodes from both the normal data and attack data, and compare the patterns to identify those that are from the attack data only. The procedure of comparing patterns and identifying intrusion patterns is discussed in Section 5.3. We used a time window of 2 seconds when computing the following frequent episodes.

– **Port Scan and Netsonar Scan:** The attacker systematically makes connections to each port (i.e., service) of a target host (i.e, the destination host) in order to find out which ports are accessible. In the connection records, there should be a host (or hosts) that receives many connections to its "different" ports in a short period of time. Further, a lot of these connections have the "REJ" flag since many ports are normally unavailable (hence the connections are rejected).

  - Data mining strategy: use *dst_host* as both the *axis* attribute and the *reference* attribute to find the "same destination host" frequent sequential "destination host" patterns;
  - Evidence in intrusion data: there are several patterns that suggest the attack, for example,

$$(dst\_host = host_v, flag = REJ), (dst\_host = host_v, flag = REJ) \rightarrow$$
$$(dst\_host = host_v, flag = REJ), [0.65, 0.2, 2s]$$

    But no patterns with $flag = REJ$ are found when using the *service* as the *axis* attribute and *dst_host* as the *reference* attribute since a large number of different services are attempted in a short period time, and as a result, for each service the "same destination host" sequential patterns are not frequent;
  - Contrast with normal data: patterns related to $flag = REJ$ indicate that the "same" service is involved.

[5] Scripts and descriptions of many intrusions can be found using the search engine in `http://www.rootshell.com`

– **Syn Flood:** The attacker makes a lot of "half-open" connections, by sending only a "syn request" but not establishing the connection, to a port of a target host in order to fill up the victim's connection-request buffer. As a result, the victim will not be able handle new incoming requests. This is a form of "denial-of-service" attack. In the connection records, there should exist a host where one of its port receives a lot of connections with flag "S0" (i.e., only the "syn request" packet is seen) in a short period of time.

  - Data mining strategy: use *service* as the *axis* attribute and *dst_host* as the *reference* attribute to find the "same destination host" frequent sequential "service" patterns;
  - Evidence in intrusion data: there is very strong evidence of the attack, for example,

$$(service = http, flag = S0), (service = http, flag = S0) \rightarrow$$
$$(service = http, flag = S0), [0.6, 0.15, 2s]$$

  - Contrast with normal data: no patterns with $flag = S0$.

– **ICMP Flood:** The attacker sends a very large number of *icmp echo* messages to a target host in an attempt to use up its resources since the kernel will be busy processing the incoming messages. This is another form of "denial-of-service" attack. In the connection records, there should exist a host that receives a lot of *icmp echo* connections in a short period of time.

  - Data mining strategy: use *service* as the *axis* attribute and *dst_host* as the *reference* attributes to find the "same connection host pair" frequent sequential "service" patterns;
  - Evidence in intrusion data: there is very strong evidence of the attack, for example,

$$(service = icmp\_echo, src\_host = host_h, dst\_host = host_v),$$
$$(service = icmp\_echo, src\_host = host_h, dst\_host = host_v) \rightarrow$$
$$(service = icmp\_echo, src\_host = host_h, dst\_host = host_v), [0.45, 0.1, 2s]$$

- Contrast with normal data: no such patterns.

Other types of intrusions in IWSS16 do not show evidence in connection record sequences since their attack actions can be embedded in a single connection. Section 5.3.1 discusses how to mine patterns from the packet level data and improve the detection models.

## 5.3. Feature selection and construction

We use the mined frequent episodes from network connection records as guidelines to construct temporal and statistical features for building classification models. An iterative procedure that involves pattern mining and comparison, feature construction from patterns, and model building and evaluation, is needed in order to find the most effective classification model. In each iteration, we choose a different data mining strategy regarding the choices of axis attributes and reference attributes. These choices are limited among the "essential" attributes (see Section 4.2), that is, *service*, *dst_host*, *src_dst*, or *src_port*. Note that the exact *timestamp* is never frequent and is thus omitted. Since intrusions are generally targeted to some victim host(s) in the network, we start with *service* and *dst_host*.

For building misuse detection models, we apply the frequent episodes program to both normal connection data and intrusion data, then compare the resulting patterns to find the "intrusion only" patterns. The details of the pattern comparison algorithm is described in (Lee et al., 1999). Briefly, since the number of patterns may be very large and there are rarely exactly matched patterns from two data sets, this algorithm uses a heuristic, in the form of domain-specific "order of importance" among attributes[6], and considers two episodes related to two different sets of "important" attributes (e.g., *service*, *dst_host*, and *flag*) as more different, while episodes with the same set of "important" attributes but different sets of "unimportant" attributes (e.g., *src_bytes*) as more alike. This algorithm then outputs a user-specified top percent of the most "intrusion only" patterns.

[6] From the data schema perspective, "order of importance" specifies which attribute is more important, compared with others, in describing the data.

**Input**: a frequent episode, and the set of existing features in connection records, $\mathcal{F}$
**Output**: the updated $\mathcal{F}$
**Begin**
(1) Let $F_0$ (e.g., *dst_host*) be the reference attribute used to mine the episode;
(2) Let $w$, in seconds, be the minimum width of the episode;
/* all the following features consider only the connections in past $w$
 * seconds that share the same value in $F_0$ as the current connection
 */
(3) Let $count\_same_{F_0}$ be the number of these connections;
(4) $\mathcal{F} = \mathcal{F} \cup \{count\_same_{F_0}\}$;
(5) **for** each "essential attribute" $F_1$ other than $F_0$ **do begin**
(6) **if** the same $F_1$ value is in all the item sets **then begin**
(7) Let $count\_same_{F_1}$ be the number of connections that share the same $F_1$ value as the current connection;
(8) $\mathcal{F} = \mathcal{F} \cup \{count\_same_{F_1}\}$;
**end else**
/* there are different $F_1$ or no $F_1$ values at all */
(9) Let $count\_diff_{F_1}$ be the number of different $F_1$ values in the connections;
(10) $\mathcal{F} = \mathcal{F} \cup \{count\_diff_{F_1}\}$;
**end**
**end**
(11) **for** each value $V_2$ of an non-essential attribute $F_2$ **do begin**
(12) **if** $V_2$ is in all the item sets **then begin**
(13) Let $count\_same_{V_2}$ be the number of connections that share the same $V_2$ value as the current connection;
(14) $\mathcal{F} = \mathcal{F} \cup \{count\_same_{V_2}\}$;
(15) **end else if** $F_2$ is a numerical attribute **then begin**
(16) Let $average_{F_2}$ be the average of the $F_2$ values of the connections;
(17) $\mathcal{F} = \mathcal{F} \cup \{average_{F_2}\}$;
**end**
**end**
**end**

*Figure 9.* Constructing Features from Frequent Episode

Each of the intrusion patterns, for example, the "syn flood" pattern

$$(service = http, flag = S0), (service = http, flag = S0) \rightarrow (service = http, flag = S0)$$

is used for constructing additional features into the connection records according to the algorithm in Figure 9. This "syn flood" pattern results in the following additional features: a count of connections to the same *dst_host* in the past 2 seconds, and among these connections, a count of those that have the same *service*, and count of those that have the "S0" *flag*.

An open problem here is how to decide the right time window value $w$. We mined sequential patterns using different $w$ values, from 0.5 to 20 with an increment of 0.5, and plotted the number of patterns generated at each run. Our experience showed that this plot tends to stabilize after the initial sharp jump. We call the smallest $w$ in the stable region $w_0$. In (Lee and Stolfo, 1998), we reported experiments of using different $w$ values to calculate temporal and statistical features for classification models. Our results showed the plot of accuracy of the classifier also stabilizes after $w \geq w_0$ and tends to taper off. Intuitively, a requirement for a good window size is that its set of sequential patterns is stable, that is, sufficient patterns are captured and noise is small. We therefore use $w_0$ for adding temporal and statistical features.

For building anomaly detection models when there is no intrusion data available, we don't need the pattern comparison step. We simply use the feature construction procedure outlined in Figure 9 on all the frequent episodes mined from the normal data. The classifiers briefly described in Section 5 include these temporal and statistical features: for the past 140 seconds, the number of connections with the same *service* as the current connection, and among these connections, the count of "SF" *flag*, the average of *duration*, the average of *src_bytes*, and the average of *dst_bytes*.

#### 5.3.1. *Experiments on IWSS16 dataset*

We report our experiments of building misuse detection models for the IWSS16 attacks. We applied the feature construction process discussed above, and for each connection record we added these temporal and statistical measures on the connections computed over the prior 2 seconds:

– For connections that have the same destination host as the current connection: the total number of such connections, the number of connections with the "S0" or "S1" flags (connection establishment errors), the number of connections with "REJ" flag, and the number of different services;

Table III. Labeled Connection Records of IWSS16 Dataset

| protocol | flag | dst_count | src_count | srv_S01_count | ... | label |
|---|---|---|---|---|---|---|
| telnet | SF | 1 | 1 | 0 | ... | normal |
| icmp_echo | SF | 1 | 1 | 0 | ... | normal |
| icmp_echo | SF | 350 | 350 | 0 | ... | icmp_flood |
| telnet | S0 | 1 | 1 | 0 | ... | normal |
| http | S0 | 25 | 20 | 20 | ... | syn_flood |
| ... | ... | ... | ... | ... | ... | ... |

– For connections that have the same service as the current connection: the total number of such connections, the number of connections with the “S0” or “S1” flags, the number of connections with “REJ” flag, the number of distinct source hosts, and the number of distinct destination hosts.

We used the intrusion types (i.e., “normal” or one of the IWSS16) as the class labels. Table III shows examples of these labeled connections. The data set has a total of 12,027 connection records. We used 80% of the connection records from the intrusion trace as the training dataset and the remaining 20% as the testing dataset. Note that since each of the attack types was simulated more than once, we were able to have all intrusion types represented in both the training and testing datasets. The resultant RIPPER classifier had an overall accuracy of 99.1%, a false positive (i.e., “normal” classified as “intrusion”) rate of 1% and a false negative (i.e., “intrusion” classified as “normal”) rate of 0.6% on the testing dataset. Note that the false positive rate is calculated as the percentage of negatives incorrectly classified, and the false negative rate is calculated as the percentage of positives incorrectly classified. There were 62 rules in the resultant RIPPER classifier, and all were very intuitive. Table IV shows some example RIPPER rules.

Although the false negative rate is very low, the classifier actually detected only the 5 “scan and flood” intrusions described in Section 5.2.2. Each of these intrusions generates a large number of connections, each of which is labeled by the intrusion type. The other intrusions, for example “teardrop”, can each generate just one connection that is sufficient to bring down the victim host. Therefore we can not overlook the low false negative rate. However, because of the features we had chosen, the connection records of these intrusions

Table IV. RIPPER Rules for IWSS16

| RIPPER rule | Meaning |
|---|---|
| syn_flood :- flag = S0, srv_S01_count >= 4. | if the current connection flag is S0, and for connections that have the same service as the current one for the past 2 seconds, the number with S01 (i.e., S0 or S1) flag is at least 4, then this connection is a syn flood attack. |
| icmp_flood :- protocol = icmp_echo, dst_count >= 5, srv_count >= 5. | if the connection protocol is icmp echo, and for the past 2 seconds, there are at least 5 connections that have the same destination host as the current connection, and at least 5 connections that have the same service as the current one, then this connection is a icmp flood attack. |
| ... | ... |
| normal :- true. | if none of the above, it is a normal connection. |

match the characteristics of the “normal” connections. Therefore we need to modify our preprocessing script to include the “missing” system features that are distinctive for the intrusions.

We did a minimum amount of data preprocessing on the raw-packet level data. Each packet-level record has the following features: *timestamp*, *connection_id* (i.e., the connection to which the packet data belongs), *head*, and *tail*. Here for TCP protocols, *head* is the flag in the TCP header (e.g., “S” for a syn packet, “P” for a data packet, etc.); for other protocols, it is the name of the protocols, for example, “icmp echo”. *tail* is the remaining information in the packet header, for example “ack”, “win” etc. We ran our frequent episodes program with *head* as the *axis* attribute and *connection_id* as the *reference* attribute since packets of different connections are not related. The patterns from the normal data present no surprises since they merely indicate that the packets are following the most common features of the protocols. For example,

$$(head = P, tail = ack\_win) \rightarrow (head = P, tail = ack\_win)$$

means that a data packet is followed by another, each with acknowledgment and window advertisement information. However, there are several distinct patterns from the intrusion data.

$$(head = frag), (head = frag) \rightarrow (head = frag)$$

and

$$(tail = frag), (tail = frag) \rightarrow (tail = frag)$$

suggest that there are connections with many fragmented packets.

$$(head = P, tail = ack\_win\_urg) \rightarrow (head{=}P, tail = ack\_win\_urg)$$

indicates that in some connections, the "urgent" flag is used when data packets are transfered.

We modified our preprocessing scripts according to these "new" patterns: we include a flag in the connection record to indicate the presence of "urgent" data packets; we examine the packet fragments to check for violations of the TCP/IP protocols, and include a flag that indicates the existence of such violations. The resultant new RIPPER classifier, learned from the new connection-level training data, detected 4 additional intrusions, hereafter referred to as the "funny packet" intrusions. As a result, the false negative rate dropped to 0.2% [7].

There is a limit to any automated approach. Our method of mining the packet-level audit data failed to find patterns of the three remaining undetected network-based intrusions. Both "land" and "latierra" spoof a connection by faking the original host and port so that the connection appears to be originated from and destined for the same host and port. Since only a single "syn request" packet with the same host and port is seen, it is rather difficult for our frequency-based mining algorithms to discover this pattern. The "zone transfer" attack uses the TCP version of *dns* to transfer a large number of data packets. Its packet-level

[7] Among these 4 intrusions, "teardrop", "ping of death", and "bonk" all have violations on fragmentation with each attack using a different network service, "oob" (or "WinNuke") exploits the fact that some Windows systems would simply crash when receiving any "urgent" packets.

patterns are normal since many normal TCP connections of other network services also transfer a lot of data packets. We incorporated this "expert" knowledge to our connection record for the sake of completeness.

The last remaining four intrusions of the IWSS16 dataset leave no evidence in any packet header since they are host-based attacks rather than network-based. Section 6 discusses how to incorporate evidence from the multiple data sources to construct a "complete" detection model that consider both network-level and host-level audit data.

## 6. Combining multiple detection models

There are several important reasons for combining multiple detection models. First, in order to avoid becoming a performance bottleneck and an easy target of "subversion", an intrusion detection system should consist of multiple cooperative lightweight subsystems that each monitors separate parts (i.e., access points) of the entire network environment. Some current network intrusion detection systems run on the gateway that separates the protected LAN and the outside network. In order for these centralized systems to detect the full range of intrusions (e.g., guessing password to gain access) they must examine not only the header of each packet, but the data portion (e.g., how many times a password string is sent) as well. Since every packet has to go through this single complex checkpoint, network throughput will be affected. Alternatively we can have one relatively lightweight system on the gateway that checks only the packet headers, and several host-based systems that monitor the activities on the "important" hosts. A "global" detection system can then combine the evidence from these subsystems and take appropriate action. For example, upon receiving signals from a host-based subsystem that a connection involves illegal activities, the global system can instruct the gateway subsystem to intervene and terminate the offending connection.

An effective intrusion detection system must have both misuse and anomaly detection components. We trained an anomaly detection model using only the normal traffic data of the ISSW16 data set. We used the services of the connections as the class labels. The resultant RIPPER rules, with no prior knowledge of any intrusion, were able to detect the intrusions described in Section 5.2.2 because these intrusions generate a large number of connections that are out of normal characteristics according to the temporal and statistical measures. However, even after adding the flags on fragmentation and "urgent" packets, the new rule sets still could not detect the 4 "funny packet" intrusions as the misuse detection model did. The reason is the "normal" rules include only the characteristics that are unique to each service, the class label, yet for every normal connection, regardless of its service, these flags are always "off". On the other hand we can not write off the anomaly detection models since the misuse detection can detect only known intrusions.

We believe that the best way to make intrusion detection models adaptive is by combining existing models with new models trained on new intrusion data or new normal data, e.g., from a new component of the network, instead of restarting the entire model building process using the aggregate of the archived audit data and new data.

Meta-learning can be used to inductively learn a meta-classifier that combines a number of pre-existing base classifiers (i.e., component detection models). Here in order to obtain the combined model (i.e., meta-classifier), meta-level connection records are constructed where the features are predictions from the base detection models, and the class labels are the original connection labels (i.e., "normal" or an intrusion). A machine learning algorithm can then be applied to the meta-level records to produce the combined detection model.

We have previously performed a number of meta-learning experiments for credit card fraud detection (Stolfo et al., 1997). We next describe similar experiments in intrusion detection.

Table V. Meta-level Connection Records

| old_model_prediction | new_model_prediction | label |
|---|---|---|
| normal | normal | normal |
| normal | teardrop | teardrop |
| syn_flood | normal | syn_flood |
| syn_flood | syn_flood | syn_flood |
| normal | syn_flood | normal |
| normal | bonk | bonk |
| ... | ... | ... |

## 6.1. Experiments: combining for adaptability

In Section 5.3.1 we retrained a new misuse detection model after learning characteristics of the 4 "funny packet" intrusions. Here we describe our experiment on combining the old model with the new model that was trained on these 4 new intrusions.

In the training data for the old model, we removed the connection records of the 4 "funny packet" intrusions to simulate the condition that these intrusion are "new" (i.e., unobserved) and thus are absent from the old model data. We want to test the idea that new models can be specialized in detecting new intrusions. Each connection record for the new model includes the flags for fragmentation and "urgent" packets, but not any of the temporal and statistical features included in the old model, since they are irrelevant for these 4 intrusions. We still include all the attack types in the training data of the new model because we need to see whether the new features can be useful in detecting the old intrusions (i.e., the "scan and flood" attacks).

A meta-learning process is used to generate the combined detection model. Both the old model and new model are applied to a common set of "hold-out" connection records that are not used in training the models. Each meta-level connection record has the outputs (i.e., predictions) from the old model and the new model, plus the correct class label, i.e., normal or an intrusion name. Table V shows examples of the meta-level records. The resultant RIPPER rules, as expected, use the predictions from the new model for the new intrusions and the predictions from the old model for the old intrusions. Table VI shows some example meta-level RIPPER rules.

Table VI. Example Meta Rules

| RIPPER rule | Meaning |
|---|---|
| teardrop :- new_model_prediction = teardrop. | if the prediction from the new model is *teardrop*, then this connection is a *teardrop* attack. |
| syn_flood :- old_model_prediction = syn_flood. | if the prediction from the old model is *syn flood*, then this connection is a *syn flood* attack. |
| ... | ... |
| normal :- true. | if none of the above, it is a normal connection. |

Note that although the new model can also detect the "scan and flood" intrusions, e.g., syn flood, the rules, e.g., "syn_flood :- flag = S0", have high false positive rates because the new model doesn't have the temporal and statistical features that are essential in detecting the "scan and flood" intrusions. The combined model uses the predictions from the old model to detect these intrusions because the old model rules are more accurate.

The combined detection model has slightly better performance than the updated model re-trained using the entire dataset in Section 5.3.1. Here the false negative rate dropped to 0.18% (compared to 0.2%) while the false positive rate remained the same. This experiment shows that when new intrusions can be detected using a new set of features, we can train a specialized new detection model. The meta-classifier that combines the existing detection model and the new model can be just as effective as an updated model that is retrained using the updated audit data set and feature set.

But what about new intrusions that require no new features? We conducted similar meta-learning experiments on the 5 "scan and flood" intrusions. The "updated" detection model on all 5 intrusions has better performance than the meta-classifier that combines the "new" model specialized on "syn flood" and the "existing" model on the other 4 "scan and flood" intrusions. The meta-classifier has a false negative rate of 1.1%, compared with 0.6% of the detection model on the 5 intrusions; while the false positive rates are the same. We inspected the "existing" model and the "updated" model and compared the rules regarding the 4 intrusions that are common to both models. The rules in the "updated" model include more distinguishing features useful for more accurate classification. This is because with an additional class (i.e., intrusion type) of records in the training data, the machine learning algorithm has more information to make better choices on the features that can set the intrusions and normal records apart. Therefore, when the accuracy of a detection model is critical, it is more desirable to retrain the model for the new intrusions that are detectable using the existing features.

We have only begun studying this issue and an extensive set of additional experiments are needed to elucidate the tradeoff in this approach.

## 7. Related work

Recently there have been quite a few new research efforts in intrusion detection. In (Forrest et al., 1996) it was shown that the short sequences of run-time system calls of a system program can be used as its "identity". To analyze a program, its short sequences of system calls are looked up in the "normal" database that contains the exhaustively gathered short sequences from its normal runs. A large degree of mismatches suggests that the program is attacked (i.e., its weakness is exploited). Here the "normal" database is used as an anomaly detection model. While this approach seems simple, it is only suited for system programs. The number of possible execution paths of a program is perhaps limited, and therefore one can in theory build a "normal" database that exhausts all possible sequences. However network traffic patterns and user behavior are in general unlimited (i.e., they can shift and evolve). The approach of using association rules and frequent episodes and measuring their growth might provide a sufficient indicator to detect when the "normal" has changed.

In (Lane and Brodley, 1997), algorithms for modeling user behavior were discussed. The idea is to match the user command sequences with established profiles. Various heuristics for partial string matching were introduced. This approach is also very limited. For example, the algorithms assume that user commands (i.e., (*command*, *options*, *arguments*) tuples ) are first collapsed into a single dimension string of "*command options arguments*". Converting the multi-attribute connection or session records into a single attribute string is not appropriate since each attribute carries distinct and yet important information. Section 5.2.2 demonstrates that it is important to be able to analyze the data using different combinations of the attributes.

In DC-1 (Detector Constructor) (Fawcett and Provost, 1997), a rule learning step is first used to obtain each customer's fraudulent patterns, and rule selection is then used to obtain a set of general fraudulent patterns for the entire population. A monitor construction step is used to obtain the sensitivity measures of different users to these general patterns. The final detector, a classifier, is then trained using records where the features are the outputs of monitors (i.e., sensitivity measures). We face here greater challenges in feature selection and construction. DC-1 assumes call data with a set of base-level attributes. Here we need to carefully preprocess the raw audit data into meaningful connection or session records. DC-1 does not need to consider the temporal and statistical features from the base-level attributes in call data, which are critical for an effective network intrusion detection model.

## 8. Conclusion

In this paper we outlined our approach for building intrusion detection models. We proposed that association rules and frequent episodes from the audit data can be used to guide audit data gathering and feature selection, the critical steps in building effective classification models. We incorporated domain knowledge into these basic algorithms using the *axis* and *reference* attribute(s), and a *level-wise* approximate mining procedure. Our experiments on real-world audit data showed that the algorithms are effective. Intrusion detection models need to be adaptive to changing environments and new intrusion methods. We show preliminary but promising results in using meta-learning as a mechanism to incrementally incorporate new models into the overall intrusion detector.

Several tools and steps in our framework, for example, using frequent episode programs to find specific patterns, are not fully automatic (e.g., we need to manually inspect the patterns). We need to provide support for rule templates (Klemettinen et al., 1994) so that patterns can be post-processed and presented as query results to users.

It is important to include users in the knowledge discovery tasks. We are implementing a support environment that integrates the iterative processes of selecting features, and building and evaluating classification models. This environment graphically presents the patterns along with the list of features and the time windowing information to the user, and allows him/her to formulate a classification task, build and test the model using a classification engine such as JAM (Stolfo et al., 1997).

## 9. Acknowledgments

We are very grateful to Stephanie Forrest of University of New Mexico for providing us with the *sendmail* system call data, and Stuart McClure of InfoWorld for supplying us with the IWSS16 data set.

Our work has benefited from in-depth discussions with Alexander Tuzhilin of New York University, and suggestions from Charles Elkan of UC San Diego and Foster Provost of Bell Atlantic Science and Technology. We also wish to thank Phil Chan of Florida Institute of Technology, Andreas Prodromidis and Wei Fan, both of Columbia University, for their help and encouragement.

## References

Agrawal, R., T. Imielinski, and A. Swami: 1993, 'Mining Association Rules between Sets of Items in Large Databases'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. pp. 207–216.

Agrawal, R. and R. Srikant: 1994, 'Fast Algorithms for Mining Association Rules'. In: *Proceedings of the 20th VLDB Conference*. Santiago, Chile.

Agrawal, R. and R. Srikant: 1995, 'Mining Sequential Patterns'. In: *Proceedings of the 11th International Conference on Data Engineering*. Taipei, Taiwan.

Bellovin, S. M.: 1989, 'Security Problems in the TCP/IP Protocol Suite'. *Computer Communication Review* **19**(2), 32–48.

Chan, P. K. and S. J. Stolfo: 1993, 'Toward Parallel and Distributed Learning by Meta-Learning'. In: *AAAI Workshop in Knowledge Discovery in Databases*. pp. 227–240.

Cohen, W. W.: 1995, 'Fast Effective Rule Induction'. In: *Machine Learning: the 12th International Conference*. Lake Taho, CA.

Fawcett, T. and F. Provost: 1997, 'Adaptive Fraud Detection'. *Data Mining and Knowledge Discovery* **1**, 291–316.

Forrest, S., S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff: 1996, 'A Sense of Self for Unix Processes'. In: *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. Los Alamitos, CA, pp. 120–128.

Grampp, F. T. and R. H. Morris: 1984, 'Unix System Security'. *AT&T Bell Laboratories Technical Journal* **63**(8), 1649–1672.

Han, J. and Y. Fu: 1995, 'Discovery of Multiple-Level Association Rules from Large Databases'. In: *Proceedings of the 21th VLDB Conference*. Zurich, Switzerland.

Heady, R., G. Luger, A. Maccabe, and M. Servilla: 1990, 'The Architecture of a Network Level Intrusion Detection System'. Technical report, Computer Science Department, University of New Mexico.

Ilgun, K., R. A. Kemmerer, and P. A. Porras: 1995, 'State Transition Analysis: A Rule-Based Intrusion Detection Approach'. *IEEE Transactions on Software Engineering* **21**(3), 181–199.

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

Klemettinen, M., H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo: 1994, 'Finding Interesting Rules from Large Sets of Discovered Association Rules'. In: *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*. Gainthersburg, MD, pp. 401–407.

Kumar, S. and E. H. Spafford: 1995, 'A Software Architecture to Support Misuse Intrusion Detection'. In: *Proceedings of the 18th National Information Security Conference*. pp. 194–204.

Lane, T. and C. E. Brodley: 1997, 'Sequence Matching and Learning in Anomaly Detection for Computer Security'. In: *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*. pp. 43–49.

Lee, W. and S. J. Stolfo: 1998, 'Data Mining Approaches for Intrusion Detection'. In: *Proceedings of the 7th USENIX Security Symposium*. San Antonio, TX.

Lee, W., S. J. Stolfo, and K. W. Mok: 1998, 'Mining Audit Data to Build Intrusion Detection Models'. In: *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. New York, NY.

Lee, W., S. J. Stolfo, and K. W. Mok: 1999, 'Mining in a Data-flow Environment: Experience in Network Intrusion Detection'. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99)*.

Lent, B., A. Swami, and J. Widom: 1997, 'Clustering Association Rules'. In: *Proceedings of the 13th International Conference on Data Engineering*. Birmingham, UK.

Lunt, T.: 1993, 'Detecting Intruders in Computer Systems'. In: *Proceedings of the 1993 Conference on Auditing and Computer Technology*.

Lunt, T., A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey: 1992, 'A Real-time Intrusion Detection Expert System (IDES) - final technical report'. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California.

Mannila, H. and H. Toivonen: 1996, 'Discovering Generalized Episodes Using Minimal Occurrences'. In: *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*. Portland, Oregon.

Mannila, H., H. Toivonen, and A. I. Verkamo: 1995, 'Discovering Frequent Episodes in Sequences'. In: *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*. Montreal, Canada.

McClure, S., J. Scambray, and J. Broderick: 1998, 'Test Center Comparison: Network Intrusion-detection Solutions'. In: *INFOWORLD May 4, 1998*.

Srikant, R., Q. Vu, and R. Agrawal: 1997, 'Mining Association Rules with Item Constraints'. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. Newport Beach, California, pp. 67–73.

Stolfo, S. J., A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan: 1997, 'JAM: Java Agents for Meta-Learning over Distributed Databases'. In: *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. Newport Beach, CA, pp. 74–81.

Utgoff, P. E., N. C. Berkman, and J. A. Clouse: 1997, 'Decision Tree Induction Based on Efficient Tree Restructuring'. *Machine Learning* **29**, 5–44.