A Data Mining Framework for Adaptive Intrusion Detection*

Wenke Lee Salvatore J. Stolfo

Kui W. Mok

Computer Science Department

Columbia University

500 West 120th Street, New York, NY 10027

{wenke,sal,mok}@cs.columbia.edu

June 30, 1998

Abstract

In this paper we describe 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 experiments on audit data of system programs and network activities showed that classification models can 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.

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

We modify these two basic algorithms to use *axis attribute(s)* and *variable attribute(s)* as forms 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 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

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 [HLMS90] 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 (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 weakness 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 [Bel89, GM84]; 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 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 a deep understanding of known intrusions. As a result, the effectiveness and adaptability of the intrusion detection systems are limited in the face of new computing environ-

ments 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 are 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 statistical features for another phase of automated learning. Classifiers based on these selected features are then computed (inductively learned) using the (appropriate formatted) audit data. These classifiers can be used as intrusion detection models since they can classify (decide) whether an observed system activity is "legitimate" or "intrusive".

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. 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 incorporate general domain knowledge into the association rules and frequent episodes algorithms in order to compute "useful" patterns from audit data. 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. We also present our experimental results on combining multiple classification models for collaborative intrusion detection. Finally, 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*. Most current intrusion detection systems use one or both of these two approaches.

2.1 Anomaly Detection

Anomaly detection, for example IDES [LTG+92], 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 login session (of a user). Deviation from a profile can be computed as

the weighted sum of the deviations of the constituent statistical measures. Profiles can be updated periodically (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.

2.2 Misuse Detection

Misuse detection refers to techniques that use patterns of known intrusions (for example, *more than three consecutive failed logins* within 2 minutes is a penetration attempt) or weak spots of a system (for example, 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 [Lun93] uses rules, STAT [IKP95] uses state transition diagrams [IKP95], and [KS95] uses Colored Petri nets. 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.

2.3 Difficulties in Building Intrusion Detection Systems

Currently there is no systematic approach for building intrusion detection systems. A system builders' intuition and experience guides the selection of the statistical measures for anomaly detection [Lun93]. 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 (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 na-

ture in understanding the system audit data, selecting statistical measures and modeling intrusion patterns.

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

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 [LS98], of applying data mining techniques to build intrusion detection models. This framework consists of classification (and meta-classification [CS93]), association rule [AIS93] and frequent episode [MTV95] 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 are concise and intuitive classification rules (that can be easily inspected and edited by security experts when needed) that can detect intrusions.

We try to eliminate, as much as possible, the manual and ad hoc elements. 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 (meta-classification) [CS93] 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 are generic and mechanical, it is much easier to update the intrusion detection models using our framework. Migration to a new computing environments or dealing with new intrusion methods means applying the data mining and machine learning process to the relevant audit data. 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.

3 Classification Models for Intrusion Detection

In this section we briefly summarize our early experiments on constructing classification models for anomaly detection (detailed in [LS98]). 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 (for example, *open*, *read*, ...) of a program provides very detailed information about its execution. We obtained from Stephanie Forrest of University of New Mexico such 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 [FHSL96], we preprocessed the data by using a length n sliding window to scan the system call sequence and create a list of records, each of which has n consecutive system calls. The nth system call of each record is the class label, and the first n-1 system calls are the (positional) attributes. RIPPER [Coh95] was then applied to learn the classification rules from the normal *sendmail* data. The resultant rule set, consisting of about 250 rules each with 2 or 3 attribute 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 (deviations) suggests an "anomaly". Our experiments (detailed in [LS98]) 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 hackers. These intrusions are aimed at disrupting network services. "Packet sniffing" programs, such as *tcpdump* [JLM89], can capture traffic data for analysis and monitoring. We obtained a set of *tcpdump* data on network traffic, available via http at "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 (whether there is an error according to the protocol) as the attributes (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 (deviations from the profile) on the normal testing data and the intrusions are very close. This is because here the temporal nature of network traffic is not captured in the classification model. For each connection we then added several temporal 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 [LS98].

3.1 The Challenges

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 afterthought of computer system design, there is no standard auditing mechanism and data format specifically 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 statistical measures over the features can be very valuable (additional) features (for example, *the total number of connection failures in the past 10 seconds*). 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). [FP97] 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 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 (Incremental Tree Induction) [UBC97], 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.

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 (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. 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 the Web site logs. We assume that audit data records are timestamped.

As described in [LSM98], 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 [AIS93], 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 [AS94].

Since we look for correlation among values of different features (attributes), and the (pre-processed) audit data usually has multiple columns of features, 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 (bit) vector* that records the transactions in which the itemset is contained. When a length k candidate itemset c_k is generated by joining two length k-1 frequent item sets l^1_{k-1} and l^2_{k-1}, the row vector of c_k is simply the bitwise AND product of the row vectors of l^1_{k-1} and l^2_{k-1}. The *support* of c_k 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 our application domain.

The problem of finding frequent episodes based on minimal occurrences was introduced in [MT96]. 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 [MT96] 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. 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.

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
...	...	...	...	...	...

Table 1: Network Connection Records

4.2 Using the *Axis* Attributes(s)

These basic algorithms does not consider any domain knowledge and as a result it can generate many "irrelevant" rules. In [KMR+94] rule templates specifying the allowable attribute values are used to post-process discovered rules. In [SVA97] 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 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 audit data of network connections shown in Table 1.

Here we already discretize 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 some degree 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 service attributes 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 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 (that is, the length of an itemset) is bounded here by the number of rows (instead of the number of columns 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 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 $window$ (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 ($service = smtp$, $src\_bytes = 200$, $dst\_bytes = 300$, $flag = SF$), ($service = telnet$, $flag = SF$) $\rightarrow$ ($service = http$, $src\_bytes = 200$). Here we in effect have combined the associations (among attributes) and the sequential patterns (among the records) into a single rule. This rule formalism provides rich and useful information about the audit data.

4.3 Using the *Variable* Attribute(s)

The basic algorithms produce propositional rules. In some situations, we may need to consider the first-order properties of the patterns. Although general first-order rules require completely different algorithms, we can still add restrictive yet useful first-order extensions to the basic algorithms. Suppose we want to study the behavior of visits to a Web site, where the (pre-processed) log

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
...	...	...	...

Table 2: Web Log Records

records are shown in Table 2.

Again, here we already pre-process the original data (and discretize the continuous attribute values). Suppose we are interested in the sequential *request* patterns by the (visiting) hosts. For example, the records in Table 2 show that requests "/images", "/images" and "/shuttle/missions/sts-71" are highly correlated since for a number of hosts, each of them makes the same sequence of requests. We therefore would like to have rules of the form: ($host = X$, $request = a$), ($host = X$, $request = b$) $\rightarrow$ ($host = X$, $request = c$). Here each *request* is associated with the **same** *host*, yet the actual *host* value may not be given in the rule (since any particular *host* value may not be frequent), hence *host* is a **variable**. We can not omit *host* from the rule since requests made by different hosts are in general irrelevant.

We extended the frequent episodes algorithm to consider *variable* attribute(s). Briefly, when forming an itemset (on non-*variable* attribute values), an additional condition is that, within the itemset's minimal occurrences (that is, a sequence of records), the value(s) of the *variable* attribute(s) must be equal.

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 $L = \{$lager 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

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, ($service = smtp$, $src\_bytes = 200$), ($service = smtp$, $src\_bytes = 200$) $\rightarrow$ ($service = smtp$, $src\_bytes = 200$). 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" (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 ($service = smtp$, $src\_bytes = 200$), ($service = http$, $src\_bytes = 200$) $\rightarrow$ ($service = smtp$, $src\_bytes = 300$). 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 value) 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 when presenting episode rules about *auth*.

Our approach here is different from the algorithms in [HF95] 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 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. 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.

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. The concept of $combining$ here is similar to $clustering$ in [LSW97]. 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 ($service$=$smtp$, src_bytes=300) can be combined to ($service$=$smtp$, $200 \leq src\_bytes \leq 300$). To compute the (statistical 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 perfect (exact) match 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 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 it as the LBL dataset). 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.

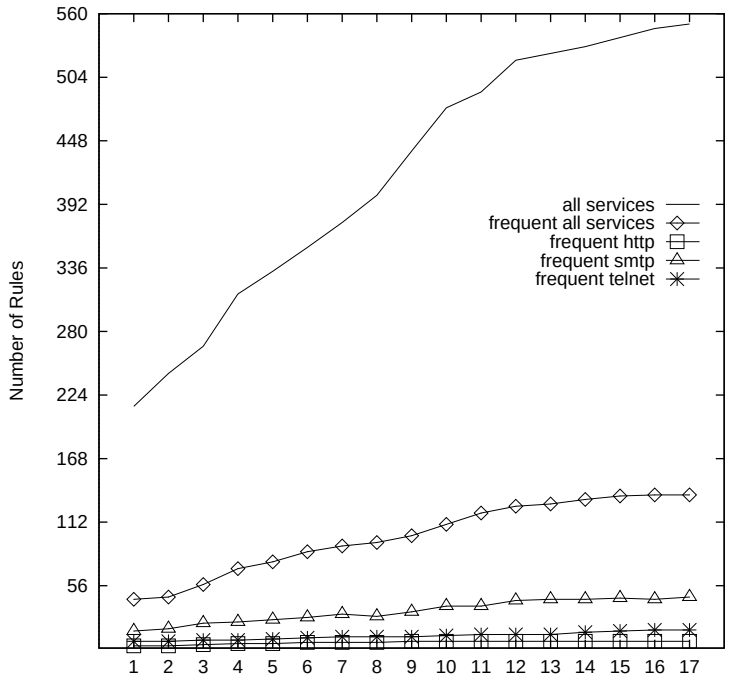
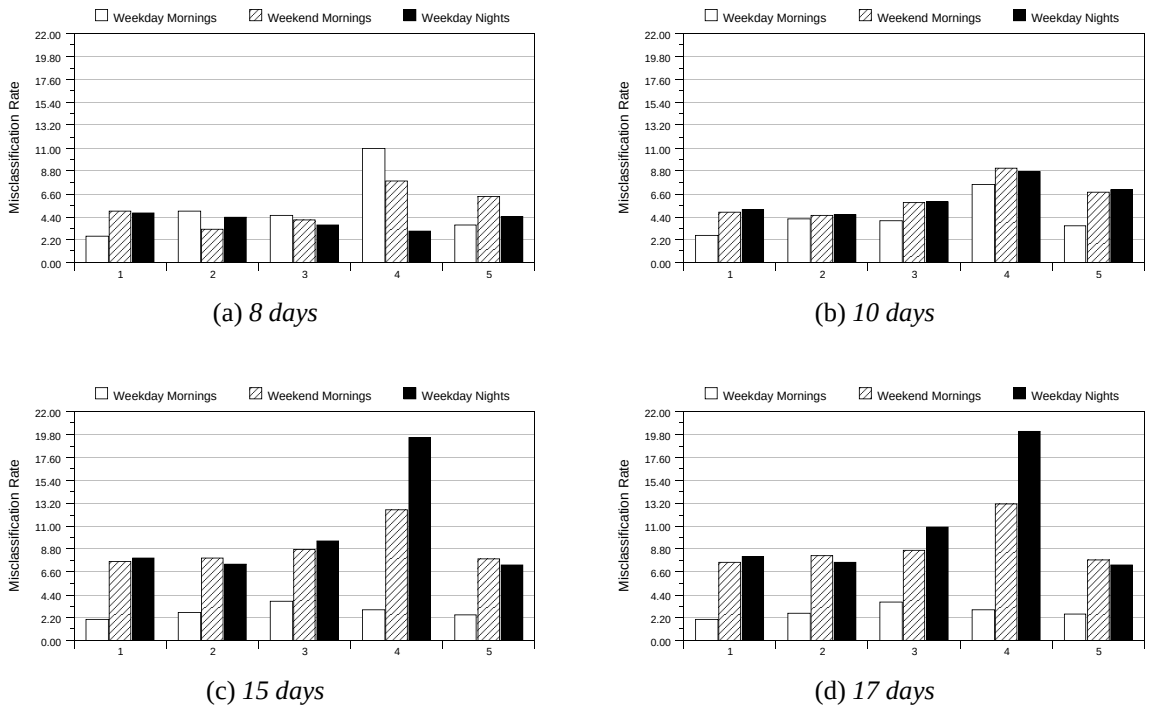


Figure 2: The Number of Rules vs. The number of Audit Data Sets

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 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 parent rules. $merge\_count$ indicates how frequent the behavior represented by the merged rule is encountered across a period of time (days). We say the rules with $merge\_count \geq min\_frequency$ as the 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 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



(a) *8 days* (b) *10 days*

(c) *15 days* (d) *17 days*

Figure 3: Misclassification Rates of Classifier Trained on First 8, 10, 15 and 17 Weekdays

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. 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 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 3 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 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 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 (our indicator on audit data) grows in weekday 8 and 10, but stabilizes from day 15 to 17.

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 (all) behavior in the new audit data that conforms to the known behavior. Figure 4 shows that in general the $similarity$ of the weekday mornings are much larger than the weekend mornings and the weekday nights.

Note that in general these mined patterns cannot be used directly to classify the records (i.e., they cannot tell *which* records are an anomaly). However they are very valuable in off-line analysis. We built a graphical tool to visually show the *diff* of two rule sets. When comparing the weekday morning rule set with a weekend morning, we see the later 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 in the weekend morning (so each machine is relatively frequent), whereas in the weekdays,

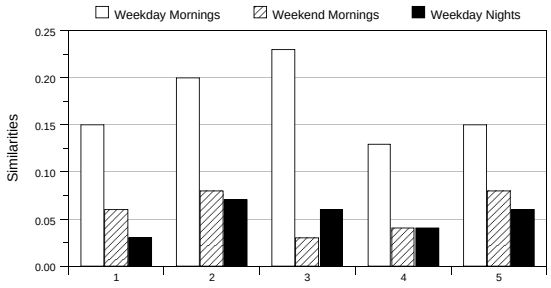


Figure 4: Similarity Measures Against the Merged Rule Set of Weekday Mornings

many machines used *telnet* (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 [MSB98]. (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 features such as *service*, *duration*, *flag*, etc., that is, no temporal statistical features were added.

According to their attack methods[1], 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 (test) the patterns of these attacks. Here we used a time window of 2 seconds.

- **Port Scan and Netsonar Scan:** The attacker systematically makes connections to each port (service) of a target host (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).

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

– Data mining strategy: use *dst_host* as both the *axis* attribute and the *variable* 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, ($flag = REJ$, $dst\_host = 207.217.205.23$) $\rightarrow$ ($flag = REJ$, $dst\_host = 207.217.205.23$); but no patterns with $flag = REJ$ are found when using the *service* as the *axis* attribute (and *dst_host* as the *variable* attribute) since a large number of difference services (ports) are attempted in a short period time. 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.

- **Ping Scan:** The attacker systematically sends ping (*icmp echo*) requests to a large number of different hosts to find out which host is available. In the connection records, there should be a host that makes *icmp echo* connections to many different hosts in a short period of time.

  – Data mining strategy: use *service* as the *axis* attribute and *src_host* as the *variable* attribute to find the "same source host" frequent sequential "service" patterns;

  – Evidence in intrusion data: there are several patterns that suggest the attack, for example, ($service = icmp\_echo$, $src\_bytes = 0$, $src\_host = 207.217.205.66$), ($service = icmp\_echo$, $src\_bytes = 0$, $src\_host = 207.217.205.66$) $\rightarrow$ ($service = icmp\_echo$, $src\_bytes = 0$, $src\_host = 207.217.205.66$). Note that there is no *dst_host* in this rule, suggesting that *icmp echo* is sent to "different" hosts;

  – Contrast with normal data: no such patterns.

- **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" (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 *variable* 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, ($service = http$, $flag = S0$), ($service = http$, $flag = S0$) $\rightarrow$ ($service = http$, $flag = S0$);
- Contrast with normal data: no patterns with $flag = S0$.

- **ICMP Flood:** The attacker sends a very large number of a *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 sends a lot of *icmp echo* connections to the same destination host in a short period of time.

  - Data mining strategy: use *service* as the *axis* attribute and both *src_host* and *dst_host* as the *variable* 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, ($service = icmp\_echo$, $src\_host = 207.217.205.6$, $dst\_host = 207.217.205.44$), ($service = icmp\_echo$, $src\_host = 207.217.205.6$, $dst\_host = 207.217.205.44$) $\rightarrow$ ($service = icmp\_echo$, $src\_host = 207.217.205.6$, $dst\_host = 207.217.205.44$);
  - 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

An important use of the mined patterns is as the basis for feature selection and model construction. It is obvious that the features (attributes) in the association rules should be included in the classification models. The time windowing information and the attributes 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. The classifiers briefly described 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 same *service* and the same *src_bytes*, the average of $duration$, the average of *dst_bytes*, etc. Our experiments showed that when using none of the temporal statistical features, or only the *count* features or *average* features, the classification performance was much worse.

In general raw action (e.g., packet or event) level data needs to be summarized into higher-level connection/session data using manually written preprocessing scripts before the data mining tasks. In some cases (as experiments described in Section 5.3.1 will show), we can use the patterns mined from the raw data to improve the preprocessing scripts because these patterns reveal the action-level characteristics, and can suggest what intrinsic system features need to be included into the connection or session level records.

5.3.1 Experiments on IWSS16 Dataset

We report our experiments of building misuse detection models for the IWSS16 attacks. We used the mined intrusion patterns described in Section 5.2.2 to construct features for the classification models. For each connection record we added these temporal 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 distinct (different) services;

- For connections that have the same service (as the current connection): the total number of such connections, the number of distinct source hosts, and the number of distinct destination hosts.

We used the intrusion types ("normal" or one of the IWSS16) as the class labels. 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 a few times, 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 ("normal" classified as "intrusion") rate of 1% and a false negative ("intrusion" classified as "normal") rate of 0.6% on the testing dataset. The rules were quite intuitive. For example, "if the service is *http*, the flag is S0, and (for the past 2 seconds) the number of connections that have S01 (S0 or S1) flags is greater then 7, then this connection is a syn flood".

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 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* (the connection to which the packet data belongs), *head*, and *tail*. Here for TCP protocols, *head* is the flag in the TCP header (for example "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 *variable* attribute (this is very important 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), and as a result, the false negative rate dropped to 0.2% [2].

There is a limit to any automate 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) with flag "S0" that appears to originate from and destinate to the same host and port. Since only a single "syn request" packet (with same host and port) is seen, it is rather difficult for our frequency-based mining algorithms to discover this pattern. "zone transfer" uses the TCP version of *dns* to transfer a large number of data packets. Its packet-level patterns are normal since many (normal) TCP connections (of other network services) also transfer a lot of data packets. We incorporated these "expert" knowledge to our connection record for the sake of completeness.

The last remaining four intrusions (of the IWSS16) 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 a separate part (access point) 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

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

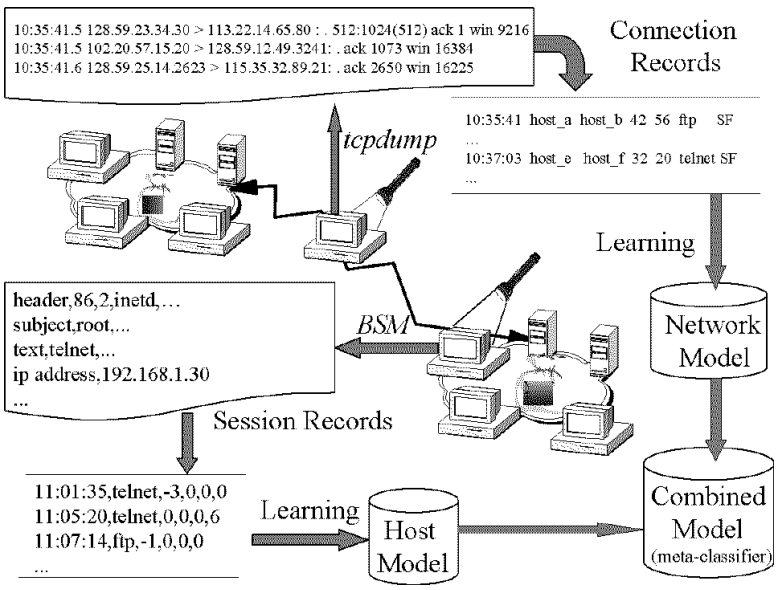


Figure 5: Combining Multiple Cooperative Intrusion Detection Models

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 may 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 actions. 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 example of such cooperative intrusion detection models is shown in Figure 5.

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 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 (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 the 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 old (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 (component detection models). Here in order to obtain the combined model (meta-classifier), meta-level connection records are constructed where the attributes are predictions (evidence) from the base (constituent) detection models, and the class labels are the original connection labels ("normal" or an intrusion). A machine learning algorithm can then be applied to the meta-level records to produce the combined detection model.

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 experimented 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" (to appear in the future) and thus are absent from the old model data. For the new model, each connection record includes the flags for fragmentation and "urgent" packets, but not any of the temporal statistical features (as in the old models) since they are irrelevant for these intrusions. This was done to test the idea that new models need only be specialized for the new intrusions. Our experiments showed that we need to be careful in preparing the training data for the new model. Although we only need to have the connection records generated by the 4 intrusions (and not any other intrusion records), we also need to include all the "normal" connection records that are part of the training data of the "old" model. Otherwise the new model will have a very high false positive rate.

Each meta-level connection record has the outputs (predictions) from the old model and the new model, plus the intrusion name as the class label. 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. For example:

> teardrop :- new=teardrop. [meaning: if the prediction from the new model is *teardrop*, then this connection is a *teardrop* attack]
>
> ...
>
> ping scan :- old=ping scan. [meaning: if the prediction from the old model is *ping scan*, then this connection is a *ping scan* attack]
>
> ...
>
> normal :- true. [meaning: if none of the above, it is a normal connection]

This 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 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 (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 attributes (features) that apparently are more effective for classification. This is because with an additional class (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.

6.2 Experiments: Combining for Effectiveness

As stated in Section 5.3.1, there are several intrusions in IWSS16 that attack the hosts rather than the network protocols. We can combine the network detection models and the host-based models to detect such intrusions. Here we experimented using a sample dataset from the DARPA Intrusion Detection Evaluation Dataset [3].

This DARPA sample dataset consists of *tcpdump* data and (host-based) BSM [Sun] audit data. The dataset contains only 15 minutes of activity with several intrusions, for example, "port scan", "ping scan" and "guessing password", etc. We built an anomaly detection model for the network *tcpdump* data, and a misuse detection model for the host BSM data. The BSM session records contain some special attributes for intrusion detection purposes, for example: the number of times the user/intruder tries password(s), a flag indicating whether the user/intruder successfully gains access, a flag indicating whether all processes are terminated upon exit, etc.

Using the network services as the class labels, the network detection model was able to detect only the "port scan" and "ping scan" intrusions (because of anomalies, e.g., an actual "ftp" connection is predicted as "http" according to the model). The host model was able to detect "port scan" and some of the intrusions missed by the network model, for example, "guessing password". We constructed the meta-level connection records using the predictions from the network model and the host model. Since the network detection model uses the services as class labels, we first convert its original predictions to "normal" (when a predicted service is the same as the actual) or "abnormal" (otherwise). Some connection records may not have corresponding session records in the BSM data because these connections are destined to hosts other than the one(s) running BSM. For such records, we simply repeat the predictions of the network model in the corresponding meta-level records. The resultant meta-classifier, with "normal" and the intrusion types as

[3] Currently the IWSS16 dataset does not contain host-based audit data, although InfoWorld has agreed to gather such data for our research purposes. The DARPA dataset is still in preparation and only small sample datasets are available. We expect to receive a large amount of high quality audit data in the coming weeks and will conduct more extensive experiments.

the class labels, detects all these intrusions by using the "abnormal" predictions from the network model to detect "ping scan", and the host model predictions to detect "port scan" and "guessing password". Note that if there are more than one intrusions that can be detected by the anomaly detector but are missed by the misuse detector, then the resultant meta-classifier (that combines the anomaly detector and the misuse detector) needs to be an anomaly detector as well.

7 Related Work

Recently there have been quite a few new research efforts in intrusion detection. In [FHSL96] 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 (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).

In [LB97], 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 ($(command, options, arguments)$ 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) [FP97], 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 attributes (features) are the outputs of monitors (sensitivity measures). We face 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 statistical features (form the base-level attributes in call data). Whereas this is a critical requirement for an effective network intrusion detector.

8 Conclusion

In this paper we outlined out 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 *variable* 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 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 [KMR+94] 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 attributes (features) and the time windowing information to the user, and allows he/she to formulate a classification task, build and test the model using a classification engine such as JAM [SPT+97].

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

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216, 1993.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[Bel89] S. M. Bellovin. Security problems in the tcp/ip protocol suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[Coh95] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[CS93] P. K. Chan and S. J. Stolfo. Toward parallel and distributed learning by meta-learning. In *AAAI Workshop in Knowledge Discovery in Databases*, pages 227–240, 1993.

[FHSL96] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[FP97] Tom Fawcett and Foster Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1:291–316, 1997.

[GM84] F. T. Grampp and R. H. Morris. Unix system security. *AT&T Bell Laboratories Technical Journal*, 63(8):1649–1672, October 1984.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21th VLDB Conference*, Zurich, Switzerland, 1995.

[HLMS90] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical report, Computer Science Department, University of New Mexico, August 1990.

[IKP95] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[JLM89] V. Jacobson, C. Leres, and S. McCanne. tcpdump. available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

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

[KS95] S. Kumar and E. H. Spafford. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.

[LB97] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press, July 1997.

[LS98] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.

[LSM98] W. Lee, S. J. Stolfo, and K. W. Mok. Mining audit data to build intrusion detection models. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, New York, NY, August 1998. AAAI Press. to appear.

[LSW97] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proceedings of the Thirteenth International Conference on Data Engineering*, Birmingham, UK, 1997.

[LTG+92] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real-time intrusion detection expert system (IDES) - final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

[Lun93] Teresa F. Lunt. Detecting intruders in computer systems. In *Proceedings of the 1993 Conference on Auditing and Computer Technology*, 1993.

[MSB98] Stuart McClure, Joel Scambray, and John Broderick. Test Center Comparison: Network intrusion-detection solutions. In *INFOWORLD May 4, 1998*. INFOWORLD, 1998.

[MT96] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

[SPT+97] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, Newport Beach, CA, August 1997. AAAI Press.

[Sun] SunSoft, Mountain View, CA. *SunSHIELD Basic Security Module Guide*.

[SVA97] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 67–73, Newport Beach, California, August 1997. AAAI Press.

[UBC97] P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44, October 1997.