A Framework for Constructing Features and Models for Intrusion Detection Systems

Wenke Lee
North Carolina State University
and
Salvatore J. Stolfo
Columbia University

Intrusion detection (ID) is an important component of infrastructure protection mechanisms. Intrusion detection systems (IDSs) need to be accurate, adaptive, and extensible. Given these requirements and the complexities of today's network environments, we need a more systematic and automated IDS development process rather than the pure knowledge encoding and engineering approaches. This paper describes a novel framework, MADAM ID, for Mining Audit Data for Automated Models for Intrusion Detection. This framework uses data mining algorithms to compute activity patterns from system audit data and extracts predictive features from the patterns. It then applies machine learning algorithms to the audit records that are processed according to the feature definitions to generate intrusion detection rules. Results from the 1998 DARPA Intrusion Detection Evaluation showed that our ID model was one of the best performing of all the participating systems. We also briefly discuss our experience in converting the detection models produced by off-line data mining programs to real-time modules of existing IDSs.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Monitoring*; D.4.6 [**Operating Systems**]: Security and Protection; H.2.8 [**Database Management**]: Database Applications—*Data Mining*; I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning*

General Terms: Design, Experimentation, Security

Additional Key Words and Phrases: Intrusion Detection, Data Mining, Feature Construction

---

This paper is based on the authors' published papers in the Proceedings of the 1999 IEEE Symposium on Security and Privacy and the Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, as well as Wenke Lee's Ph.D. thesis in the Computer Science Department at Columbia University.
Name: Wenke Lee
Address: 1010 Main Campus Drive, Box 7534; Computer Science Department, North Carolina State University; Raleigh, NC 27695-7534, USA. Email: wenke@csc.ncsu.edu.
Name: Salvatore J. Stolfo
Address: 1214 Amsterdam Avenue; Computer Science Department, Columbia University, Mailcode 0401; New York, NY 10027-7003, USA. Email sal@cs.columbia.edu.

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 and authorization, encryption, and defensive programming, intrusion detection is often used as another wall to protect computer systems.

The two main intrusion detection techniques are *misuse detection* and *anomaly detection*. Misuse detection systems, for example, IDIOT [Kumar and Spafford 1995] and STAT [Ilgun et al. 1995], use patterns of well-known attacks or weak spots of the system to match and identify known intrusions. For example, a signature rule for the "guessing password attack" can be "there are more than 4 failed login attempts within 2 minutes". Misuse detection techniques in general are not effective against novel attacks that have no matched rules or patterns yet. Anomaly detection (sub)systems, for example, the anomaly detector of IDES [Lunt et al. 1992], flag observed activities that deviate significantly from the established normal usage profiles as anomalies, i.e., possible intrusions. For example, the normal profile of a user may contain the averaged frequencies of some system commands used in his or her login sessions. If for a session that is being monitored, the frequencies are significantly lower or higher, then an anomaly alarm will be raised. Anomaly detection techniques can be effective against unknown or novel attacks since no *a priori* knowledge about specific intrusions is required. However, anomaly detection systems tend to generate more false alarms than misuse detection systems because an anomaly can just be a new normal behavior. Some IDSs, e.g., IDES and NIDES [Anderson et al. 1995], use both anomaly and misuse detection techniques.

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 IDSs need to be adaptive in such a way that frequent and timely updates are possible.

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 [Lunt 1993]. 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 [Allen et al. 2000].

Some of the recent research and commercial IDSs have started to provide built-in mechanisms for customization and extension. For example, both Bro [Paxson 1998] and NFR [Network Flight Recorder Inc. 1997] filter network traffic streams into a series of events, and execute scripts, e.g., Bro policy scripts and NFR'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 experience 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 handle a fixed set of network traffic event types. On a few occasions we were forced to make changes to the source code of the original IDS to handle new event types.

Our research aims to develop a more systematic and automated approach for building IDSs. We have developed a set of 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* (i.e., patterns) of intrusions and normal activities. This approach significantly reduces the need to manually analyze and encode intrusion patterns, as well as the guesswork in selecting statistical measures for normal usage profiles. The resultant models can be more effective because they are computed and validated using large amount of audit data. Further, data mining programs can be applied to multiple streams of evidence, each from a detection module that specializes in a specific type(s) of intrusion or a specific component of the network system, e.g., a mission-critical host, to learn the combined detection model that considers all the available evidence. Therefore, using our framework, IDSs can be extended and adapted easily via automated integration of new modules.

The rest of the paper is organized as follows. Section 2 outlines the main components of our framework. Section 3 briefly describes several data mining programs, and discusses how they can be applied to discover frequent intrusion and normal activity patterns, which are the basis for building misuse detection models and user anomaly detection models. Section 4 describes how to construct temporal and statistical features using the frequent patterns mined from audit data. Section 5 reports the results of our experiments on building intrusion detection models using the audit data from the DARPA evaluation program. Section 6 briefly describes our approach of converting off-line learned models into real-time intrusion detection modules. Section 7 discusses related research projects. Section 8 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 (i.e., the fields describing 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 extracting useful models from large stores of data [Fayyad et al. 1996]. 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 databases. 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, for example, may 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 have developed a framework, MADAM ID (for Mining Audit Data for Automated Models for Intrusion Detection), described in [Lee and Stolfo 1998; Lee et al. 1999a; Lee et al. 1999b; Lee 1999]. The main idea is to apply data mining techniques to build intrusion detection models. The main components of the framework include programs for learning classifiers and meta-classifiers [Chan and Stolfo 1993], association rules [Agrawal et al. 1993] for link analysis, and frequent episodes [Mannila et al. 1995] for sequence analysis. It also contains 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 detect intrusions, and can be easily inspected and edited by security experts when needed.

The process of applying MADAM ID can be summarized in Figure 1. Raw (binary) audit data is first processed into ASCII network packet information (or host event data), which is in turn summarized into connection records (or host session records) containing a number of basic features, e.g., *service*, *duration*, etc. Data mining programs are then applied to the connection records to compute the frequent patterns, i.e., association rules and frequent episodes, which are in turn analyzed to construct additional features for the connection records. Classification programs, for example, RIPPER [Cohen 1995], are then used to inductively learn the detection models. This process is of course iterative. For example, poor performance of the classification models often indicates that more pattern mining and feature construction is needed.

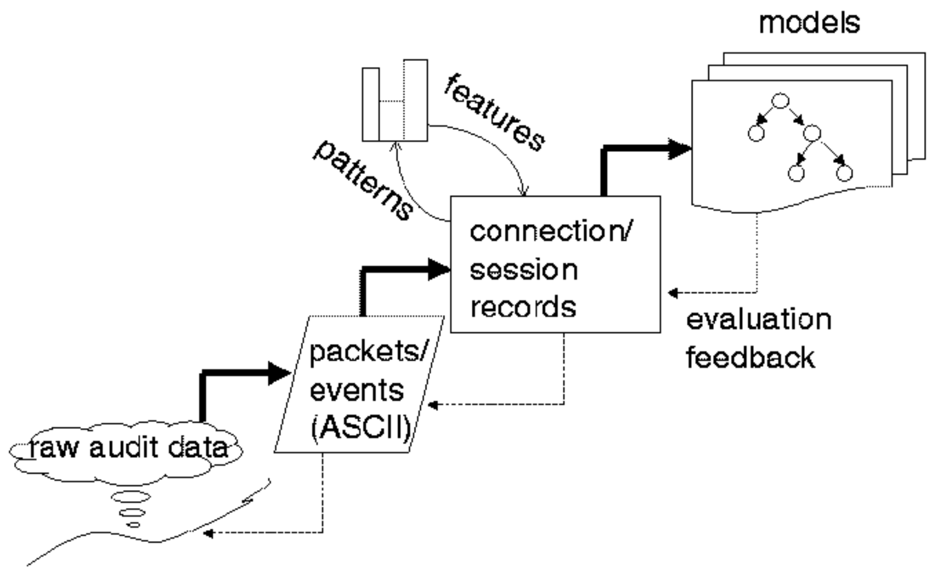


Fig. 1. The data mining process of building id models.

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 is used to learn the correlation of intrusion evidence from multiple detection models, and to produce a combined detection model.

Our framework does not eliminate the need to pre-process and analyze raw audit data, e.g., *tcpdump* [Jacobson et al. 1989] or *BSM* [SunSoft 1995] audit data. 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 low level building blocks. Bro and NFR can be regarded as examples of such robust utilities, as they both perform IP packet filtering and reassembling, and allow event handlers to output summarized connection records. Our framework assumes that such building blocks are available when constructing IDSs.

Note that currently MADAM ID produces misuse detection models for network and host systems as well as anomaly detection models for users. We are extending MADAM ID to build network and host anomaly detection models. Also note that the detection models produced by MADAM ID are intended for off-line analysis. In Section 6, we will briefly discuss how to convert these models to on-line detection modules.

Table 1. Telnet Records

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 2. Example RIPPER Rules from Telnet Records Shown in Table 1

RIPPER Rule	Meaning
guess :- failed_logins $\geq$ 4.	If number of failed logins is at least 4, then this telnet connection is “guess”, a guessing password attack.
overflow :- hot $\geq$ 3, compromised $\geq$ 2, root_shell = 1.	If the number of hot indicators is at least 3, the number of compromised conditions is at least 2, and a root shell is obtained, then this telnet connection is a buffer overflow attack.
...	...
normal :- true.	If none of the above, then this connection is “normal”.

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 (i.e., fields).

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 intrusion.

Given a set of records, where one of the features is the class label, i.e., the concept to be learned, classification algorithms can compute a model that uses the most discriminating feature values to describe each concept. For example, consider the telnet connection records shown in Table 1. *label* is the concept to be learned. “normal” represents normal connections and “guess” and “overflow” represent various kinds of intrusions. *hot* is the count of access to system directories, creation and execution of programs, etc., *compromised* is the count of file/path “not found” errors and “Jump to” instructions, etc. RIPPER [Cohen 1995], a classification rule learning program, generates rules, shown in Table 2, for classifying the telnet connections. The symbol left to “:-” is the class label, and the comma separated expressions on the right are conjuncts, i.e., (sub)conditions, of the classification rule. We see that RIPPER indeed selects the discriminating feature values into the classification rules for 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 directly on the set of features provided in the training data. From a theoretical point of view, the goal of constructing a classification model is that after (selectively) applying a sequence of feature value tests, the dataset can be partitioned into "pure" subsets, i.e., each in a target class. Therefore when constructing a classification model, a classification algorithm searches for features with large *information gain* [Mitchell 1997], defined as the reduction in *entropy*, which characterizes the "impurity" of a dataset. It is thus very important that the dataset indeed includes features with large information gain. 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 [Lee and Stolfo 1998] 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 (e.g., ping-of-death, teardrop, etc.), adding per-host and per-service temporal statistics resulted in significant improvement in 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 sequential patterns from the network audit data, and then use these patterns as guidelines to select and construct temporal statistical features. Section 3.3 discusses this process in greater detail.

3.1.1 *Meta-classification.* Meta-learning [Chan and Stolfo 1993] is a mechanism for inductively learning the correlation of predictions by a number of (base) classifiers. Each record in the training data for meta-learning contains the true class label of the record and the predictions made by the base classifiers. The resultant meta-classifier thus "combines" the base models because it uses their predictions to make the final prediction. The motivations for meta-learning include: to improve classification accuracy, that is, to produce a meta-classifier that is more accurate than any individual base classifier; and to improve efficiency and scalability, that is, to combine the models rather than the potentially huge volume of data from different data sources. This general approach has been extensively studied [Stolfo et al. 1997] and empirically evaluated in a related domain of credit card fraud detection and has been shown to be effective and scalable.

In order to avoid becoming a performance bottleneck and an easy attack target, an IDS should consist of multiple cooperative lightweight subsystems that each monitors a separate part, e.g., 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 (i.e., buffers) during a TCP-based DoS attack and cease to function. On the other hand, 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 use meta-learning as a means to combine multiple

Table 3. Shell Command Records

time	hostname	command	arg1	arg2
am	pascal	mkdir	dir1	
am	pascal	cd	dir1	
am	pascal	vi	tex	
am	pascal	tex	vi	
am	pascal	mail	fredd	
am	pascal	subject	progress	
am	pascal	vi	tex	
am	pascal	vi	tex	
am	pascal	mail	williamf	
am	pascal	subject	progress	
	...	...	...	...
am	pascal	vi	tex	
am	pascal	latex	tex	
am	pascal	dvips	dvi	-o
...	...	...	...	...
am	pascal	logout		

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 [Ko et al. 1994], programmers edit and compile C files frequently, etc. These consistent behavior patterns 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

$$X \rightarrow Y, [c, s]$$

Here X and Y are item sets, 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 [Agrawal et al. 1993].

Consider the shell input commands during one telnet session by a secretary, shown in Table 3. 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.

The original association rules algorithm searches for all possible frequent associations among the set of given features. However, not all associations are necessarily useful for analyzing program or user behavior. We utilized the "schema" level information, i.e., data definitions, about the audit records to direct the pattern mining process. Observe that certain features are essential in describing the data, while others provide only *auxiliary* information. Domain knowledge is used to determine what are the appropriate essential features for an application. In shell command data, since the combination of the exact "time" and "command" uniquely identifies each record, "time" and "command" are the *essential features*; likewise, in network connection data, timestamp, source and destination hosts, source port, and service (i.e., destination port) are the essential features because their combination uniquely

Table 4. Example Association Rules from Shell Command Data Shown in Table 3

Association Rule	Meaning
$command = vi \rightarrow time = am$, $hostname = pascal$, $arg1 = tex$, $[1.0, 0.28]$	When using vi to edit a file, the user is always (i.e. 100% of the time) editing a *tex* file, in the morning, and at host *pascal*; and 28% of the command data matches this pattern.
$command = subject \rightarrow time = am$, $hostname = pascal$, $arg1 = progress$, $[1.0, 0.11]$	The subject of the user's email is always ((i.e. 100% of the time) about "progress", such emails are in the morning, and at host *pascal*; and 11% of the command data matches this pattern.

identifies a connection record. We argue that the relevant association rules should describe patterns related to the essential features.

We call these essential features(s) *axis* features when they are used as a form of item constraint, which specifies the conditions on the item sets of an association rule. We restrict the association rules algorithm to only output rules that include axis feature values. In practice, we need not designate all essential features as the axis features. For example, some network analysis tasks require statistics about various network services while others may require the patterns related to the destination hosts. Accordingly, we can use *service* as the axis feature to compute the association rules that describe the patterns related to the services of the connections, and use *destination host* as the axis feature to compute patterns related to hosts.

In the case of shell command records, we use *command* as the axis feature. Table 4 shows some example association rules from the shell command data in Table 3. 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 5.1.8 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 audit data in order to understand the temporal and statistical nature of many attacks as well as the normal behavior of users and programs. We use frequent episodes to represent the sequential 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 [Mannila and Toivonen 1996]

$$X, Y \rightarrow Z, [c, s, w]$$

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 extensions to the original frequent episodes algorithm. Our

Table 5. Network Connection Records

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

extended algorithm computes frequent sequential patterns in two phases. First, it finds the frequent associations using the axis features(s) as previously described. Then it generates the frequent serial patterns from these associations. Thus, our approach combines the associations among features and the sequential patterns among the records into a single rule.

Another interesting schema-level fact about audit records is that some essential features can be the *references* of other features. These reference features normally carry information about some "subject", and other features describe the "actions" that refer to the same "subject". For example, if we want to study the sequential patterns of connections to the same destination host, then *dst_host* is the "subject" and *service* is the action. In this case, we can designate *dst_host* as the *reference* feature. When forming an episode, our program tests the condition that, within the episode's minimal occurrences, the event records covered by its constituent item sets have the same reference feature value.

4. FEATURE CONSTRUCTION

We use the mined frequent episodes, which also contain associations among the features, from audit records as guidelines to construct temporal statistical features for building classification models. This process involves first identifying the "intrusion only" patterns, then parsing these patterns to define features accordingly. In this section, we use network connection data as an example to illustrate the feature construction process.

Raw *tcpdump* output are first summarized into network connection records using pre-processing programs, where each record has a set of "intrinsic" features. For example, the *duration*, *service*, *src_host* and *dst_host* (source and destination hosts), *src_port* (source port), *src_bytes* and *dst_bytes* (number of data bytes), a *flag* indicating normal or error status according to the protocols, etc., are intrinsic features of a single connection. Table 5 shows examples of connection records. Note that these "intrinsic" features are for general network analysis purposes, and not specific to intrusion detection.

Table 6. Example Intrusion Pattern

Frequent Episode	Meaning
$(flag = S0$, $service = http$, $dst\_host = victim)$, $(flag = S0$, $service = http$, $dst\_host = victim) \rightarrow (flag = S0$, $service = http$, $dst\_host = victim)$ $[0.93, 0.03, 2]$	93% of the time, after two *http* connections with $S0$ flag are made to host *victim*, within 2 seconds from the first of these two, the third similar connection is made, and this pattern occurs in 3% of the data

4.1 Identifying the Intrusion Patterns

We apply the frequent episodes program to both the exhaustively gathered normal connection dataset and the dataset that contains an intrusion. We then compare the resulting patterns to find the "intrusion only" patterns, i.e., those that exhibit only in the intrusion dataset. The details of the pattern comparison algorithm is described in [Lee et al. 1999b]. Briefly, since the number of patterns may be very large and there are rarely exactly matched patterns from two data sets, we used heuristic algorithms to automatically identify the intrusion only patterns. The idea is to first convert patterns into numbers in such a way that "similar" patterns are mapped to "closer" numbers. Then pattern comparison and intrusion pattern identification are accomplished through comparing the numbers and rank ordering the results. We devised an "encoding" procedure that converts each pattern into a numerical number, where the order of digit significance corresponds to the order of importance of the features. We used the following heuristic ordering on the importance of the features: *flag*, the axis feature, the reference feature, the rest of the essential attributes and then the rest of the features in alphabetical order. *flag* is considered as the most important in describing a pattern because it carries the summary information of the connection behavior with regard to the protocol specifications. Each unique feature value is mapped to a digit value in the encoding process. The "distance" of two patterns is then simply a number where each digit value is the digit-wise absolute difference between the two encodings. A "comparison" procedure computes the "intrusion" score for each pattern from the intrusion dataset, which is its lowest distance score against all patterns from the normal dataset, and outputs the user-specified top percentage patterns that have the highest intrusion scores as the "intrusion only" patterns.

As an example, consider the SYN flood attack records shown in Table 5. The attacker used many spoofed source addresses to send a lot of S0 connections (i.e., only the first SYN packet is sent) to a port (e.g., *http*) of the victim host in a very short time span (e.g., all in timestamp 1.1). Table 6 shows one of the top intrusion only patterns, produced using *service* as the axis feature and *dst_host* as the reference feature.

4.2 Constructing Features from Intrusion Patterns

Each of the intrusion patterns is used as a guideline for adding additional features into the connection records to build better classification models. We use the following automatic procedure for parsing a frequent episode and constructing features:

—Assume F_0 (e.g., *dst_host*) is used as the reference feature, and the width of the episode is w seconds.

—Add the following features that examine only the connections in the past w seconds that share the same value in F_0 as the current connection:

  —A feature that computes "the count of these connections";

  —Let F_1 be *service*, *src_dst* or *dst_host* other than F_0 (i.e., F_1 is an essential feature). If the same F_1 value (e.g., "http") is in all the item sets of the episode, add a feature that computes "the percentage of connections that share the same F_1 value as the current connection"; otherwise, add a feature that computes "the percentage of different values of F_1".

  —Let V_2 be a value (e.g., "S0") of a feature F_2 (e.g., $flag$) other than F_0 and F_1 (i.e., V_2 is a value of a non-essential feature). If V_2 is in all the item sets of the episode, add a feature that computes "the percentage of connections that have the same V_2"; otherwise, if F_2 is a numerical feature, add a feature that computes "the average of the F_2 values".

This procedure parses a frequent episode and uses three operators, *count*, *percent*, and *average*, to construct statistical features. These features are also temporal since they measure only the connections that are within a time window w and share the same reference feature value. The intuition behind the feature construction algorithm comes from the straightforward interpretation of a frequent episode. For example, if the same feature value appears in all the itemsets of an episode, then there is a large percentage of records that have the same value. We treat the essential and non-essential features differently. The essential features describe the *anatomy* of an intrusion, for example, "the same *service* (i.e., *port*) is targeted". The actual values, e.g., "http", is often not important because the same attack method can be applied to different targets, e.g., "ftp". On the other hand, the actual non-essential feature values, e.g., $flag = S0$, often indicate the *invariant* of an intrusion because they summarize the connection behavior according to the network protocols.

This SYN flood pattern shown in Table 6 results in the following additional features: a count of connections to the same *dst_host* in the past 2 seconds, and among these connections, the percentage of those that have the same *service*, and the percentage of those that have the "S0" $flag$.

4.3 Discussions

We examined the theoretical underpinnings of the feature construction process in [Lee 1999]. We outline the results here and explain why the features constructed from the intrusion patterns can be utilized to build more accurate classification models. First, the intrusion only patterns are the results of intrusion records. That is, the "intrusion" dataset must contain "intrusion records", i.e., unique records, unique sequences of records, or records or sequences with unique frequencies, in order for it to have intrusion only patterns. Second, for the temporal and statistical features constructed from the intrusion patterns, their values in the intrusion records that are responsible for resulting in the intrusion only patterns will be very different from the feature values in the normal connection records. For example, for the feature from the SYN flood pattern, *for the connections to the same destination* *host in the past 2 seconds, the percentage of those that have S0 flag*, normal records have values close to 0, but SYN flood records have values in the range of greater than 80%. The constructed features have high information gain because their value ranges can separate intrusion records from the normal records. In fact, they normally have higher information gain than the existing set of features. For example, the feature "flag" has very low information gain because some normal connections also have a "S0" value. As discussed in Section 3.1, a classification algorithm needs to select features with the highest information gain when computing a classification model. Therefore, when the features constructed from the intrusion patterns are added to the audit data, a more accurate classification model can be computed. This is precisely the purpose of our feature construction process.

An open problem is how to decide the right time window value w. We mine sequential patterns using different w values, for example, from 0.1 to 20 with an increment of 1, and plot the number of patterns generated at each run. Our experience shows that this plot tends to stabilize after the initial sharp jump. We call the smallest w in the stable region w_0. In [Lee and Stolfo 1998], we reported experiments of using different w values to calculate temporal 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.

5. EXPERIMENTS

In this section, we describe our experiments in building intrusion detection models on the audit data from the 1998 DARPA Intrusion Detection Evaluation Program. In these experiments, we applied the algorithms and tools of MADAM ID to process audit data, mine patterns, construct features, and build RIPPER classifiers.

We first describe the experiments on *tcpdump* data. The results of these experiments were submitted to DARPA and were evaluated by MIT Lincoln Lab. We then report recent experiments on *BSM* data, which were performed after the DARPA evaluation. We discuss our experiences and evaluate the strengths and weaknesses of MADAM ID.

5.1 Experiments on *tcpdump* Data

We participated in the DARPA Intrusion Detection Evaluation Program, prepared and managed by MIT Lincoln Lab. The objective of this study was to survey and evaluate the state of the art in intrusion detection research. A standard set of extensively gathered audit data, which includes a wide variety of intrusions simulated in a military network environment, is provided by DARPA. Each participating site was required to build intrusion detection models or tweak their existing system parameters using the training data, and send the results, i.e., detected intrusions on the test data back to DARPA for performance evaluation. We report our experience here.

5.1.1 *The DARPA Data.* We were provided with about 4 gigabytes of compressed *tcpdump* data of 7 weeks of network traffic. This data can be processed into about 5 million connection records of about 100 bytes each. The data contains the content (i.e., 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.

The data contains four main categories of attacks:

—DoS, denial-of-service, for example, ping-of-death, teardrop, smurf, SYN flood, etc.,

—R2L, unauthorized access from a remote machine, for example, guessing password,

—U2R, unauthorized access to local superuser privileges by a local unprivileged user, for example, various buffer overflow attacks,

—PROBING, surveillance and probing, for example, port-scan, ping-sweep, etc.

In addition, there were anomalous user behavior such as "a manager becomes (i.e., behaves like) a system administrator".

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

We used a Bro "connection finished" event handler to output a summarized record for each connection. Each connection record includes a set of "intrinsic" features shown in Table 7.

Table 7. Intrinsic Features of Network Connection Records

Feature	Description	Value Type
duration	Length (number of seconds) of the connection	Continuous
protocol_type	Type of the protocol, e.g. TCP, UDP, etc.	Discrete
service	Network service on the destination, e.g., http, telnet, etc.	Discrete
src_bytes	Number of data bytes from source to destination	Continuous
dst_bytes	Number of data bytes from destination to source	Continuous
flag	Normal or error status of the connection	Discrete
land	1 - connection is from/to the same host/port; 0 - otherwise	Discrete
wrong_fragment	Number of "wrong" fragments	Continuous
urgent	Number of urgent packets	Continuous

5.1.3 *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 used this information to select intrusion data to perform pattern mining and feature construction, and to label each connection record with "normal" or an attack type to create training data for building classification models.

Since the amount of audit data is huge, for example, some days have several millions of connection records due to some nasty DoS attacks, we did not aggregate all the connection records into a single training data set. Instead, we extracted all the connection records that fall within a surrounding time window of plus and minus 5 minutes of the whole duration of each attack to create a data set for each attack type. We also extracted sequences of normal connections records to create the normal data set that has the same distribution as the original data set.

5.1.4 *Manual and Automatic Feature Construction.* Following the feature construction approach described in Section 4, for each attack type, e.g., SYN flood, port-scan, etc., we performed pattern mining and comparison using its intrusion data set and the normal data set. We constructed features according to the top 20% intrusion-only patterns of each attack type. Here we summarize the temporal and statistical features automatically constructed by our system:

—The "same host" feature that examine only the connections in the past 2 seconds that have the same destination host as the current connection:
  —the count of such connections, the percentage of connections that have the same service as the current one, the percentage of different services, the percentage of SYN errors, and the percentage of REJ (i.e., rejected connection) errors;
—The "same service" features that examine only the connections in the past 2 seconds that have the same service as the current connection:
  —the count of such connections, the percentage of different destination hosts, the percentage of SYN errors, and the percentage of REJ errors.

We call these the "time-based traffic" features for connection records. They are summarized in Table 8.

Table 8. Traffic Features of Network Connection Records

Feature	Description	Value Type
count	Number of connections to the same host as the current connection in the past 2 seconds	Continuous
	the following features refer to these same-host connections	
serror_%	% of connections that have "SYN" errors	Continuous
rerror_%	% of connections that have "REJ" errors	Continuous
same_srv_%	% of connections to the same service	Continuous
diff_srv_%	% of connections to different services	Continuous
srv_count	Number of connections to the same service as the current connection in the past 2 seconds	Continuous
	the following features refer to these same-service connections	
srv_serror_%	% of connections that have "SYN" errors	Continuous
srv_rerror_%	% of connections that have "REJ" errors	Continuous
srv_diff_host_%	% of connections to different hosts	Continuous

There are several "slow" PROBING attacks that scan the hosts (or ports) using a much larger time interval than 2 seconds, for example, one in every minute or even one in every (few) hour(s). As a result, these attacks did not produce intrusion-only patterns with the time window of 2 seconds. We sorted the connection records by the destination hosts, and applied the same pattern mining and feature construction process. Instead of using a time window of 2 seconds, we now used a "connection" window of 100 connections, and constructed a mirror set of "host-based traffic" features as the time-based traffic features.

We discovered that unlike most of the DoS and PROBING attacks, the R2L and U2R attacks don't have any intrusion-only frequent patterns. This is because most of the DoS and PROBING attacks involve sending a lot of connections to some host(s) in a very short period of time, and therefore can have frequent sequential patterns that are different from the normal traffic. The R2L and U2R attacks are embedded in the data portions of the packets and normally involve only a single connection. Therefore, it is unlikely that they can have any unique frequent traffic patterns. In other words, our automatic feature construction process, which is based on frequent patterns of connection records, would fail to produce any features for these attacks.

Table 9. Content Features of Network Connection Records

Feature	Description	Value Type
hot	Number of "hot indicators"	Continuous
failed_logins	Number of failed login attempts	Continuous
logged_in	1 - successfully logged in; 0 - otherwise	Discrete
compromised	Number of "compromised" conditions	Continuous
root_shell	1 - root shell is obtained; 0 - otherwise	Discrete
su	1 - "su root" command attempted; 0 - otherwise	Discrete
file_creations	Number file creation operations	Continuous
shells	Number of shell prompts	Continuous
access_files	Number of write, delete, and create operations on access control files	Continuous
outbound_cmds	Number of outbound commands in a ftp session	Continuous
hot_login	1 - the login belongs to the "hot" list (e.g., *root*, *adm*, etc.); 0 - otherwise	Discrete
guest_login	1 - the login is a "guest" login (e.g., *guest*, *anonymous*, etc.) ; 0 - otherwise	Discrete

After studying the outcome of this mining process, we focussed our attention to the content of the connections. Ideally, we should apply data mining programs to compute patterns from the connection content data and construct appropriate features for R2L and U2R attacks. However, our current data mining algorithms cannot deal with unstructured data contents of IP packets. We instead relied on domain knowledge to define suitable features for R2L and U2R attacks. In the Bro 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 to indicate whether the data contents suggest suspicious behavior. These features are: number of failed logins, successfully logged in or not, whether logged in as root, whether a root shell is obtained, whether a su command is attempted and succeeded, number of access to access control files (e.g., "/etc/passwd", ".rhosts", etc.), number of compromised states on the destination host (e.g., file/path "not found" errors, and "Jump to" instructions, etc.), number of hot indicators, (e.g., access to system directories, creation and execution of programs, etc.), and number of outbound connections during a *ftp* session. These features are summarized in Table 9. Our approach here is to include an extensive set of indicators, and then let classification programs decide, from the vast amount

Table 10. Example "Traffic" Connection Records

label	service	flag	count	srv_count	rerror_%	diff_srv_%	...
normal	ecr_i	SF	1	1	0	1	...
smurf	ecr_i	SF	350	350	0	0	...
satan	user-level	REJ	231	1	85%	89%	...
normal	http	SF	1	0	0	1	...
...	...	...	...	...	...	...	...

Table 11. Example RIPPER Rules for DoS and PROBING Attacks

RIPPER Rule	Meaning
smurf :- count $\geq$ 5, srv_count $\geq$ 5, service = ecr_i.	If the service is ICMP echo request, and for the past 2 seconds, the number of connections that have the same destination host as the current one is at least 5, and the number of connections that have the same service as the current one is at least 5, then this is a smurf attack (a DoS attack).
satan :- rerror_% $\geq$ 83%, diff_srv_% $\geq$ 87%.	If for the connections in the past 2 seconds that have the same destination host as the current connection, the percentage of rejected connections is at least 83%, and the percentage of different services is at least 87%, then this is a satan attack (a PROBING attack).

of audit data, which minimal set of discriminating features should actually be used to identify intrusions.

5.1.5 *Detection Models.* It is evident from the feature construction process that different categories of intrusions require different sets of constructed features for detection purposes. We therefore built classification models using different feature sets:

—The "time-based traffic" model: each connection record contains the "intrinsic" and the "time-based traffic" features. Table 10 shows some example labeled connection records. The resultant RIPPER classifier detects the DoS and PROBING attacks. Table 11 shows some example RIPPER rules.

—The "host-based traffic" model: each connection record contains the "intrinsic" and the "host-based traffic" features. The resultant RIPPER classifiers detect the slow PROBING attacks.

—The "content" model: each connection record contains the "intrinsic" and the "content" features. Table 12 shows some example labeled connection records. The resultant RIPPER classifier detects the R2L and U2R attacks. Table 13 shows some example RIPPER rules.

These classification models each specializes for a certain type of intrusion. We then constructed a meta-level classifier to combine these detection models. Each meta-level training record consists of four features, the three predictions each from one of the base models, plus the true class label, i.e., "normal" or an attack type. RIPPER was then applied to learn the rules that combine the evidence from the "time-based traffic", "host-based traffic", and "content" classifiers to make a (fi-

Table 12. Example TCP Connection Records

label	service	flag	hot	failed_logins	compromised	root_shell	su	...
normal	ftp	SF	0	0	0	0	0	...
normal	telnet	SF	0	0	0	3	1	...
guess	telnet	SF	0	6	0	0	0	...
normal	telnet	SF	0	0	0	0	0	...
overflow	telnet	SF	3	0	2	1	0	...
normal	rlogin	SF	0	0	0	0	0	...
guess	telnet	SF	0	5	0	0	0	...
overflow	telnet	SF	3	0	2	1	0	...
normal	telnet	SF	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...

Table 13. Example RIPPER Rules for R2L and U2R Attacks

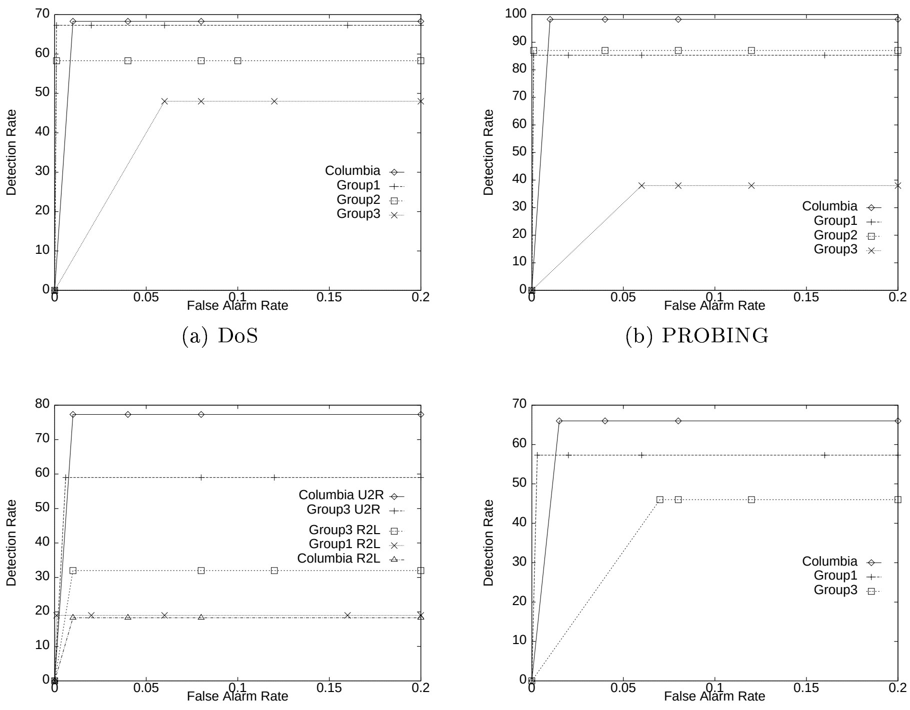
RIPPER Rule	Meaning
guess :- failed_logins $\geq$ 4.	If number of failed logins is at least 4, then this telnet connection is "guess", a guessing password attack.
overflow :- hot $\geq$ 3, compromised $\geq$ 2, root_shell = 1.	If the number of hot indicators is at least 3, the number of compromised conditions is at least 2, and a root shell is obtained, then this telnet connection is a buffer overflow attack.
...	...
normal :- true.	If none of the above, then this connection is "normal".

nal) prediction on a connection. The resulting meta-level rules basically use the predictions from the "content" model to detect R2L and U2R attacks, and the combination of "time-based traffic" and "host-based traffic" models to detect the DoS and (fast and slow) PROBING attacks. That is, the meta-classifier predicts a connection as an attack of R2L or U2R whenever the "content" model does so; and an attack of DoS or PROBING whenever the "time-based traffic" model does so, or whenever the "time-based traffic" model predicts "normal" but the "host-based traffic" model predicts a PROBING attack.

Table 14. Model Complexities

Model	# of Features in Records	# of Rules	# of Features Used in Rules
content	22	55	11
traffic	20	26	4+**9**
host traffic	14	8	1+**5**

Table 14 summarizes the complexity of the base models in terms of the number of features in a connection record, the number of RIPPER rules produced, and the number of distinct features actually used in the rules. The numbers in bold, for example, **9**, indicate the number of automatically constructed temporal and statistical features being used in the RIPPER rules. We see that for both the "traffic" and "host-based traffic" models, our feature construction process contribute the majority of the features. We should point out that not all features in the connection records were selected by RIPPER. This is because RIPPER, like most classification algorithms, has a built-in "feature selection" process to select the most discriminating and generalizable features according to their statistical significance, i.e., information gain, and performance on a hold-out test dataset that simulates the "unseen/future" data. Because of the large amount of audit data, a human expert is not able to manually gather and test various statistics, and thus tend to do a poor job in selecting the features. As a result, hand crafted "signature" rules tend to be very specific to a small intrusion data set. Alternative classification algorithms that compute underlying probability distributions may indeed require all features be evaluated in their resultant models. A crucial issue here is the tradeoff between model accuracy and model cost. The RIPPER output indicates that some features are irrelevant and hence we need not compute these at run-time, thus reducing the cost of detection. This is the subject matter of our ongoing research.



(a) DoS (b) PROBING

(c) U2R and R2L (d) Overall

Fig. 2. Performance of *tcpdump* misuse detection models: ROC curves on detection rates and false alarm rates.

5.1.6 *Results.* We report the performance of our detection models as evaluated by MIT Lincoln Lab. We trained our intrusion detection models, i.e., the base models and the meta-level classifier, using the 7 weeks of labeled data, and used them to make predictions on the 2 weeks of unlabeled test data (i.e., we were not told which connection is an attack). The test data contains a total of 38 attack types, with 14 types in test data only (i.e., our models were not trained with instances of these attack types, hence these are considered as "new" attack types).

Figure 2 shows the ROC curves of the detection models by attack categories as well as on all intrusions. In each of these ROC plots, the x-axis is the false alarm rate, calculated as the percentage of normal connections classified as an intrusion; the y-axis is the detection rate, calculated as the percentage of intrusions detected. A data point in the upper left corner corresponds to optimal performance, i.e., high detection rate with low false alarm rate. We compare here our models with other participants (denoted as Group 1 through 3) in the DARPA evaluation program[1]. These participating groups used knowledge engineering approaches to build their intrusion detection models. We can see from the figure that our detection model has the best overall performance, and in all but one attack category, our model is one of the best two. However, it is also clear that all models performed very poorly on R2L attacks. For all intrusions, an overall detection rate of below 70% is hardly satisfactory in a mission critical environment.

Table 15. Comparing Detection Rates (in %) on Old and New Attacks

Category	Old	New
DoS	79.9	24.3
PROBING	97.0	96.7
U2R	75.0	81.8
R2L	60.0	5.9
Overall	80.2	37.7

Although our models were intended for misuse detection, we had hoped that the features we constructed would be general enough so that the models can detect new variations of the known intrusions. Table 15 compares the detection rates of old intrusions and new intrusions. Here new intrusions refer to those that did not have corresponding instances in the training data. We see that our models were able to detect a large percentage of new PROBING and U2R attacks, but not as effective for new DoS and R2L attacks.

5.1.7 *Discussion.* PROBING attacks have relatively limited variance because they all involve making connections to a large number of hosts or ports in a given time frame. Likewise, the outcome of all U2R attacks is that a root shell is obtained without legitimate means, e.g., login as root, su to root, etc. Thus, for these two categories of attacks, given some representative instances in the training data, our data mining system was able to construct features that capture their general behavior patterns. As a result, our detection models can detect a high percentage of old and new PROBING and U2R attacks. On the other hand, DoS and R2L have a wide variety of behavior because they exploit the weaknesses of a large number of

[1]The tested systems produced binary output, hence, the ROC's are not continuous. These plots are duplicated from the presentation slides of a report given by Lincoln Lab in a DARPA PI meeting. The slides can be viewed on line via `http://www.cs.columbia.edu/~sal/JAM/PROJECT/MIT/mit-index.html`.

different network or system services. The features constructed based on the available attack instances are very specialized to the known attack types. Our detection models therefore missed a large number of new DoS and R2L attacks.

The results here are not entirely surprising since our models are *misuse* detection models. We need to use anomaly detection models on network traffic or system programs to guard against the new and diversified attacks. Anomaly detection is much more challenging than misuse detection. For example, we need to first decide whether we should build normal profile for each network service or a group of services, and for each host or a groups of hosts. The feature construction process will likely be more complex since unlike a relatively small number of intrusion-only patterns, normal network traffic can have a large number of variations. Network anomaly detection is an important problem and an active area of research that are we are pursuing.

5.1.8 *User Anomaly Detection.* Thus far we have discussed only the detection of attacks from outside the network. "Insiders" misusing their privileges can also seriously compromise security. These "insider attacks" are hard to detect because the insiders don't need to break-in. The goal of user anomaly detection is to determine whether the behavior of a user is normal (i.e., legitimate) or not.

It is often very difficult to classify a single event by a user as normal or abnormal because of 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. We used Bro event handlers to examine the telnet sessions, and extract the shell commands of the users. We further pre-processed the shell commands by replacing timestamps with am, pm, and nt (for night), and eliminated the input (i.e., contents) of edit and sendmail commands, and kept only the filename extensions. Table 3 shows examples of the processed command data. These shell command records were used for user anomaly detection.

Our initial exploratory approach is to mine the frequent patterns from the command data, and merge or add the patterns into an aggregate set to form the normal usage profile of a user. A new pattern 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 16 describes the consistent behavior of the 6 users for anomaly analysis. Note that since we were the only group that performed anomaly detection on the test data, Lincoln Lab did not evaluate our results. We report our experiments on the training data here.

We applied our frequent episode algorithms 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

Table 16. User Descriptions

User	Normal Activities
sysadm	Logs in as root, cats the password file, and runs commands such as top.
programmer1	Writes public domain C code, uses a vi editor, compiles the C code, reads and sends mails, and executes unix commands.
programmer2	A similar user profile, but works in afternoons and evenings.
secretary	Edits latex files, runs latex, reads mails, and sends mails.
manager1	Reads and sends mails
manager2	Reads mails.

Table 17. User Anomaly Description

User	Anomaly Description
programmer2	Logs in from beta
secretary	Logs in at night
sysadm	Logs in from jupiter
programmer1	Becomes a secretary
secretary	Becomes a manager
programmer1	Logs in at night
sysadm	**Becomes a programmer**
manager1	Becomes a sysadm
manager2	Logs in from pluto

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 used the first 4 weeks as a data gathering period, during which we simply merged 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 used the 5th week as the training period, during which we compared the patterns from each session to the profile of the time segment. We recorded the normal range of the similarity scores during this week. The data in the 6th week has some user anomalies, as described in Table 17. For each of the anomalous sessions, we compared its patterns against the original user's profile, and then compared the resulting similarity score against the recorded normal range of the same time segment. In Table 18, 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 described in Table 17. We see that all anomalous sessions can be clearly detected since their similarity scores are much smaller than the normal range. For example, the row in bold in Table 18 shows that, when the sysadm becomes programmer, his/her patterns have 0 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.

Once user abnormal behavior is observed, we need to investigate the nature of the anomaly. We report our experiments on finding out how "user job functions" are

Table 18. Similarity Against User's Own Profile: in Normal Use and in the Anomaly described in Table 17

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 19. Similarity Against Group Profiles: in Normal Use and in the Anomaly described in Table 17

	Normal				Anomaly			
User	P	S	M	SA	P	S	M	SA
p2	(0.33, 0.71)	(0.04, 0.11)	(0.00, 0.03)	(0.00, 0.07)	0.00	0.00	0.00	0.00
s	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	0.04	0.00	0.00	0.00
sa	(0.00, 0.00)	(0.00, 0.00)	(0.00, 0.00)	(0.51, 0.81)	0.00	0.00	0.00	0.00
p1	(0.12, 0.57)	(0.06, 0.09)	(0.00, 0.00)	(0.04, 0.14)	0.04	**0.11**	0.00	0.00
s	(0.02, 0.18)	(0.08, 0.73)	(0.00, 0.00)	(0.00, 0.00)	0.17	0.00	**0.50**	0.00
p1	(∞, ∞)	(∞, ∞)	(∞, ∞)	(∞, ∞)	0.27	0.00	0.00	0.00
sa	(0.00, 0.00)	(0.00, 0.00)	(0.00, 0.00)	(0.51, 0.81)	**0.24**	0.03	0.00	0.00
m1	(0.14, 0.17)	(0.00, 0.00)	(0.29, 1.00)	(0.00, 0.00)	0.02	0.00	0.00	0.61
m2	(0.50, 1.00)	(0.00, 0.00)	(1.00, 1.00)	(0.00, 0.00)	0.00	0.00	0.00	0.00

violated. The problem can be stated as follows. Assume that there are n possible groups of users according to their job functions. When a user in group i does not behave according to the group profile (i.e., the normal job functions), we want to identify which group, e.g., group j, the user has become. That is, we want to know what "illegal" job functions the user has performed.

In our experiments, we first built group profiles for the p group (the programmers), the s group (the secretary), the m (the managers), and the sa group (the sysadm). From the data of the first 4 weeks, the patterns of all the users of a group were aggregated to form the group profile. The data of the 5th week was used to establish the range of similarity measures for all the users of each group. The user anomalies described in Table 17 include "illegal job function" cases during the 6th week: programmer1 becomes a secretary, secretary becomes a manager, and sysadm becomes a programmer. Table 19 compares the similarity measure of each user in an anomalous session with his/her normal similarity range gathered for the same time segment. From the normal similarity measures of each user with respect to the four groups, we can see that each user indeed has the largest similarity measure with his/her own group. From the similarity measures of the user anomalies, as the bold entries in Table 19 show, for each "illegal job function" case, the similarity measure of the targeted group is the largest. For example, when sysadm becomes a programmer, the similarity measure with the p group, 0.24, is the largest, and the similarity measure with the sa group is outside the normal range.

In summary, although formal evaluation statistics are not available to determine

Table 20. Example BSM Event Records

time	auid	sid	event	pid	obname	...	ruid	euid
08:05:22	0	0	inetd_connect	0	?	...	0	0
...	...	...	...	...	...	...	...	...
08:05:22	-2	0	execve	415	/usr/sbin/in.telnetd	...	0	0
...	...	...	...	...	...	...	...	...
08:05:31	2104	417	setaudit	417	?	...	0	0
...	...	...	...	...	...	...	...	...
08:05:31	2104	417	chdir	418	/home/tristank	...	2104	2104
...	...	...	...	...	...	...	...	...

the error rates of our approach in user anomaly detection, the initial results are encouraging. We believe that our approach is worthy of future study.

5.2 Experiments on BSM Data

The DARPA data also contains Solaris BSM (Basic Security Module) [SunSoft 1995] audit data for a designated host, *pascal*. In this section, we describe our experiments in building host-based intrusion detection models using BSM data. The purpose of these experiments was to show that our algorithms for pattern mining and feature construction are not specific to a particular audit data source, e.g. *tcpdump*. We also wanted to investigate whether combining models from *tcpdump* and BSM can result in better detection performance.

When BSM is enabled in a host machine, there exists an *audit trail* for the host. An audit trail is a time-ordered sequence of actions that are audited on the system, and consists of one or more *audit files*. Each *audit record* in an audit file describes a single *audit event*, which can be a kernel event, i.e., a system call, or a user-level event, i.e., a system program (e.g., *inetd*, *in.rshd*, etc.) invocation.

We define *audit session* here as the collection of all audit events of an "incoming" or "outgoing" session on the host. Examples of these host sessions include *login* (e.g., terminal login, *telnet* login, *rlogin*, etc.), *rsh*, *ftp*, and *sendmail*, etc. It is easy to see that each host session often corresponds to a network connection. We can therefore correlate the predictions on host sessions by a host-based intrusion detection model with the predictions on the corresponding connections by a network intrusion detection model, to yield a higher accuracy.

As in the case of building network intrusion detection models, we also need to first perform a sequence of data preprocessing tasks on the raw BSM data. We extended the preprocessor component of USTAT [Ilgun 1992] to process the binary BSM data into ASCII event data. Table 20 shows examples of the event records. Here a "?" means the value is not given in the original BSM audit record. Each event record contains a number of basic features, defined in Table 21.

We developed a program to process the event data into session records. A brief description of the procedure is the following:

—Watch for the beginning of a session, which is the execution of
  —the *inetd_connect* event (for *telnet*, *rlogin*, *rsh*, etc.), or
  —the *execve* event on a system program *in.fingerd* (for incoming *finger* request) or *finger* (outgoing), *mail.local* (incoming) or *sendmail* (outgoing), *ftpd* (incoming) or *ftp* (outgoing), etc.

Table 21. Features of BSM Event Records

Feature	Description	Value Type
time	Timestamp of the event	Discrete
auid	Audit user id, inherited by all child processes started by the user's initial process of a session	Discrete
sid	Audit session id, assigned for each login session and inherited by all descendant processes	Discrete
event	Audit event name	Discrete
pid	Process id of the event	Discrete
obname	The name of the object, i.e., full path of the file, that the event operates on	Discrete
arg1 - arg4	Arguments of the system call	Discrete
text	Short information of the event, e.g., "successful login"	Discrete
error_status	Error status of the event	Discrete
return_value	Return value of a system call event	Discrete
tmid	Terminal id (port and ip address) of the event	Discrete
ip header	The source and destination ip addresses and ports for the network connection handled by the event	Discrete
socket	The local and remote ip addresses and ports of the socket used by the event	Discrete
ruid	The real user id of the event	Discrete
rgid	The real group id of the event	Discrete
euid	The effective user id of the event	Discrete
egid	The effective group id of the event	Discrete

—Record the *setaudit* event, which assigns the *auid* (audit user id) and *sid* (audit session id) of a session.

—Examine all audit records that share the same combination of *auid* and *sid* to summarize a number of session features, which are described in Section 5.2.1.

—Records the termination of a session.

The DARPA BSM data contains for each day about 500 sessions on host *pascal*. The vast majority of the intrusions in the BSM data are U2R buffer overflow attacks.

5.2.1 *Defining Session Features.* We experimented with feature construction for session records. We first computed frequent patterns from the event records. Here each dataset prepared for pattern mining contains all event records of a session that have positive *auid* and *sid* values. That is, we are only interested in events that are "accountable" to the users.

Since we are looking for general rather than session-specific event patterns, we first removed *auid* and *sid* from the datasets. We also replaced *ruid* and *euid* with a flag *same_reid* to indicate whether *ruid* agrees with *euid*. We designated *event* as the *axis* attribute since it is obviously the most essential attribute in describing event data.

After the initial few rounds of experiments, we discovered that the patterns are all related to very specific *obname* or *event* values. There are many kernel events (system calls) that cannot be directly linked to user-level commands. We reasoned that for intrusion detection purposes, we only need to analyze user-level commands and their operations on the file system. We therefore only kept the following types of event records: read, write, create, delete, execute, change owner or permission, rename, and link. The *event* value of each event record is replaced by the appropriate type name, e.g., *open_r* is replaced by *read*. We only kept the original *obname* if the event is *execute*, otherwise we used "user" to replace all *obname* values that indicate files in the user's directories, and "system" to replace the *obname* values that indicate files in the system's directories. We also removed all event records that have "?" (i.e., missing) *obname* values.

We aggregated event patterns of all normal sessions into a normal pattern set. And for each U2R session, we mined its event patterns and compared them with the normal patterns. We used the top 20% of intrusion-only patterns for each U2R attack, for example,

$$\begin{aligned}&(event = execute, obname = /home/tristank/ffbexploit, same\_reid = 1),\\&(event = execute, obname = /usr/sbin/ffbconfig, same\_reid = 0) \rightarrow\\&(event = execute, obname = /usr/bin/ksh, same\_reid = 0)\end{aligned}$$

and

$$\begin{aligned}&(event = execute, obname = /usr/bin/pwd, same\_reid = 0) \rightarrow\\&(event = read, obname = home, same\_reid = 0)\end{aligned}$$

These patterns are very "unique" because in the normal pattern set, patterns with $same\_reid = 0$ are those related to *read* operations only, for example,

$$\begin{aligned}&(event = read, obname = system, same\_reid = 0),\\&(event = read, obname = system, same\_reid = 0) \rightarrow\\&(event = read, obname = sytem, same\_reid = 0)\end{aligned}$$

and

$$\begin{aligned}&(event = read, obname = home, same\_reid = 0) \rightarrow\\&(event = execute, obname = /usr/bin/cat, same\_reid = 1)\end{aligned}$$

We could have used a mechanical (i.e., automatic) pattern parsing and feature construction procedure for the intrusion-only patterns of the above forms. For example, we can add features that record the executions of the relevant specific events as described by the patterns. However, we very quickly realized that many U2R buffer overflow attacks share the same characteristics in their intrusion-only patterns. For example, another attack has the following intrusion-only pattern:

$$\begin{aligned}&(event = execute, obname = /home/tristank/formatexploit, same\_reid = 1),\\&(event = execute, obname = /usr/bin/fdformat, same\_reid = 0) \rightarrow\\&(event = execute, obname = /usr/bin/ksh, same\_reid = 0)\end{aligned}$$

These buffer overflow patterns indicate that there is an execution of a user program, follow by a SUID (setuid) system utility, and finally a shell in SUID state (i.e., a root shell with effective user ID as root and different from the real user ID). We need to construct features that capture the general behavior of the attack method and the final outcome, rather than the specific system utilities being exploited. However, since the event data contains very low level and specific information of the operating system, we need to use domain knowledge to interpret the patterns to construct the more abstract and general features. Although our experiments here showed the limitations of fully automatic feature construction when dealing with low level event data, we believe that the intrusion-only patterns resulted from pattern mining and comparison can still provide a very helpful starting point for the manual feature definition process.

Table 22. Features of BSM Session Records

Feature	Description	Value Type
duration	Length (number of seconds) of the session	Continuous
service	Operating system or network service, e.g., *telnet*, responsible for this session	Discrete
logged_in	Whether the user successfully logged in (when using *telnet*, rsh, etc).	Discrete
failed_logins	Number of failed login attempts	Continuous
process_count	Number of processes in the session	Continuous
suid_sh	Whether a shell is executed in suid state	Discrete
suid_p	Whether a suid system program is executed	Discrete
user_p	Whether a user program is executed	Discrete
su_attempted	Whether a su command is issued	Discrete
access_files	Number of write, delete, and create operations on access control files	Continuous
file_creations	Number of file creations	Continuous
hot_login	Whether the login belongs to the "hot" list	Discrete
guest_login	Whether the login belongs to the "guest" list	Discrete

We defined a set of features, described in Table 22, for the BSM session records. Some of the features, i.e., those in bold, are from the buffer overflow patterns, while others are similar to the "content" features described in Session 5.1.3.

5.2.2 *Misuse Detection Models on* BSM *Data.* We labeled each BSM session record as "normal" or an intrusion name using the DARPA supplied list files. BSM session records from all 7 weeks were aggregated into a single dataset for training a misuse detection model. Table 23 shows some examples of the labeled BSM session records.

Table 23. Example BSM Session Records

label	service	suid_sh	suid_p	user_p	file_creations	...
normal	smtp	0	0	0	0	...
normal	telnet	0	1	1	3	...
normal	telnet	0	1	0	0	...
buffer_overflow	telnet	1	1	1	2	...
normal	ftp	0	0	0	0	...
wraz_master	ftp	0	0	0	42	...
...	...	...	...	...	...	...

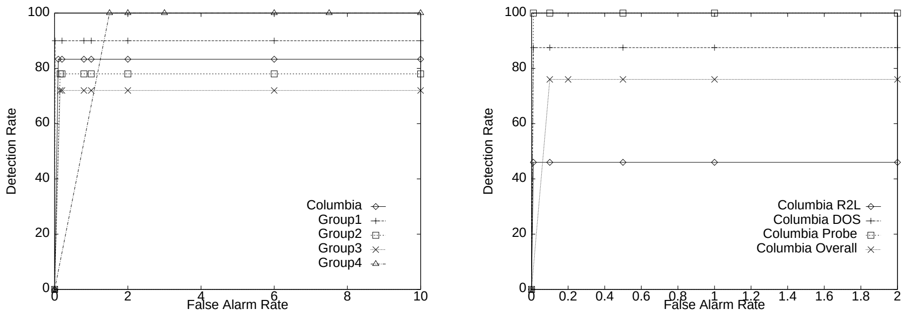
RIPPER was applied to the dataset to learn a set of classification rules. Table 24 shows some examples of these detection rules.

We evaluated the performance of the rule set on the test data using the list files provided by DARPA (available after the completion of the official evaluation). Figure 3 shows the ROC curves of the detection models on BSM data. Here sub-figure (a) compares our model with other participants in the DARPA Evaluation, in terms of performance in detecting the U2R attacks (DARPA only evaluated performance on U2R). Sub-figure (b) shows the performance of our model in detecting DoS,

Table 24. Example RIPPER Rules for BSM Session Records Shown in Table 23

RIPPER Rule	Meaning
buffer_overflow :- suid_sh = 1.	If a shell is executed in the SUID state, then this is a buffer overflow attack.
wraz_master :- file_creations ≥ 40, service = ftp, guest_login = 1.	If the service is ftp, the user logs in as a guest (*anonymous*), and the number of file creation operations is at least 42, then this is a wraz attack (in which the attacker logs to anonymous FTP site, creates a hidden directory, and uploads a lot of files, often pirated copies of software, for others to download).

R2L, and PROBING attacks, as well as the overall performance (when all attacks are considered).



(a) U2R (b) DoS,R2L,PROBING, and overall

Fig. 3. Performance of BSM misuse detection models: ROC curves on detection rates and false alarm rates.

The BSM model here may seem to have slightly better performance than the *tcpdump* models reported in Section 5.1.5. However, the *tcpdump* models were computed using the *tcpdump* data of the entire network while the BSM model was constructed using the BSM data, which is available from only one host and contains much fewer attacks and normal sessions. We should note that the BSM model, like the *tcpdump* models, has good performance in detecting PROBING, U2R, and DoS attacks, but very poor performance in detecting R2L attacks. In fact, when we compared the predictions made by the BSM model and the *tcpdump* models, we found that they simply agree with each other's predictions on the pairs of corresponding host session and network connection. That is, the BSM misuse detection model, as we have constructed, provides no additional predictive power. This is because:

—by the nature of misuse detection, a truly innovated intrusion, e.g., one that uses a service that was not modeled, can go undetected no matter what audit data source is used for analysis; and

—the features used in the BSM model and the *tcpdump* models are very similar, that is, the models are looking for similar set of evidences, although in different data sources.

While this seems discouraging if we have hoped for an accuracy improvement by combining the two models, this is in fact encouraging if our goal is to combine a light-weight *tcpdump* model that only checks the IP headers with a number of host-based models that monitor the operating system activities. Our experiments in meta-learning, where a combined model was computed based on a model for *tcpdump* header-only connection data and a model for BSM host session data, indeed showed that the same level of accuracy was maintained as using a heavy-weight *tcpdump* model that also checks the IP data contents.

6. CONVERTING LEARNED MODELS TO REAL-TIME IDS MODULES

Effective intrusion detection should be in real-time to minimize security compromises. We therefore need to study how our models, produced off-line by MADAM ID, can be converted to modules of real-time IDSs, and how they perform in a real-time environment. We are developing a system for translating RIPPER rules into real-time detection modules of NFR (Network Flight Recorder), a system that includes a packet capturing engine and N-code programming support for specifying packet "filtering" logic.

NFR offers a fairly simple framework for network monitoring. It sniffs packets from the network, reassembles them, and then passes them to filter functions for further processing, e.g., calculating network traffic statistics. Filter functions are written in N-code, a preemptive, event-driven scripting language that supports a wide range of operations on network packet data. In order to use NFR for network intrusion detection, one needs to implement an extensive set of N-code filters that look for evidences in the network packets.

We seek a mechanical means of converting our off-line detection models into real-time modules. Our approach is to first implement all the required features used in the RIPPER ruleset as N-code "feature filters", and then implement a translator that can automatically convert each RIPPER rule into an N-code "rule filter". For example, a detection rule,

```
pod :- wrong_fragment >= 1, protocol_type = icmp.
```

can be automatically converted into the following N-code:

```
filter pod () {
   if (wrong_fragment() > 1 && protocol_type () == icmp)
       alarm (``ping_of_death'');
}
```

as long as the features, i.e., wrong_fragment and protocol_type, have been implemented as N-code filter functions in NFR.

The advantage of this approach is that when a real-time IDS such as NFR is shipped with MADAM ID to a customer site, MADAM ID can process the local audit data and compute the site-specific intrusion detection rules, which are then automatically converted into N-code "rule filters".

Although often ignored in off-line analysis, efficiency is a very important consideration in real-time intrusion detection. Specifically, in off-line analysis, it is implicitly assumed that all connections have already finished, therefore, we have the luxury to compute all the features and check the detection rules one by one; whereas in real-time detection, we need to detect and respond to an intrusion as soon as it happens, often during an on-going connection. We are therefore studying how to generate efficient real-time execution plan for a detection ruleset. In particular, we find that many intrusions have cheap "necessary" conditions, which contain features that are easy to compute and are available early in the connection. For example, SYN flood has a necessary condition "flag=S0"; whereas its sufficient condition includes checking the feature "for the connections to the same destination host as the current connection in the past 2 seconds, the percentage of those that have the S0 flag". The necessary condition is cheaper since the feature "flag" relies on only the information of the current connection; whereas the feature for the sufficient condition is computed by looking up a number of connection records. Note that a violation of the necessary condition means that there is no need to check the sufficient condition for the intrusion. For example, if "flag is not S0", we know that the connection can not be a SYN flood. Therefore, by checking the violations of the cheap necessary conditions first and filtering out a large number of unnecessary checking of ID rules, the overall execution time of the ruleset can be reduced.

We have designed and implemented an algorithm for finding the necessary conditions for intrusions, and are implementing the ruleset filtering algorithm in NFR.

7. RELATED WORK

Network intrusion detection has been an on-going research area [Mukherjee et al. 1994]. More recent systems, e.g. Bro [Paxson 1998], NFR [Network Flight Recorder Inc. 1997], and EMERALD [Porras and Neumann 1997] all made extensibility their primary design goals. Both Bro and NFR provide high-level scripting languages for codifying the site-specific intrusion detection rules, which are executed in run-time as event handlers by the packet filtering and re-assembly engines. Our research focuses on developing methods for constructing intrusion detection models. Our motivation is to use robust IDSs such as Bro and NFR as the building blocks, and provide a framework so that site-specific models can be computed and installed automatically.

EMERALD provides an architecture to facilitate enterprise-wide deployment and configuration of intrusion detectors. The meta-learning mechanism in our framework is designed to automate the process of learning a "resolver", which is needed to combine the alarms from the distributed detectors to make a determination of the state of the (entire) network. The meta-learning results reported in this paper are preliminary. We will study how to incorporate network configuration information into the meta-learning process.

There are several learning-based research efforts in intrusion detection. Warrender et al. [Warrender et al. 1999] showed that a number of machine-learning approaches, e.g., rule induction, Hidden Markov Model, etc., can be used to learn concise and generalizable representation of the "self" identity of a system program, which is the short sequences of run-time system calls made by the program [Forrest et al. 1996]. These learned models were shown to be able to accurately detect anomalies caused by exploits on the system programs. Ghosh et al. [Ghosh and Schwartzbard 1999] showed that, using program activity data (e.g., system calls, arguments, return values, and permissions, etc.) from BSM audit logs, Artificial Neural Networks can be used to learn anomaly and misuse detection models for system programs. Our research aims to develop algorithms and techniques that can be used to build ID models for both networks and hosts. Towards this end, we design the data mining algorithms to be independent of audit data sources. We concentrate our effort on the feature construction process because we believe this is the most critical step in the process of building ID models.

Lane et al. developed algorithms for analyzing user shell commands and detecting anomalies [Lane and Brodley 1999]. 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.

8. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we outlined a data mining framework for constructing intrusion detection models. The key idea is to first apply data mining programs to audit data to compute frequent patterns, extract features, and then use classification algorithms to compute detection models. To facilitate adaptability and extensibility, we proposed the use of meta-learning as a means to construct a combined model that incorporate evidence from multiple base models.

We extended the basic association rules and frequent episodes algorithms to accommodate the special requirements in analyzing audit data. Our experiments showed that the frequent patterns mined from audit data can be used as reliable user anomaly detection models, and as guidelines for selecting temporal statistical features to build effective classification models. Results from the 1998 DARPA Intrusion Detection Evaluation Program showed that our detection models performed as well as the best systems built using the manual knowledge engineering approaches.

Anomaly detection models are the only means to detect the truly "innovative" intrusions. Therefore, for future work, we will develop algorithms for learning network anomaly detection models.

IDSs need to maximize user-defined security goals while minimizing costs. This requires that ID models be sensitive to cost factors, that at the minimum should include the development cost, the operational cost, i.e., the needed resources, the cost of damages of an intrusion, and the cost of detecting and responding to a potential intrusion. For example, when the response cost of an intrusion exceed its damage cost, the ID models may choose to ignore, i.e., not to response to, the intrusion. It is particularly important to direct the limited resources, i.e., the computing and human resources, to detect and respond to the most damaging intrusions when the adversaries are launching a large amount of automated attacks in an attempt to overload the IDS. That is, we need ID models that can dynamically determine what are the most cost-saving actions that should be taken. We will extend our framework so that our algorithms can incorporate the user-defined cost factors and policies to compute cost-sensitive ID models.

9. ACKNOWLEDGMENTS

This research is supported in part by grants from DARPA (F30602-96-1-0311). We wish to thank our colleagues at Columbia University, Kui Mok, Chris Park, Matt Miller, Wei Fan, and Andreas Prodromidis, for their help and encouragement. We also wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (1993), pp. 207–216.

ALLEN, J., CHRISTIE, A., FITHEN, W., MCHUGH, J., PICKEL, J., AND STONER, E. 2000. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, CMU/SEI.

ANDERSON, D., FRIVOLD, T., AND VALDES, A. 1995. Next-generation intrusion detection expert system (NIDES): A summary. Technical Report SRI-CSL-95-07 (May), Computer Science Laboratory, SRI International, Menlo Park, California.

CHAN, P. K. AND STOLFO, S. J. 1993. Toward parallel and distributed learning by meta-learning. In *AAAI Workshop in Knowledge Discovery in Databases* (1993), pp. 227–240.

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

FAYYAD, U., PIATETSKY-SHAPIRO, G., AND SMYTH, P. 1996. The KDD process of extracting useful knowledge from volumes of data. *Communications of the ACM 39*, 11 (November), 27–34.

FORREST, S., HOFMEYR, S. A., SOMAYAJI, A., AND LONGSTAFF, T. A. 1996. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (Los Alamitos, CA, 1996), pp. 120–128. IEEE Computer Society Press.

GHOSH, A. K. AND SCHWARTZBARD, A. 1999. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium* (August 1999).

ILGUN, K. 1992. USTAT: A real-time intrusion detection system for Unix. Master's thesis, University of California at Santa Barbara.

ILGUN, K., KEMMERER, R. A., AND PORRAS, P. A. 1995. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering 21*, 3 (March), 181–199.

JACOBSON, V., LERES, C., AND MCCANNE, S. 1989. *tcpdump*. available via anonymous ftp to ftp.ee.lbl.gov.

KO, C., FINK, G., AND LEVITT, K. 1994. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference* (December 1994), pp. 134–144.

KUMAR, S. AND SPAFFORD, E. H. 1995. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference* (1995), pp. 194–204.

LANE, T. AND BRODLEY, C. E. 1999. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security 2*, 3 (August), 295–331.

LEE, W. 1999. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. Ph. D. thesis, Columbia University.

LEE, W. AND STOLFO, S. J. 1998. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium* (San Antonio, TX, January 1998).

LEE, W., STOLFO, S. J., AND MOK, K. W. 1999a. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy* (May 1999).

LEE, W., STOLFO, S. J., AND MOK, K. W. 1999b. Mining in a data-flow environment: Experience in network intrusion detection. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99)* (August 1999).

LUNT, T. 1993. Detecting intruders in computer systems. In *Proceedings of the 1993 Conference on Auditing and Computer Technology* (1993).

LUNT, T., TAMARU, A., GILHAM, F., JAGANNATHAN, R., NEUMANN, P., JAVITZ, H., VALDES, A., AND GARVEY, T. 1992. A real-time intrusion detection expert system (IDES) - final technical report. Technical report (February), Computer Science Laboratory, SRI International, Menlo Park, California.

MANNILA, H. AND TOIVONEN, H. 1996. 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).

MANNILA, H., TOIVONEN, H., AND VERKAMO, A. I. 1995. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining* (Montreal, Canada, August 1995).

MITCHELL, T. 1997. *Machine Learning*. McGraw-Hill.

MUKHERJEE, B., HEBERLEIN, L. T., AND LEVITT, K. N. 1994. Network intrusion detection. *IEEE Network*.

NETWORK FLIGHT RECORDER INC. 1997. Network flight recorder. http://www.nfr.com.

PAXSON, V. 1998. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium* (San Antonio, TX, 1998).

PORRAS, P. A. AND NEUMANN, P. G. 1997. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference* (Baltimore MD, October 1997).

STOLFO, S. J., PRODROMIDIS, A. L., TSELEPIS, S., LEE, W., FAN, D. W., AND CHAN, P. K. 1997. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (Newport Beach, CA, August 1997), pp. 74–81. AAAI Press.

SUNSOFT. 1995. *SunSHIELD Basic Security Module Guide*. Mountain View, CA: SunSoft.

WARRENDER, C., FORREST, S., AND PEARLMUTTER, B. 1999. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy* (May 1999).