Mining Audit Data to Build Intrusion Detection Models*

Wenke Lee and **Salvatore J. Stolfo** and **Kui W. Mok**
Computer Science Department
Columbia University
500 West 120th Street, New York, NY 10027
{wenke,sal,mok}@cs.columbia.edu

Abstract

In this paper we discuss a data mining framework for constructing intrusion detection models. The key ideas are to mine system audit data for consistent and useful patterns of program and user behavior, and use the set of relevant system features presented in the patterns to compute (inductively learned) classifiers that can recognize anomalies and known intrusions. Our past experiments showed that classifiers can be used to detect intrusions, provided that sufficient audit data is available for training and the right set of system features are selected. 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)* as a form of item constraints to compute only the relevant ("useful") patterns, and an iterative *level-wise* approximate mining procedure to uncover the low frequency (but important) patterns. We report our experiments in using these algorithms on real-world audit data.

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. Intrusion prevention techniques, such as user authentication (e.g. using passwords or biometrics), are not sufficient because as systems become ever more complex, there are always system design flaws and programming errors that can lead to security holes (Bellovin 1989). Intrusion detection is therefore needed as another wall to protect computer systems. There are mainly two types of intrusion detection techniques. *Misuse detection*, for example STAT (Ilgun, Kemmerer, & Porras 1995), uses patterns of well-known attacks or weak spots of the system to match and identify intrusions. *Anomaly detection*, for example IDES (Lunt *et al.* 1992), tries to determine whether deviation from the established normal usage patterns can be flagged as intrusions.

Currently many intrusion detection systems are constructed by manual and ad hoc means. In misuse detection systems, intrusion patterns (for example, *more than three consecutive failed logins*) need to be hand-coded using specific modeling languages. In anomaly detection systems, the features or measures on audit data (for example, the *CPU usage* by a program) that constitute the profiles are chosen based on the experience of the system builders. As a result, the effectiveness and adaptability (in the face of newly invented attack methods) of intrusion detection systems may be limited.

Our research aims to develop a systematic framework to semi-automate the process of building intrusion detection systems. A basic premise 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. We therefore take a data-centric point of view and consider intrusion detection as a data analysis task. Anomaly detection is about establishing the normal usage patterns from the audit data, whereas misuse detection is about encoding and matching intrusion patterns using the audit data. We are developing a framework, first described in (Lee & Stolfo 1998), of applying data mining techniques to build intrusion detection models. This framework consists of classification (and meta-classification (Chan & Stolfo 1993)), association rule (Agrawal, Imielinski, & Swami 1993) and frequent episode (Mannila, Toivonen, & Verkamo 1995) 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 product is concise and intuitive classifica-

*This research is supported in part by grants from DARPA (F30602-96-1-0311) and NSF (IRI-96-32225 and CDA-96-25374).

tion rules that can detect intrusions.

The rest of the paper is organized as follows. We first examine the lessons we learned from our past experiments on building classification models for detecting intrusions, namely we need tools for feature selection and audit data gathering. We then propose a framework and discuss how to incorporate domain knowledge into the association rules and frequent episodes algorithms to discover "useful" patterns from audit data. We report our experiments in using the patterns both as a guideline for gathering "sufficient" training (audit) data, and as the basis for feature selection. Finally we outline open problems and our future research plans.

The Challenges

In (Lee & Stolfo 1998) we described in detail our experiments in building classification models to detect intrusions to *sendmail* and TCP/IP networks. The results on the *sendmail* system call data showed that we needed to use as much as 80% of the (exhaustively gathered) normal data to learn a classifier (RIPPER (Cohen 1995) rules) that can clearly identify normal *sendmail* executions and intrusions. The results on the *tcpdump* (Jacobson, Leres, & McCanne 1989) of network traffic data showed that by the temporal nature of network activities, when added with temporal statistical features, the classification model had a very significant improvement in identifying intrusions. These 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 mechanisms and data format specifically for intrusion analysis purposes. 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 statistical measures over the features (for example, *the average duration of connections in the past 30 seconds*) need to be considered as additional features. Traditional feature selection techniques, as discussed in the machine learning literature, cannot be directly applied here since they don't consider (across record boundary) sequential correlation of features. In (Fawcett & Provost 1996) Fawcett and Provost presented some very interesting ideas on automatic selection of features for a cellular phone fraud detector. An important assumption in that work is that there are some general patterns of fraudulent usage for the entire customer population, and individual customers differ in the "threshold" of these patterns. Such assumptions do not hold here since different intrusions have different targets on the computer system and normally produce different evidence (and in different audit data sources).

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 limited ("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 (Utgoff, Berkman, & Clouse 1997), can't deal with new data with new (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. This indicator should be simple to compute and must be incrementally updated.

Mining Audit Data

We attempt 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 (intuitively) capture the intra- and (temporal) inter- audit record patterns. These patterns are then utilized, with user participation, to guide the data gathering and feature selection processes.

The Basic Algorithms

From (Agrawal, Imielinski, & Swami 1993), let $\mathcal{A}$ be a set of attributes, and $\mathcal{I}$ be a set of values on $\mathcal{A}$, called items. Any subset of $\mathcal{I}$ 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 (which is a stream of commands and their arguments) of a user 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 & Srikant 1994).

The problem of finding frequent episodes based on minimal occurrences was introduced in (Mannila &

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 & Toivonen 1996) in that we don't consider a separate $window$ constraint on the LHS (left hand side) of the rule. Our implementation of the frequent episodes algorithm utilized the data structures and library functions of the association rules algorithm.

Using the *Axis* Attribute(s)

These basic algorithms do not consider any domain knowledge and as a result they can generate many "irrelevant" rules. In (Klemettinen *et al.* 1994) rule templates specifying the allowable attribute values are used to post-process the discovered rules. In (Srikant, Vu, & Agrawal 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 what rules/patterns are interesting. We cannot assume such strong prior knowledge on all audit data.

We use *axis* attribute(s) as a form of item constraints. Intuitively the axis attribute(s) is the essential attribute(s) of a record (transaction). We consider the correlations among non-axis attributes as not interesting. During candidate generation, an itemset must contain value(s) of the axis attribute(s). Consider the following audit data of network connections (with some attributes omitted):

time	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
3.7	2	smtp	300	200	SF
...					

Here we already discretize the continuous attribute values (except the timestamps) into proper bins. The basic association rules algorithm may generate rules such as $src\_bytes = 200 \rightarrow flag = SF$. These rules are not useful and to some degree are misleading. There is no intuition for the association between the number of bytes from the source, *src_bytes*, and the normal status ($flag$=SF) of the connection. Since the most important information of a connection is its service, we use it as the axis attribute. The resulting association rules then describe only 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, e.g., *src_bytes*=200, is from a different 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 (that is, the length of an itemset) is bounded here by the number of records (instead of the number of attributes as in association rules), which is usually 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 in the database, the time difference between 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 $(A|B)$). Further, assume that the records with $A \cup B$ are evenly distributed, then the width of the interval is at most $w = \frac{kt}{sm}$. There can be a large number of serial patterns on $(A|B)$ if s is high and $window$ (the interval width threshold) is large, since k_{max}, the maximal length of the patterns, can be large and $w \leq window$ still holds.

Instead of using the basic algorithm, here we first find the frequent associations using the axis attribute(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). Here we in effect have combined the associations (among attributes) and the sequential patterns (among the records) into a single rule. This rule formalism not only eliminates irrelevant rules, it also provides rich and useful information about the audit data.

Level-wise Approximate Mining

Sometimes it is important to discover the low frequency patterns. In daily network traffic, some services, for example, *gopher*, account for very low occurrences. 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

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

Figure 1: Level-wise Approximate Mining of Frequent Episodes

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). 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 an episode is generated, it must contain at least one "new" (low frequency) axis value. For example, in the second iteration (where *smtp* now is an old axis value) we get an episode rule (*service* = *smtp*, *src_bytes* = 200), (*service* = *http*, *src_bytes* = 200) $\rightarrow$ (*service* = *smtp*, *src_bytes* = 300). 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 values) because they are not as interesting (representative). Hence our procedure is "approximate" mining. We still include all the old (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 into the episode rules about *auth*.

Our approach here is different from the algorithms in (Han & 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.

Using the Mined Patterns

In this section we report 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 statistical features.

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 about the target, we compute the patterns from each new audit data set, and *merge* the new rules into the existing aggregate rule set. The 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.

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

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 ε (a user-defined threshold). The concept of *combining* here is similar to *clustering* in (Lent, Swami, & Widom 1997). 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 com-

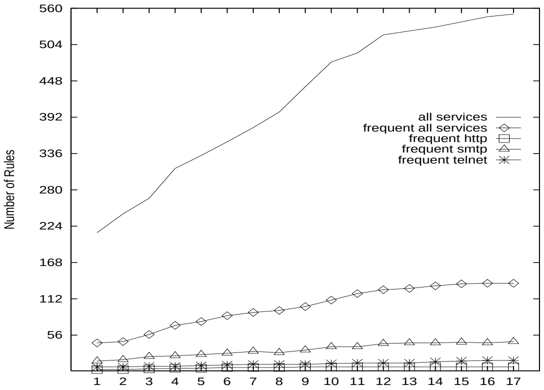

Figure 2: The Number of Rules vs. The number of Audit Data Sets

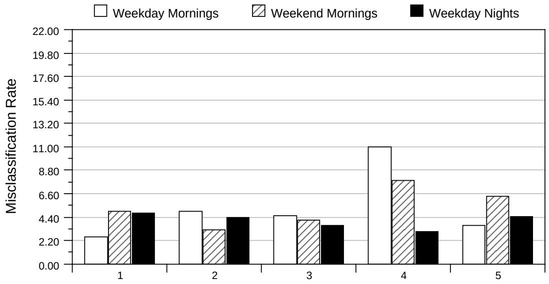

Figure 3: Misclassification Rates of Classifier Trained on First 8 Weekdays

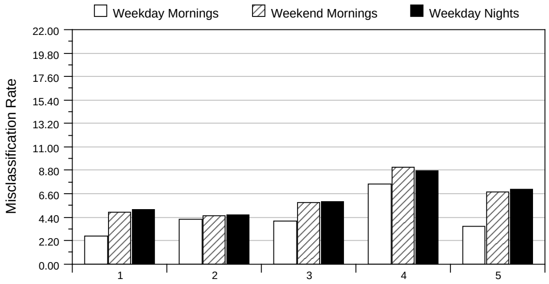

Figure 4: Misclassification Rates of Classifier Trained on First 10 Weekdays

bined, 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*, *src_bytes*=300) can be combined to (*service*=*smtp*, $200 \leq$ *src_bytes* ≤ 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.

Experiments Here we test our hypothesis that the merged rule set can indicate whether the audit data has covered sufficient variations of behavior. We obtained one month of TCP/IP network traffic data from "http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html" (there are total about 780,000 connection records). We segmented the data by day. And for 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 (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.

Here we describe the experiments and results on building anomaly detection models for the "weekday morning" traffic data on connections originated from LBL to the outside world (there are about 137,000 such connections for the month). We decided to compute the frequent episodes using the network *service* as the axis attribute. Recall from our earlier discussion that this formalism captures both association and sequential patterns. For the first three weeks, we mined the patterns from the audit data of each weekday morning, and merged them into the aggregate rule set. For each rule we recorded *merge_count*, the number of merges on this rule. Note that if two rules r_1 and r_2 are merged into r, its *merge_count* is the sum from the two rules. *merge_count* indicates how frequent the behavior represented by the merged rule is encountered across a period of time (days). We call the rules with *merge_count* $\geq$ *min_frequency* the 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 keeps increasing. We visually inspected the new rules 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 rules of the existing services. Figure 2 shows that the rate of change slows down during the last week. Further, when we examine the frequent rules (here we used $min\_frequency = 2$ to filter out the "one-time" patterns), we can see in the figure that the rule sets (of all services as well as the individual services) grow at a much slower rate and tend to stabilize.

We used the set of frequent rules of all services as the indicator on whether the audit data is sufficient. 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. We used the services of the connections as the class labels, and included a number of temporal statistical features (the details of feature selection is discussed in the next session). The classifiers were tested using the audit data (not used in training) from the mornings and nights of

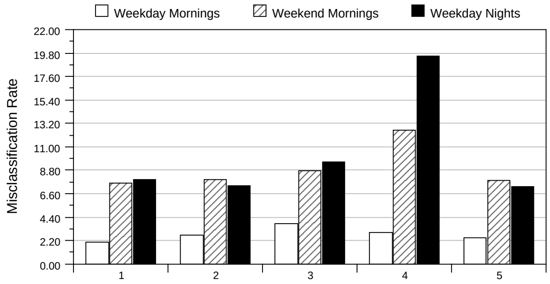

Figure 5: Misclassification Rates of Classifier Trained on First 15 Weekdays

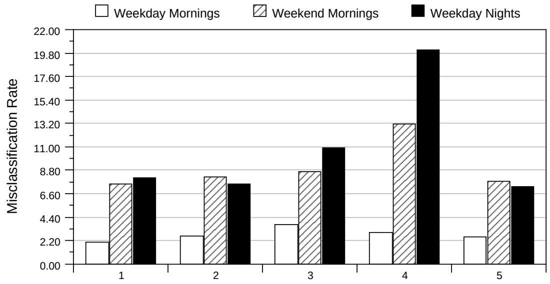

Figure 6: Misclassification Rates of Classifier Trained on First 17 Weekdays

the last 5 weekdays of the month, as well as the last 5 weekend mornings. Figures 3, 4, 5 and 6 show the performance of these four classifiers in detecting anomalies (different behavior) respectively. In each figure, we show the misclassification rate (percentage of misclassifications) on the test data. Since the classifiers model the weekday morning traffic, we wish to see this rate to 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 (audit) data perform better. Further, the last two classifiers are effective in detecting anomalies, and their performance are very close (see figures 5 and 6). This is not surprising at all because from the plots in figure 2, the set of frequent rules (our indicator on audit data) is growing in weekday 8 and 10, but stabilizes from day 15 to 17. Thus this indicator on audit data gathering is quite reliable.

Feature Selection

An important use of the mined patterns is as the basis for feature selection. When the axis attribute is used as the class label attribute, features (the attributes) in the association rules should be included in the classification models. The time windowing information and the features in the frequent episodes suggest that their statistical measures, e.g., the average, the count, etc., should also be considered as additional features in an anomaly detector.

Experiments We examined the frequent rules from the audit data to determine what features should be used in the classifier. Here when the same value of an attribute is repeated several times in a frequent episode rule, it suggests that we should include a corresponding *count* feature. For example given ($service = smtp$, $src\_bytes = 200$), ($service = smtp$, $src\_bytes = 200$) $\rightarrow$ ($service = smtp$, $src\_bytes = 200$) $0.81, 0.42, 140$, we add a feature, the count of connections that have the same *service* and *src_bytes* as the current connection record in the past 140 seconds. When an attribute (with different values) is repeated several times in the rule, we add a corresponding *average* feature. For example, given ($service = smtp$, $duration = 2$), ($service = telnet$, $duration = 10$) $\rightarrow$ ($service = http$, $duration = 1$), we add a feature, the average *duration* of all connections in the past 140 seconds. The classifiers in the previous section included a number of temporal statistical features: the count of all connections in the past 140 seconds, the count of connections with the same *service* and the same *src_bytes*, the average *duration*, the average *dst_bytes*, etc. Our experiments showed that when using none of the temporal statistical features, or using just the *count* features or *average* features, the classification performance was much worse.

In (Lee & Stolfo 1998) we reported that as we mined frequent episodes using different window sizes, the number of serial episodes stabilized after the time window reached 30 seconds. We showed that when using 30 seconds as the time interval to calculate the temporal statistical features, we achieved the best classification performance. Here, we sampled the weekday morning data and discovered that the number of episodes stabilized at 140 seconds. Hence we used it as the *window* in mining the audit data and as the time interval to calculate statistical features.

Off-line Analysis

Since the merged rule set was used to (identify and) collect "new" behavior during the audit data gathering process, one naturally asks "Can the final rule set be directly used to detect anomalies?". Here we used the set of frequent rules to distinguish the traffic data of the last 5 weekday mornings from the last 5 weekend mornings, and the last 5 weekday nights. We use a *similarity* measure. Assume that the merged rule set has n rules, and the size of the new rule set from a new audit data set is m, 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 (all) behavior in the new audit data that conforms to the known behavior. Our experiments showed that the *similarity* of the weekday mornings are much larger than the weekend mornings and the weekday nights. However in general these mined patterns cannot be used directly to classify the records (i.e., they cannot tell *which* records are anomalous). They are very valuable in off-line analysis. For example, we built a graphical tool to visually show the differences between rule sets. We can easily identify the different behavior across audit data sets.

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* attribute(s) and a *level-wise* approximate mining procedure. Our experiments on real-world audit data showed that the algorithms are effective.

To the best of our knowledge, our research was the first attempt to develop a systematic framework for building intrusion detection models. We plan to refine our approach and further study some fundamental problems. For example, can we automatically select the optimal set of temporal statistical features? Are classification models best suited for intrusion detection (i.e. what are the better alternatives)?

It is important to include system designers in the knowledge discovery tasks. We are implementing a support environment that graphically presents the mined patterns along with the list of features and the time windowing information to the user, and allows him/her to (iteratively) formulate a classification task, build and test the model using a classification engine such as JAM (Stolfo *et al.* 1997).

Acknowledgments

Our work has benefited from in-depth discussions with Alexander Tuzhilin of New York University, and suggestions from Charles Elkan of UC San Diego.

References

Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*.

Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 207–216.

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

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

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

Fawcett, T., and Provost, F. 1996. Combining data mining and machine learning for effective user profiling. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 8–13. Portland, OR: AAAI Press.

Han, J., and Fu, Y. 1995. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21th VLDB Conference*.

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

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

Klemettinen, M.; Mannila, H.; Ronkainen, P.; Toivonen, H.; and Verkamo, A. I. 1994. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, 401–407.

Lee, W., and Stolfo, S. J. 1998. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*.

Lent, B.; Swami, A.; and Widom, J. 1997. Clustering association rules. In *Proceedings of the Thirteenth International Conference on Data Engineering*.

Lunt, T.; Tamaru, A.; Gilham, F.; Jagannathan, R.; Neumann, P.; Javitz, H.; Valdes, A.; and Garvey, T. 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 Toivonen, H. 1996. Discovering generalized episodes using minimal occurences. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1995. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery in Databases and Data Mining*.

Srikant, R.; Vu, Q.; and Agrawal, R. 1997. Mining association rules with item constraints. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 67–73. Newport Beach, California: AAAI Press.

Stolfo, S. J.; Prodromidis, A. L.; Tselepis, S.; Lee, W.; Fan, D. W.; and Chan, P. K. 1997. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 74–81. Newport Beach, CA: AAAI Press.

Utgoff, P. E.; Berkman, N. C.; and Clouse, J. A. 1997. Decision tree induction based on efficient tree restructuring. *Machine Learning* 29:5–44.