A Data Mining Framework for Building Intrusion Detection Models[1]

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

Abstract

There is often the need to update an installed Intrusion Detection System (IDS) due to new attack methods or upgraded computing environments. Since many current IDSs are constructed by manual encoding of expert security knowledge, changes to IDSs are expensive and slow. In this paper, we describe a data mining framework for adaptively building Intrusion Detection (ID) models. The central idea is to utilize auditing programs to extract an extensive set of features that describe each network connection or host session, and apply data mining programs to learn rules that accurately capture the behavior of intrusions and normal activities. These rules can then be used for misuse detection and anomaly detection. Detection models for new intrusions or specific components of a network system are incorporated into an existing IDS through a meta-learning (or co-operative learning) process, which produces a meta detection model that combines evidence from multiple models. We discuss the strengths of our data mining programs, namely, classification, meta-learning, association rules, and frequent episodes. We report our results of applying these programs to the (extensively gathered) network audit data from the DARPA Intrusion Detection Evaluation Program.

1 Introduction

As network-based computer systems play increasingly vital roles in modern society, they have become the target of intrusions by our enemies and criminals. In addition to intrusion prevention techniques, such as user authentication (e.g. using passwords or biometrics), avoiding programming errors, and information protection (e.g., encryption), intrusion detection is often used as another wall to protect computer systems.

Intrusion detection techniques can be categorized into *anomaly detection* and *misuse detection*. Anomaly detection systems, for example, IDES [13], flags observed activities that deviates significantly from the established normal (statistical-based) usage profiles as anomalies (i.e., possible intrusions). Misuse detection systems, for example, IDIOT [8] and STAT [5], use patterns of well-known attacks or weak spots of the system to match and identify intrusions.

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

submitted to the 1999 IEEE Symposium on Security and Privacy.

While accuracy is the essential requirement of an IDS, its extensibility and adaptability are also critical in today's network computing environment. There are multiple "penetration points" for intrusions to take place in a network system. For example, at the network level, carefully crafted "malicious" IP packets can crash a victim host; at the host level, vulnerabilities in system software can be exploited to yield an illegal root shell. Since activities at different penetration points are normally recorded in different audit data sources, an IDS often needs to be extended to incorporate (additional) modules that specialize on certain components (e.g., hosts, subnets, etc.) of the network systems. The large traffic volume in security related mailing lists and Web sites suggest that new system security holes and intrusion methods are continuously being discovered. Therefore it is imperative that IDSs be updated frequently and timely.

Currently building an effective IDS is an enormous knowledge engineering task. System builders rely on their intuition and experience to select the statistical measures for anomaly detection [14]. Experts first analyze and categorize attack scenarios and system vulnerabilities, and hand-code the corresponding rules and patterns for misuse detection. Because of the manual and ad hoc nature of the development process, current IDSs have limited extensibility and adaptability. Many IDSs only handle one particular audit data source, and their updates are expensive and slow.

Some of the recent research and commercial IDSs have started to provide built-in mechanisms for customization and extension. For example, both Bro [19] and NFR [18] filter network traffic stream into a series of events, and execute scripts (e.g., Bro policy scripts and NRR's N-Codes) that contain site-specific event handlers, i.e., intrusion detection and handling rules. The system administration personnel at each installation site must now assume the roles of both security experts and IDS builders because they are responsible for writing the correct event handling functions. Our first-hand experiences with both Bro and NFR show that while these systems provide great flexibility, writing the scripts involves a lot of effort (in addition to learning the scripting languages). For example, there is no means to "debug" the scripts. These systems also handles a fixed set of network traffic event types. In a few occasions we were forced to make changes to the source code of the original IDS to generate new event types.

Our research aims to develop a more systematic approach in building IDSs. That is, we are developing a set of automatic tools that can be applied to a variety of audit data sources to generate intrusion detection models. We take a data-centric point of view and consider intrusion detection as a data analysis process. Anomaly detection is about finding the normal usage patterns from the audit data, whereas misuse detection is about encoding and matching the intrusion patterns using the audit data. The central theme of our approach is to apply data mining programs to the extensively gathered audit data to compute models that accurately capture the *actual behavior* (patterns) of intrusions and normal activities. This automatic approach eliminates the need to manually analyze and encode intrusion patterns, as well as the guesswork in selecting statistical measures for normal usage profiles. More importantly, the same data mining tools can be applied to multiple streams of evidence, each from a detection module that specializes on a (new) type of intrusion or a specific component (e.g., a mission-critical host) of the network system, to learn the combined detection model that considers all the available evidence. Thus our framework facilitates the construction of adaptive IDSs.

The rest of the paper is organized as follows. Section 2 outlines the main components of our framework. Section 3 illustrates how to use our data mining programs to discover intrusion patterns and compute normal profiles. Section 4 reports the results of our experiments on building intrusion detection models using the audit data from the DARPA evaluation program. Section 5 discusses related research projects. Section 6 outlines our future research plans.

2 A Systematic Framework

A basic premise for intrusion detection is that when audit mechanisms are enabled to record system events, distinct evidence of legitimate activities and intrusions will be manifested in the audit data. Because of the sheer volume of audit data, both in the amount of audit records and in the number of system features (fields of the audit records), efficient and intelligent data analysis tools are required to discover the behavior of system activities.

Data mining generally refers to the process of (automatically) extracting models from large stores of data [4]. The recent rapid development in data mining has made available a wide variety of algorithms, drawn from the fields of statistics, pattern recognition, machine learning, and database. Several types of algorithms are particularly useful for mining audit data:

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

Link analysis: determines relations between fields in the database records. Correlations of system features in audit data (for example, the correlation between *command* and *argument* in the shell command history data of a user) can serve as the basis for constructing normal usage profiles. A programmer would have "emacs" highly associated with "C" files;

Sequence analysis: models sequential patterns. These algorithms can discover what (time-based) sequence of audit events are frequently occurring together. These frequent event patterns provide guidelines for incorporating temporal statistical measures into intrusion detection models. For example, patterns from audit data containing network-based denial-of-service (DOS) attacks suggest that several per-host and per-service measures should be included.

We are developing a framework, first proposed in [11], of applying data mining techniques to build intrusion detection models. This framework consists of programs for learning classifiers (and meta-classification [2]), association rules [1] (for link analysis) and frequent episodes [16] (for sequence analysis), 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 rules (that can be easily inspected and edited by security experts when needed) that can detect intrusions.

In our approach, the learned 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. Meta-learning (meta-classification) is used to learn the correlation of intrusion evidence from multiple detection models, and produces a (final) combined detection model.

It is very important to point out that our framework does not eliminate the need to pre-process and analyze raw audit data, e.g., *tcpdump* [6] and *BSM* [22] output. In fact, to build intrusion detection models for network systems, our data mining programs use (pre-processed) audit data where each record corresponds to a high level event, e.g., a network connection or host session. Each record normally includes an extensive set of features that describe the characteristics of the event, for example, the duration of a connection, the number of bytes transfered, etc.

While analyzing (and summarizing) raw audit data is an essential task for an IDS, we argue that generic utilities should first be developed by network and operating system experts, and made available to all IDSs as the lowest level building blocks. Bro and NFR can be regarded as examples of such robust and free utilities, as they both perform IP packet filtering and reassembling, and allow event handlers to output summarized connection records. Our framework assumes such building blocks when constructing IDSs.

3 Mining Audit Data

In this section we describe our data mining algorithms, and illustrate how to apply these algorithms to generate detection models from audit data. Here audit data refers to (pre-processed) timestamped audit records, each with a number of features (fields).

label	service	flag	hot	failed_logins	compromised	root_shell	su	duration	...
normal	telnet	SF	0	0	0	0	0	10.2	...
normal	telnet	SF	0	0	0	3	1	2.1	...
guess	telnet	SF	0	6	0	0	0	26.2	...
normal	telnet	SF	0	0	0	0	0	126.2	...
overflow	telnet	SF	3	0	2	1	0	92.5	...
normal	telnet	SF	0	0	0	0	0	2.1	...
guess	telnet	SF	0	5	0	0	0	13.9	...
overflow	telnet	SF	3	0	2	1	0	92.5	...
normal	telnet	SF	0	0	0	0	0	1248	...
...	...	...	...	...	...	...	...	...	...

Table 1: Telnet Records

3.1 Classification

Intrusion detection can be thought of as a classification problem: we wish to classify each audit record into one of a discrete set of possible categories, normal or a particular kind of intrusions.

Given a set of records, where one of the features is the class label (concept), classification algorithms can compute a model that uses the most distinguishing (unique) feature values to describe each concept. For example, consider the telnet connection records shown in Table 1. RIPPER [3], a classification rule learning program, generates the following rules for classifying the telnet connections:

> guess :- failed_logins $>=$ 5. [meaning: if number of failed logins is greater than 5, then this telnet connection is "guess", a guessing password attack.]
>
> overflow :- hot = 3, compromised = 2, root_shell = 1. [meaning: if the number of hot indicators is 3, the number of compromised conditions is 2, and a root shell is obtained, then this telnet connection is a buffer overflow attack. Here, hot indicators include access to system directories, creation and execution of programs, etc. Compromised conditions include file/path "not found" errors, and "Jump to" instructions, etc.]
>
> normal :- true. [meaning: if none of the above, then this telnet connection is "normal".]

Here we see that RIPPER indeed selects the unique feature values in identifying the intrusions. These rules can be first inspected and edited by security experts, and then be incorporated into misuse detection systems.

The accuracy of a classification model depends heavily on the set of features in the training data. For example, if the features "hot", "compromised" and "root_shell" were removed from the records in Table 1, RIPPER would not be able to produce accurate rules to identify buffer overflow connections. In [11] we showed that due to the temporal nature of network events, especially certain intrusions such as probing (e.g., port-scan, ping-sweep, etc.) and denial-of-service (ping-of-death, teardrop, etc.), adding per-host and per-service temporal statistics resulted in significant improvement on the accuracy of the classification models. Thus selecting the right set of system features is a critical step when formulating the classification tasks. Our strategy is to first mine the frequent episodes from the network audit data, and then use these patterns as guidelines to select and construct temporal statistical features. Section 3.3 discuss this process in more detail.

3.1.1 Meta-Classification

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 IDS should consist of multiple cooperative lightweight subsystems that each monitor a separate part (access point) of the entire network environment. For example, an IDS that inspects the full data contents of each IP packet and keeps track of all (opened) connections may run out of memory (buffers) during a DOS attack (e.g., SYN flood) and cease to function. However, a more lightweight IDS that only inspects the header of each IP packet can detect only those intrusions that are targeted to the network protocols, and not those that try to exploit the hosts (e.g., guess password, buffer overflow, etc.). A solution is to 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 mission-critical hosts. A "global" detection system can then combine the evidence from these subsystems and take appropriate actions.

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

Meta-learning [2] is a mechanism to inductively learn the correlation of predictions (classifications) by a number of pre-existing base classifiers. That is, the resulting meta-classifier combines (the classification power of) all the base classifiers.

An example of using meta-classification to construct cooperative intrusion detection models is shown in Figure 1. Here 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 classification algorithm (e.g., RIPPER) can then be applied to the meta-level records to produce the combined detection model.

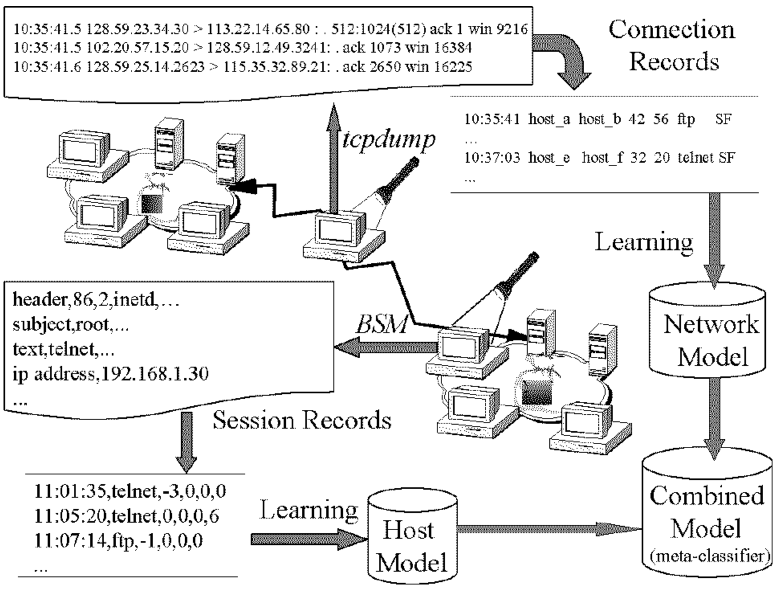

Figure 1: Combining Multiple Cooperative Intrusion Detection Models

3.2 Association Rules

There is empirical evidence that program executions and user activities exhibit frequent correlations among system features. For example, certain privileged programs only access certain system files in specific directories [7]. Programmers edit and compile C files frequently. These consistent behaviors should be included in normal usage profiles.

The goal of mining association rules is to derive multi-feature (attribute) correlations from a database table. Given a set of records, where each record is a set of items, $support(X)$ is defined as the percentage of records that contain item set X. An association rule is an expression $X \rightarrow Y, [c, s]$ [1]. Here $X \cap Y = \emptyset$, $s = support(X \cup Y)$ is the support of the rule, and $c = \frac{support(X \cup Y)}{support(X)}$ is the confidence.

Table 2 shows the shell input commands during one telnet session by a secretary. Here we keep only the filename extensions, remove the (input) contents of mail bodies and files, and use "am" to represent all the morning timestamps. Not all associations, for example, $hostname = pascal \rightarrow arg1 = tex$, are useful. We extended the original association rules algorithm to make use of the "order of importance" relations among the system features. We use the term *axis* feature(s) to denote the most important features(s) [12]. In the case of shell command records, "command" is the axis feature. Our program only outputs rules that include axis feature values, for example,

$$command = vi \rightarrow time = am, hostname = pascal, arg1 = tex, [1.0, 0.28]$$

$$command = subject \rightarrow time = am, hostname = pascal, arg1 = progress, [1.0, 0.11]$$

	time	hostname	command	arg1	arg2
1	am	pascal	mkdir	dir1	
2	am	pascal	cd	dir1	
3	am	pascal	vi	tex	
4	am	pascal	tex	vi	
5	am	pascal	mail	fredd	
6	am	pascal	subject	progress	
7	am	pascal	vi	tex	
8	am	pascal	vi	tex	
9	am	pascal	mail	williamf	
10	am	pascal	subject	progress	
...	...	...	...	...	
18	am	pascal	vi	tex	
19	am	pascal	latex	tex	
20	am	pascal	dvips	dvi	-o
...	...	...	...	...	
209	am	pascal	logout		

Table 2: Shell Command Records

Each of these association rules conveys information about the user's behavior. The rules mined from each telnet/login session (of the same user) can be merged into an aggregate rule set to form the user's normal profile. Section 4.3 details our experiments of using association rules for anomaly detection.

3.3 Frequent Episodes

There is often the need to study the frequent sequential patterns of network events in order to understand the nature of many attacks. We use frequent episodes to represent the (sequential) inter- audit record patterns.

Given a set of timestamped event records, where each record is a set of items, an interval $[t_1, t_2]$ is the sequence of event records that starts from timestamp t_1 and ends at t_2. The width of the interval is defined as $t_2 - t_1$. Let X be a set of items, an interval is a minimal occurrence of X if it contains X and none of its proper sub-intervals contains X. Define $support(X)$ as the ratio between the number of minimum occurrences that contain X and the total number of event records. A frequent episode rule is the expression $X, Y \rightarrow Z, [c, s, w]$ [15]. Here X, Y and Z are item sets, and they together form an episode. $s = support(X \cup Y \cup Z)$ is the support of the rule, and $c = \frac{support(X \cup Y \cup Z)}{support(X \cup Y)}$ is the confidence. The width of each of the occurrences must be less than w.

We introduced several extentions to the original frequent episodes algorithm. Our extended algorithm computes frequent sequential patterns in two phases. First, it finds the frequent associations using the axis attributes(s) as previously described. Then it generates the frequent serial patterns from these associations.

timestamp	duration	service	src_host	dst_host	src_bytes	dst_bytes	flag	...
1.1	0	http	spoofed_1	victim	0	0	S0	...
1.1	0	http	spoofed_2	victim	0	0	S0	...
1.1	0	http	spoofed_3	victim	0	0	S0	...
1.1	0	http	spoofed_4	victim	0	0	S0	...
1.1	0	http	spoofed_5	victim	0	0	S0	...
1.1	0	http	spoofed_6	victim	0	0	S0	...
1.1	0	http	spoofed_7	victim	0	0	S0	...
...	...	...	...	...	...	...	...	...
10.1	2	ftp	A	B	200	300	SF	...
12.3	1	smtp	B	D	250	300	SF	...
13.4	60	telnet	A	D	200	12100	SF	...
13.7	1	smtp	B	C	200	300	SF	...
15.2	1	http	D	A	200	0	REJ	...
...	...	...	...	...	...	...	...	...

Table 3: Network Connection Records

Thus, our approach combines the associations (among attributes) and the sequential patterns (among the records) into a single rule. Our algorithm can also make use of the "reference" relations among the system features. For example, if we want to study the sequential patterns of connections to the same destination host, we can designate "dest_host" as the *reference* feature. When forming an episode, an additional condition is that within its minimal occurrences the event records covered by its constituent item sets have the same reference feature value.

We use the mined frequent episodes from network connection records as guidelines to construct temporal statistical features for building classification models. Raw *tcpdump* output are first summarized into network connection records, where each record has a set of features that are "intrinsic" to a connection, for example, the duration, service, number of bytes transfered, etc. Table 3 shows examples of connection records. We apply the frequent episodes program to both normal connection data and intrusion data, and study the resulting patterns (we outlined algorithms for visualizing and comparing frequent sequential patterns in [10]). As an example, consider the SYN flood attack shown in Table 3. Here the attacker, using many spoofed source addresses, sends a lot of S0 (only the first SYN packet is sent) connections to a port (*http*) of the victim host in a very short time span. When using "service" as the axis feature and "dest_host" as the reference feature, our algorithm produces this unique pattern (not seen in normal data) that describes the attack,

$$(service = http, flag = S0, dst\_host = victim), (service = http, flag = S0, dst\_host = victim)$$
$$\rightarrow (service = http, flag = S0, dst\_host = victim)[0.92, 0.13, 2]$$

We performed such pattern mining comparisons using intrusion data of several well-known attacks, e.g., port-scan, ping-sweep, etc. We used a number of variations of axis features and reference features (but limiting the choices to be among "service", "src_host", and "dst_host") for each attack in order to understand what the behavior of the intrusion from different perspectives. For example, when "service" was used as both the axis and reference feature, ping-sweep data generated very distinct patterns, and for port-scan, "dst_host" was both the axis and reference feature.

Each of these intrusion patterns can serve as guidelines for adding additional features into the connection records to build better classification models:

- Assume F_0 (e.g., dst_host) is used as the reference feature, and the width of the episode is w seconds.
- For each connection record, examine all connections in the past w seconds that share the same value in F_0:
  - Let F_1 be "service", "src_dst" or "dst_host" other than F_0, if the same F_1 value (e.g., "http") is in all the item sets of the episode, add a count of connections that share the same F_1 value (as the current connection); otherwise, add a count of different (unique) values of F_1.
  - Let V_2 be a value (e.g., "S0") of a feature (e.g., "flag") other than F_0 and F_1, if V_2 is in all the item sets of the episode, add a count of connections that have the same V_2.

An open problem here is how to decide the right time window value w. We mine sequential patterns using different w values, from 0.1 to 20 with an increment of 1, and plot the number of patterns generated at each run. Our experience show that this plot tends to stabilize after the initial sharp jump. We call the smallest w in the stable region w_0. In [11], we reported experiments of using different w values to calculate temporal statistical features for classification models. Our results showed the plot of accuracy of the classifier also stabilizes after $w \geq w_0$ and tend to taper off. Intuitively, a requirement for a good window size is that its set of sequential patterns is stable, that is, sufficient patterns are captured and noise is small. We therefore use w_0 for adding temporal statistical features.

4 Experiments in Building Intrusion Detection Models

We are currently participating in the DARPA Intrusion Detection Evaluation Program (prepared and managed by MIT Lincoln Labs). The objective of this program is to survey and evaluate research in intrusion detection. A standard set of extensively gathered audit data, which includes a wide variety of intrusions simulated in a real-world network environment (a military installation), is provided by DARPA. Each participating site is to build intrusion detection models (or tweak their existing system parameters) using the training data, and send the results (detections) on the test data back to DARPA for performance evaluation. We report our preliminary results here[2].

4.1 The DARPA data

We were provided with *tcpdump* data of 7 weeks of network traffic. The data contains contents (the data portion) of every packet transmitted between hosts inside and outside a simulated military base. BSM audit data from one UNIX Solaris host for some network sessions were also provided.

Four main categories of attacks were simulated, including denial-of-service (e.g., ping-of-death, teardrop, smurf, SYN flood, etc.), unauthorized access from a remote machine (e.g., guessing password), unauthorized access to local superuser privileges by a local unprivileged user (e.g., various of buffer overflow attacks), and surveillance and probing (e.g., port-scan, ping-sweep, etc.). In addition, there were anomalous user behavior such as "a manager becomes (behaves like) a system administrator".

4.1.1 Data Pre-processing

We use Bro as the packet filtering and reassembling engine. We extended Bro to handle ICMP packets, and made changes to its packet fragment inspection modules (it crashed when processing data that contains teardrop or ping-of-death attacks).

In the Bro "policy scripts" (event handlers), we added functions that inspect data exchanges of interactive TCP connections (e.g., telnet, ftp, smtp, etc.). These functions assign values to a set of "content" features when the data contents suggest suspicious behavior. These total 15 content features include: number of "hot actions" (e.g., access to system directories, creation and execution of programs, etc.), number of "compromised conditions" for the destination host (e.g., file/path "not found" errors, and "Jump to" instructions, etc.), number of access to "access control" files (e.g., /etc/passwd, .rhosts), number of failed login attempts, whether a root shell is obtained, whether a "su" command succeeded, etc. Our approach here is to include as extensive a set of indicators as possible, and then let our data mining programs decide which minimal set of discriminating features should be used to identify intrusions.

[2] The detection results on the final test data are due back to DARPA in November, and the evaluation results will be made available by DARPA in December. We will then report the performance of our detection models accordingly

The “connection finished” event handler outputs a summarized record for each connection. Each connection record includes “traffic” features that describe its intrinsic characteristics: its start time, duration, service, source and destination hosts, bytes in both directions, a flag indicating how the connection is established and terminated, the number of “wrong fragments” (e.g., fragment size is too small, to large, or the offset is wrong, etc.). TCP connections have the additional “content” features.

For each connection record, we also added the following temporal statistical features according to the procedures outline in Section 3.3:

- For all connections in the past 2 seconds that have the same destination host as the concurrent connection: the number of such connections, the number of connections that have the same service as the current one, the number of unique services, the number of the S0 flag, and the number of the REJ (connection rejected) flag;

- Similarly, for all connections in the past 2 seconds that have the same service as the current connection: the number of unique destination hosts, the number of the S0 flag, and the number of the REJ (connection rejected) flag.

4.2 Misuse Detection

The training data from DARPA includes “list files” that identify the timestamp, source host and port, destination host and port, and the name of each attack. We use this information to label each pre-processed connection record as either “normal” or one of the attacks.

We use the data from one particular day for our own testing, and train our detection models using the data from all the other days. For each experiment, the training data set consists of intrusion records as well normal connection records. We apply RIPPER to learn classification rules for each type of intrusions. The classifier is then used to classify each connection in the test data as either intrusion or normal. We report both the true positive (TP) and false positive (FP) rates of the resultant classifiers. TP refers to the situation where the classification predicts an intrusion correctly. FP refers to the situation where a normal record is (falsely) identified as an intrusion.

Since the amount of audit data is huge, for example, some days have several millions of connection records (due to the nasty DOS attacks), we do not aggregate all the connection records (of all days) into a single training data set. Instead, we extract all the connection records that fall within a surrounding time window (of plus and minus 5 minutes) of the whole duration of each intrusion. We also randomly extract sequences of normal connections records.

	OVERALL		DOS		PROBING		R2L		U2R	
Experiment	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
FAT	93%	8%	95.2%	6%	90.2%	11.2%	96.5%	2.2%	99.2%	0.1%
Lean	95.3%	6.1%	96.2%	5.2%	93.2%	7.5%	96.2%	2.1%	99.2%	0.1%

Table 4: TP and FP of Misuse Detection Models

4.2.1 A Fat Classifier

Our first experiment is to build a classifier that can by itself detect all the intrusions. Each connection record has a very large set of features, including the (intrinsic) traffic features, temporal statistical features, and the content features. The "FAT" row in Table 4 shows that TP and FP rates of this classifier. Here we report the overall performance results, and also by the 4 main categories of intrusions, namely denial-of-service (DOS), probing, remote user trying to gain access to local host (R2L), and local users trying to gain access to root (U2R).

The results show that the classifier is not very effective in detecting probing attacks and DOS attacks. We observe that there are variations of the attacks that make our choices of temporal statistical features inadequate. For example, some probing attacks try to hide their traces by slowing down their pace (frequencies) significantly than the more well-known behavior. For example, a few port-scan and ping-scan attacks take upto a few minutes moving from one port or host to the next. However our statistics are computed based on a time span of 2 seconds. A more serious problem is that since there are a large number of features, the rule learning algorithm needs to use a lot of heuristics to search the huge hypothesis space, and in some cases it would create rules that are not very accurate. For example, the generated rule for identifying smurf attack (DOS using icmp echo reply flood) is simply a condition "if service = icmp echo request. The rule learner in this case simply fails to also consider the more useful temporal statistical features, the evidence of "flood".

4.2.2 Lean Classifiers

Our second experiment is to build "lean" detection models that are more specialized in certain categories of attacks. The main objective is to see whether a meta-classifier that combines a set of separately learned "lean" models can improve upon the performance results of the "FAT" classifier.

From the "FAT" models, we discover that all rules for R2L and U2R attacks use only the content features, and none of the rules for DOS and probing attacks use any of the content features. We build two separate classification models. For DOS and probing, we remove content features from the connection records. For R2L and U2R, we remove the temporal statistical features, we also only keep TCP records since content features have no meaning for other types of connections.

The meta-level training data consists of three features, the first two are the predictions on a subset of the test data by the two (base) classifiers, and the third is the true class label. RIPPER is then applied to learn the rules that combine the evidence for the two classifiers. The resulting meta-level rules basically select the more accurate base classifiers for each intrusion type.

The "Lean" row in Table 4 shows some performance improvement in detecting DOS and probing attacks. This improvement comes from the benefit of reduced feature set size, and the selection power of the meta-classifier. For example, the "smurf" rule in the traffic base classifier (that handles DOS and probing) is now "if protocol is icmp echo request; and for the connections in the past 2 seconds, there are more than 5 connections to the same destination host (as the current one), and there are more than 5 icmp echo request connections.". The meta-rule for smurf attack simply agrees with this base classifier, i.e., "if this base classifier says it is smurf". The "port-scan" rule in the traffic base classifier has also become more accurate. One of its rules is "if service is unrecognized (connection to unregistered port), and flag is REJ (a SYN packet is responded with a RST packet)". This rule can detect many of the slow-paced port-scan attacks.

4.3 Anomaly Detection

"Insiders" misusing their privileges can be hard to detect since they don't normally need to break-in and most of the IDSs and security personnel are normally focussed on guarding outside attacks. Insider problems are some of the most vexing problems for security personnel.

It is often very difficult to classify a single event by a user as normal or abnormal because the unpredictable nature of most people. A user's actions during a login session needs to be studied as a whole to determine whether he or she is behaving normally. Our approach is to mine the frequent patterns from user command data, and merge or add the patterns into an aggregate set to form the normal usage profile of the user. A new pattern (rule) can be merged with an old pattern if they have the same left-hand-sides and right-hand-sides, their support values are within a 5% of each other, and their confidence values are also within 5% of each other.

To analyze a user login session, we mine the frequent patterns from the sequence of commands during this session. This new pattern set is compared with the profile pattern set and a *similarity* score is assigned. Assume that the new set has n patterns and among them, there are m patterns that have "matches" (i.e., rules that they can be merged with) in the profile pattern set, then the similarity score is simply $\frac{m}{n}$. Obviously, a higher similarity score means a higher likelihood that the user's behavior agrees with his or her historical profile.

The DARPA data also includes user anomaly data to evaluate anomaly detection systems. Table 5

sysadm	A system administrator who logs in as root, cats the password file and runs commands such as top.
programmer1	A programmer who writes public domain C code using a vi editor, compiles the C code (sometimes successfully), reads and sends mail, and executes unix commands.
programmer2	Another programmer with a similar user profile, except that he works afternoons and evenings.
secretary	A secretary who edits latex files, runs latex, reads mail, and sends mail.
manager1	A manager who reads and sends mail
manager2	A manager who reads mail.

Table 5: User Descriptions

Week-day	User	Starttime	Anomaly Description
6th Week Mon.	programmer2	14:57:10	logs in from beta
6th Week Mon.	secretary	20:03:39	logs in at night
6th Week Tue.	sysadm	08:29:48	logs in from jupiter
6th Week Tue.	programmer1	08:52:15	becomes a secretary
6th Week Wed.	secretary	08:16:01	becomes a manager
6th Week Wed.	programmer1	21:09:24	logs in at night
6th Week Thu.	sysadm	08:54:53	becomes a programmer
6th Week Fri.	manager1	08:11:12	becomes a sysadm
6th Week Fri.	manager2	08:42:52	logs in from pluto

Table 6: Anomaly Schedule

describes the consistent behavior of the 6 users for anomaly analysis. From the tcpdump data, we extract the shell commands by these users during their telnet sessions. We pre-process the shell commands by replacing timestamps with am, pm, and nt (for night). We eliminate the input (contents) of edit and sendmail commands, and keep only the filename extensions. Table 2 shows examples of the processed command data.

We apply our frequent episode algorithm to the command data from each login session (of the same user), with "command" as the axis feature and $w = 5$ (i.e., we look for patterns within the range of 5 consecutive commands), to mine the frequent sequential patterns on the associations among user commands, their arguments, time segments, and hosts. We treat the first 4 weeks as a data gathering period, during which we simply merge the patterns into each user's profiles. Each user has 3 profiles, one for the activities of each time segment (am, pm, and nt). We treat the 5th week as the training period, during which we compare the patterns from each session to the profile of the time segment. We record the (normal) range of the similarity scores during this week. The data in the 6th week has some user anomalies, as described in Table 6. For each of the anomalous session, we compare its patterns against the (original) user's profile, and then compare the resulting similarity score against the normal range (of the same time segment). In Table 7, the column labeled “Normal” is the range of similarity of each user against his or her own profile (as recorded during the 5th week). A ∞ here means that the user did not login during the time segment in the 5th week. The column “Anomaly” is the similarity measure of the anomalous session. We see that all anomalous sessions can be clearly detected since their similarity scores are much smaller than the normal range. For example, when the sysadm becomes programmer on Thursday pm, his/her patterns have zero matches with the sysadm’s profile, while for the whole 5th week, the pm similarity scores are in the range of 0.64 to 0.95.

User	**Normal**	**Anomaly**
programmer2	(0.58, 0.79)	0.00
secretary	(∞, ∞)	0.00
sysadm	(0.84, 0.95)	0.00
programmer1	(0.31, 1.00)	0.04
secretary	(0.41, 0.98)	0.17
programmer1	(∞, ∞)	0.00
sysadm	(0.64, 0.95)	0.00
manager1	(0.57, 1.00)	0.00
manager2	(1.00, 1.00)	0.00

Table 7: Similarity with the User’s Own Profile

In order to understand the nature of the user anomaly, we can compare his or her activities with other groups to see if there is a match. We define the S group as the secretary, the M group as the two managers, the P group the two programmers and the SA group as the Sysadm. For the first 4 weeks, we do profile building on a group basis. That is, all the patterns from the users belonging to the same group are merged into the same profile. Then for the 5th week, we compare the target user against the profiles of all groups, and record the range of normal similarity scores. In Table 8, the column “Normal” indicates that each user indeed show the strongest association to his or her own group. The column “Anomaly” shows, when the user assumes another’s identity, e.g., when manager1 becomes sysadm, he or she indeed becomes more “similar” to the group of the stolen identity.

User	Normal				Anomaly			
	P	**S**	**M**	**SA**	**P**	**S**	**M**	**SA**
programmer2	(0.33, 0.71)	(0.04, 0.11)	(0.00, 0.03)	(0.00, 0.07)	0.00	0.00	0.00	0.00
secretary	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	0.04	0.00	0.00	0.00
sysadm	(0.00, 0.00)	(0.00, 0.00)	(0.00, 0.00)	(0.51, 0.81)	0.00	0.00	0.00	0.00
programmer1	(0.12, 0.57)	(0.06, 0.09)	(0.00, 0.00)	(0.04, 0.14)	0.04	0.11	0.00	0.00
secretary	(0.02, 0.18)	(0.08, 0.73)	(0.00, 0.00)	(0.00, 0.00)	0.17	0.00	0.50	0.00
programmer1	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	0.27	0.00	0.00	0.00
sysadm	(0.00, 0.00)	(0.00, 0.00)	(0.00, 0.00)	(0.51, 0.81)	0.24	0.03	0.00	0.00
manager1	(0.14, 0.17)	(0.00, 0.00)	(0.29, 1.00)	(0.00, 0.00)	0.02	0.00	0.00	0.61
manager2	(0.50, 1.00)	(0.00, 0.00)	(1.00, 1.00)	(0.00, 0.00)	0.00	0.00	0.00	0.00

Table 8: Similarity with Four Groups' Profiles

5 Related Work

Network intrusion detection has been an on-going research area [17]. More recent systems, e.g. Bro [19], NFR [18], and EMERALD [20] all made extensibility their primary design goals. Our research focuses on automatic methods for constructing intrusion detection models. The meta-learning mechanism is designed to automate the extention process of IDSs. We share the same views discussed in [21] that an IDS should be built using standard components. We believe that the operating system and networking community should be responsible for building a robust "Event" box.

In [9], algorithms for analyzing user shell commands and detecting anomalies were discussed. The basic idea is to first collapse the multi-column shell commands into a single stream of strings, and then string matching techniques and consideration of "concept drift" are used to build and update user profiles. We believe that our extended frequent episodes algorithm is a superior approach because it considers both the association among commands and arguments, and the frequent sequential patterns of such associations.

6 Conclusions and Future Directions

In this paper, we outline a data mining framework for constructing intrusion detection models. The key idea is to apply data mining programs to audit data to compute misuse and anomaly detection models, according to the observed behavior in the data. To facilitate adaptability and extensibility, we propose the use of meta-learning as a means to construct a combined model that incorporate evidence from multiple models. This mechanism makes it feasible to introduce new ID components in an existing IDS without significant re-engineering.

We extend the basic association rules and frequent episodes algorithms to accommodate the special requirements in analyzing audit data. Our experiments show that the frequent patterns mined from audit data can be used as reliable anomaly detection models, and as guidelines for selecting temporal statistical features to build effective classification models. Although our results on data from the DARPA Intrusion Detection Evaluation Programs are still preliminary, highly accurate detection models have been built automatically using our systems. We will give a final performance analysis when the current evaluation program concludes.

Our future work include devising a mechanical procedure to translate our (automatically) learned detection rules into modules, e.g., N-codes, of real-time IDSs, e.g., NFR. Some preliminary projects have just started.

7 Acknowledgments

We wish to thank the colleagues in our extended research group for their help and encouragement.

References

[1] 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.

[2] 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.

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

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

[5] 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.

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

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

[8] 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.

[9] 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.

[10] W. Lee, K. W. Mok, and S. J. Stolfo. Mining sequential patterns: Techniques, visualization, and applications. submitted for publication, August 1998.

[11] 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.

[12] 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.

[13] 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.

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

[15] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[16] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

[17] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *IEEE Network*, May/June 1994.

[18] Inc. Network Flight Recorder. Network flight recorder. http://www.nfr.com, 1997.

[19] Vern Paxon. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[20] Phillip A. Porras and Peter G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference*, Baltimore MD, October 1997.

[21] S. Stainford-Chen. Common intrusion detection framework. http://seclab.cs.ucdavis.edu/cidf.

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