Applying Data Mining to Intrusion Detection: the Quest for Automation, Efficiency, and Credibility

Wenke Lee
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
wenke@cc.gatech.edu

ABSTRACT

Intrusion detection is an essential component of the layered computer security mechanisms. It requires accurate and efficient models for analyzing a large amount of system and network audit data. This paper is an overview of our research in applying data mining techniques to build intrusion detection models. We describe a framework for mining patterns from system and network audit data, and constructing features according to analysis of intrusion patterns. We discuss approaches for improving the run-time efficiency as well as the credibility of detection models. We report the ideas, algorithms, and prototype systems we have developed, and discuss open research problems.

General Terms

Intrusion detection

Keywords

Feature construction, model efficiency, Bayesian detection rate

1. INTRODUCTION

As the Internet plays an increasingly important role in our society, criminals and enemies have been devising and launching sophisticated attacks motivated by financial, political, and even military objectives. We therefore must protect our network infrastructure.

Contrary to the myth that there can be a panacea in security, no single technology alone is the answer. Security is a process (or a chain) that is as secure as its weakest link; and flaws in hardware, software, and networks, as well as human errors can all lead to security failure [18]. Experience has taught us that we need to deploy defense-in-depth or layered network security mechanisms, which include these necessary technologies: security policy, vulnerability scanning and patching, access control and authentication, encryption, program wrappers, firewalls, *intrusion detection* (ID), and intrusion response and tolerance.

An intrusion detection system (IDS) collects and monitors operating system and network activity data, and analyzes the information to determine whether there is an attack occurring. There are two major categories of analysis: *misuse detection* and *anomaly detection*. Misuse detection uses the "signatures" of known attacks, i.e., the patterns of attack behavior or effects, to identify a matched activity as an attack instance. By definition, misuse detection is not effective against new attacks, i.e., those that do not have known signatures. Anomaly detection uses established normal profiles, i.e., the expected behavior, to identify any unacceptable deviation as the result of an attack. Anomaly detection is intended for catching new attacks. However, new legitimate behavior can also be falsely identified as an attack, resulting in a false alarm. Reports of attacks can trigger response actions (e.g., termination of the offending connections) or further investigation by security staff.

Most intrusion detection approaches rely on analysis of system and network *audit data*. Network traffic can be recorded using "packet capturing" utilities (e.g., `libpcap` [16]), and operating system activities can be recorded at the system call level (e.g., `BSM` [20]). A basic premise here is that when audit mechanisms are enabled, distinct evidence of legitimate activities and intrusions will be manifested in the audit data. In other words, instead of (statically) analyzing (all source codes of) complex software, intrusion detection uses a more practical approach of analyzing the audit records of run-time activities of networks and systems (and users).

At an abstract level, an intrusion detection system (IDS) extracts *features*, i.e., the individual pieces of evidence, from the system event-level or network packet-level audit data, and uses some modeling and analysis algorithms to reason about the available evidence. Traditionally, IDSs are developed by knowledge-engineering. Expert knowledge or intuition of networks, operating systems, and attack methods are used to select the features and hand-craft the detection rules. Given the complexities of today's network environments and the sophistication of the increasingly hostile attackers, the so-called expert knowledge is often very limited and unreliable.

On the other hand, data mining approaches can be used to extract features and compute detection models from the vast amount of audit data. The features computed from data can be more objective than the ones hand-picked by experts. The inductively learned detection models can be more generalizable than hand-coded rules (that is, they can have better performance against new variants of known normal behavior or intrusions). Therefore, data mining approaches can play an important role in the process of developing an IDS. We need to point out that data mining should *complement* rather than exclude the use of expert knowledge. Our objective should be to provide the tools, grounded on sound statistics and machine learning principles, for IDS developers to construct better ID models quickly and easily. For example, experts can view and edit the patterns and rules produced by data mining approaches, and translate them into efficient detection modules.

Not unlike other data mining applications, characteristics of audit data and requirements of intrusion detection present both opportunities and challenges to data mining research. In this paper, we give an overview of the main research issues in applying data mining techniques to build intrusion detection models. These include extracting and constructing features from audit data, computing efficient models, and improving the usability and credibility of the detection models. We describe the ideas, approaches and prototype systems we have developed over the past couple of years, and discuss future research directions.

1.1 Related Work

Several influential research IDSs were developed from mid-80's to mid-90's. STAT [7] and IDIOT [8] are misuse detection systems. NIDES [2] has an anomaly detection subsystem. These systems and most of the later research and commercial systems are developed using a pure knowledge-engineering process.

In recent years, there have been several learning-based or data mining-based research efforts in intrusion detection. Warrender et al. [22] showed that a number of machine-learning approaches, e.g., rule induction, can be used to learn the normal execution profile of a program, which is the short sequences of its run-time system calls made. These learned models were shown to be able to accurately detect anomalies caused by exploits on the programs. Lane and Brodley developed machine learning algorithms for analyzing user shell commands and detecting anomalies of user activities [9]. Ghosh et al. [6] 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. A team of researchers at Columbia University have been working on data mining-based intrusion detection since 1996 (see Stolfo et al. [19] for an overview). The main capabilities developed in this research include: pattern mining and feature construction, cost-sensitive modeling for efficient run-time model execution, anomaly detection, learning over noisy data, and correlation analysis over multiple of data streams. The ADAM project at George Mason University is developing anomaly detection algorithms based on automated audit data analysis. (see `http://ise.gmu.edu/~dbarbara/adam.html`)

2. THE QUEST FOR AUTOMATION

The main motivation of using data mining techniques to build intrusion detection models is *automation*. The intuition is that consistent patterns of normal behavior and distinct patterns of an intrusion can be computed from historical (or training) audit data, and then be used to detect future instances of the intrusion.

We can think of intrusion detection as a classification problem, i.e., a problem of labeling or predicting new (unseen) audit data as belonging to an intrusion class, the normal class, or the abnormal (probably unknown intrusion) class.

Table 1: The Need for Feature Construction. The original (standard) features are insufficient to distinguish the "SYN flood" connections from the normal ones. A specific feature for detecting "SYN flood" is added.

time	dst	service	flag	...	**%SO**	class
1.01	H1	`http`	S0	...	**70**	SYN_flood
1.01	H1	`http`	S0	...	**72**	SYN_flood
1.02	H1	`http`	S0	...	**75**	SYN_flood
...	...	...	...	...	...	...
100.1	H3	`http`	S0	...	**1**	normal
...	...	...	...	...	...	...
351.2	H5	`http`	S0	...	**3**	normal
...	...	...	...	...	...	...
632.4	H1	`http`	S0	...	**4**	normal

Unlike many data mining applications, the biggest challenge here is not the development of classification algorithms, but rather feature extraction and construction.

Audit data is "raw", i.e., in binary format, unstructured, and time dependent. For data mining, we need to first process audit data to a suitable form, i.e., ASCII tabular data with attributes (or features). We can borrow standard techniques from other applications, e.g., network modeling and analysis, for data preprocessing. However, these basic and standard features are insufficient for the purpose of intrusion detection. For example, consider the "SYN flood" intrusion (a Denial-of-Service attack) where the attacker sends a lot of connection requests within a very short period of time but does not complete the handshakes, so that the destination host is forced to keep the requests and use up its buffer space, and as a result, can no longer service other legitimate requests. The standard features of the connection records in Table 1, i.e., timestamp, destination host, service, flag, etc. cannot distinguish the "SYN flood" connections from the normal ones. A S0 flag means that there is a connection request but the handshake is not completed. A normal (or legitimate) connection can have a S0 flag because handshake packets are sometimes dropped due to temporary network congestion. However, for connections to the same service on the same host within a short period of time, if a large percentage of them have the S0 flag, then there is likely a "SYN flood" attack. Accordingly, we need a feature, %S0, in the connection data in order to distinguish "SYN flood" connections from normal ones. The research issue is how to automatically construct such features.

We have developed a data mining framework for constructing features and intrusion detection models [10]. Using this framework, raw (binary) audit data is first processed and summarized into network connection records (or host session records) containing a number of basic features: timestamp, duration, source and destination IP addresses and port numbers, protocol type, and an error condition flag. Specialized data mining programs [12; 10] are applied to compute frequent patterns, i.e., the association rules [1] describing the correlations among the features, and the frequent episodes [15] describing the frequently co-occurring events across the records. The consistent patterns of normal activities and the "unique" patterns associated with intrusions are then identified and analyzed to construct additional features for the records [13]. Machine learning algorithms (e.g., the RIPPER [4] classification rule learner) are

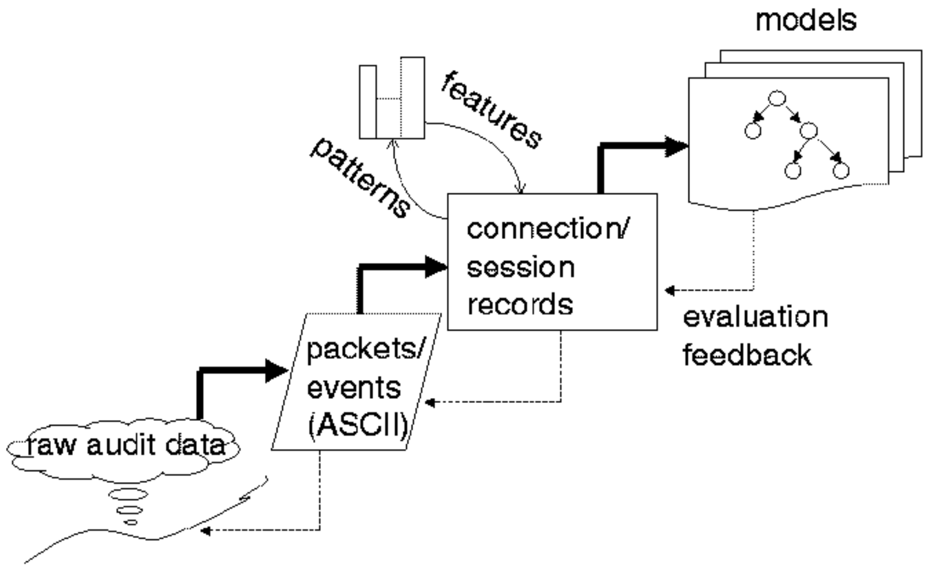


Figure 1: The Data Mining Process of Building Intrusion Detection Models.

then used to learn the detection models. Figure 1 depicts the iterative process steps of this framework. We next give an overview of the main components of this framework.

2.1 Pattern Mining and Comparison

We compute the association rules and frequent episodes from audit data, which capture the intra- and inter- audit record patterns. These frequent patterns can be regarded as the statistical summaries of network and system activities captured in the audit data, because they measure the correlations among system features and the sequential (i.e., temporal) co-occurrences of events.

The basic association rules and frequent episodes algorithms do not consider any domain knowledge. That is, assume I is the interestingness measure of a pattern p, then $I(p) = f(\mathit{support}(p), \mathit{confidence}(p))$, where f is some ranking function. As a result, the basic algorithms can generate many rules that are "irrelevant" (i.e., uninteresting) to the application. When customizing these algorithms for audit data, we incorporate *schema-level* knowledge into the interestingness measures. Assume I_S measures the interestingness of a pattern p according to the schema-level constraints, our extended interestingness measure is $I_e(p) = f_e(I_S(p), I(p))$, where f_e is a ranking function that first considers the schema-level constraints, then the support and confidence values.

We discuss two kinds of important schema-level knowledge about audit data here. First, there is a partial "order of importance" among the attributes of an audit record. Some attributes are *essential* in describing the data, while others only provide auxiliary information. For example, a network connection can be uniquely identified by the combination of its start time, source host, source port, destination host, and service (destination port). These are the essential attributes when describing network data. We call the essential attribute(s) *axis* attribute(s) when they are used as a form of *item constraints* in the association rules algorithm. During candidate generation, an item set must contain value(s) of the axis attribute(s). In other words, if p contains axis attribute(s), then $I_S(p) = 1$, else $I_S(p) = 0$. To avoid having a huge amount of "useless" episode rules, we extended the basic frequent episodes algorithm to compute frequent sequential patterns in two phases: compute the frequent associations using the axis attribute(s); then generate the frequent serial patterns from these associations.

Another interesting schema-level information is that some attributes can be the *references* of other attributes. A group of events are related if they have the same reference attribute value. For example, connections to the same destination host can be related. When mining for patterns of such related events, we need to use *reference attribute* as an item constraint. That is, when forming an episode, an additional condition is that, within its minimal occurrences, the records covered by its constituent itemsets have the same value(s) of the reference attribute(s). In other words, if the itemsets of p refer to the same reference attribute value, then $I_S(p) = 1$, else $I_S(p) = 0$.

We can compare the patterns from an intrusion dataset and the patterns from the normal dataset to identify those that exhibit only in the intrusion dataset. These patterns are then used for feature construction. The details of the pattern comparison algorithm are described in [13]. 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. 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

Table 2: Example Intrusion Pattern

Frequent Episode	Meaning
$(flag = S0, service = http, dst\_host = H1), (flag = S0, service = http, dst\_host = H1) \rightarrow (flag = S0, service = http, dst\_host = H1)$ [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

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, for the "SYN flood" attack, Table 2 shows one of the top intrusion only patterns, produced using *service* as the axis feature and *dst_host* as the reference feature.

2.2 Feature Construction

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*) 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*, or *dst* other than F_0 (i.e., F_1 is an essential feature). If the same F_1 value 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 numerical, 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

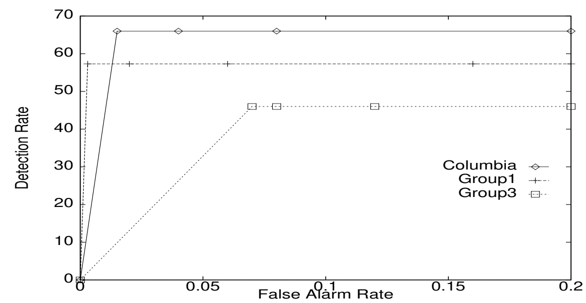


Figure 2: The Overall Detection Performance

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. The "SYN flood" pattern shown in Table 2 results in the following additional features: a count of connections to the same *dst* 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*.

2.3 Evaluation and Discussion

We participated in the official 1998 DARPA Intrusion Detection Evaluation [14]. We applied our data mining framework to construct features and intrusion detection models using the 7 weeks of labeled training data. We then used the models 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). We briefly report the performance of our detection models as evaluated by MIT Lincoln Lab [14] (see [11] for more detailed results). Figure 2 shows the ROC curves of the detection models on all intrusions. We compare here our models with other participants (denoted as Group 1 through 3, group 2 did not cover all intrusions) in the DARPA evaluation program[1]. These participating groups used pure knowledge engineering approaches. We can see from the figure that our detection model has the best overall performance, under the "acceptable" false alarm rate (under 0.02%). However, an overall detection rate of below 70% is hardly satisfactory in a mission critical environment.

There are still many open issues in this research. The most important and challenging one is a general approach for constructing anomaly detection models (which are the only means to detect new intrusions). Unlike the problem of

[1]The tested systems produced binary output, hence, the ROC's are not continuous. In fact, they should just be data points, one for each group. Lines are connected for display and comparison purposes.

building misuse detection models, where the focus is on finding a few intrusion-specific patterns and features, constructing anomaly detection models entails finding comprehensive sets of patterns and features often without the help of example intrusion data. Approaches thus far are all developed for specific problems (e.g., modeling program execution). Another issue is that our data mining algorithms compute only the frequent patterns of connection records. Many intrusions, e.g., those that embed all activities within a single connection, do not have frequent patterns in connection data. Some of these intrusions have frequent patterns in packet data. However, there is no fixed format of packet data contents, and hence we cannot use our (attribute-based) data mining programs. *Free text mining* algorithms are needed for packet data. Still, some of these intrusions involve only a single event (e.g., one command), and hence leave no frequent patterns even in packet data. Thus, we need algorithms capable of mining *rare and unexpected* patterns for these intrusions. An important issue is the usability of our data mining programs. Without system support, the iterative process shown in Figure 1 is slow and labor-intensive even when each step is supported by data processing and mining programs. We have developed a system, "an analyst's workbench", that semi-automates this process by chaining these steps and integrating the relevant tools and algorithms into a single system [10; 11]. Using the patterns, features, and detection model computed by the system, a developer(s) can gain an understanding of the new attack, and can "drive" the system, for example, adjust parameters for a new iteration, to produce an effective detection model. We need to investigate how to use *sampling* (e.g., [17]) to reduce the size of input data, while maintaining its temporal and statistical characteristic, so that the modeling process can be more efficient. We also need to develop *heuristics* for iterations so that an optimal model can be produced without exhaustively searching through the parameter spaces.

3. THE QUEST FOR EFFICIENCY

Auditing mechanisms are designed to record *all* network and system activities in great details. While this ensures that no intrusion evidence will be missed, the high-speed and high-volume data stream requires the run-time execution of detection models be very efficient. Otherwise, the long delay in data analysis simply presents a time window for attacks to succeed. The challenge for data mining is to develop techniques to compute detection models that are not only accurate but also *efficient in run-time execution*. The efficiency of a detection model is measured by its computational cost, which is derived mainly from the time cost of computing the required features. The feature cost includes not only the time required for computing its value but also the time delay of its readiness (i.e., when it can be computed). We describe two approaches for building efficient models below.

3.1 A Multiple Model Approach

Fan et al. developed a multiple model approach [5]. The main idea is to compute a few models with increasing computation costs and higher accuracy. In run-time, the low cost models are always used first, and high cost models are used only when the low cost models cannot predict with sufficient accuracy. Using network connection data as an example, we partition features into three relative cost levels. Level 1 features, e.g., *service*, are computed using at most the first three packets (or events) of a connection (or host session). They normally require only simple recording. Level 2 features are computed in the middle or near the end of a connection using information of the current connection only. They usually require just simple book keeping. Level 3 features are computed using information from all connections within a given time window of the current connection. They are often computed as some aggregates of the level 1 and 2 features. We assign *qualitative* cost values based on empirical studies: level 1 cost is 1 or 5; level 2 cost is 10; and level 3 cost is 100.

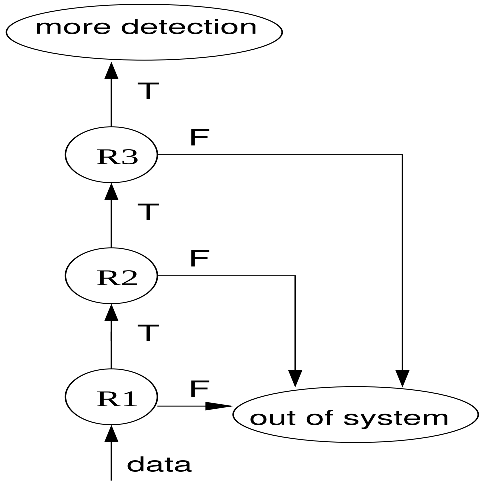
In the training phase, we use multiple training sets $T_1, \ldots, T_4$ with different feature subsets. T_1 uses only cost 1 features. T_2 uses features of costs 1 and 5, and so forth, up to T_4, which uses all available features. Rulesets $R_1, \ldots, R_4$ are learned using their respective training sets. The precision p_r is computed for every rule, r, except for the rules in R_4. A threshold value τ_c is obtained for every class. It is the precision of the corresponding rule in R_4 because we want the predictions made by other rulesets be as good as R_4. In real-time execution, R_i is evaluated first and a classification c is made. If $p_r \geq \tau_c$, the prediction c is final. Otherwise, additional features required by R_{i+1} are computed and R_{i+1} is evaluated. This process continues until a final prediction is made.

We evaluate this approach using the 1998 DARPA dataset. Our results [5] show that compared with a single model approach (that uses whatever features are needed regardless of the costs), the multiple model approach achieves a 95% reduction in terms of computational cost while maintaining a very similar accuracy.

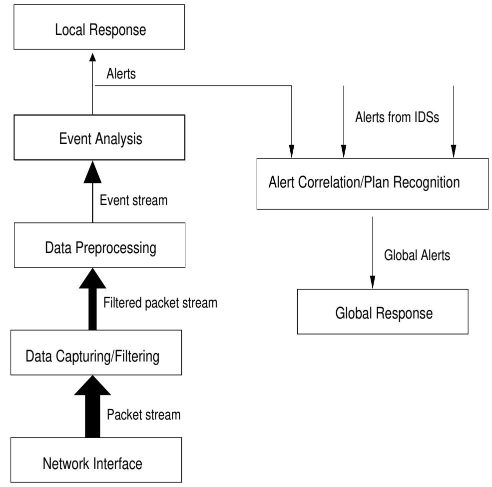
3.2 Cascaded Detection Modules

In a recent study in real-time face detection, an application domain that bears similarities to intrusion detection, Viola and Jones proposed to use cascaded modules to speed up the detection process [21]. The main idea is to use successively more complex detection modules where simpler modules can quickly filter out obvious non-objects (e.g., background regions) so that the subsequent and more complex modules can focus on the more promising regions of the image. We can use this idea to build cascaded intrusion detection modules, as shown in Figure 3(a). As data enters the cascade and is analyzed by a detection module, only the portion determined as likely intrusive, or simply, interesting, is passed for further analysis at the next layer (by a more complex module). This approach can lead to much improved run-time efficiency because only a tiny portion of the audit data, i.e., the likely intrusion data, needs to be processed by the compute-intensive (final) detection module. Unlike our multiple model approach, here the simpler modules are not required to make predictions on the intrusions. They only need to filter out the normal data.

The detection modules are *independent* and usually use different detection algorithms and features. The detection rate of a cascade of N detection modules is the product of the detection rates of the individual modules, i.e., $\prod_{i=1}^{N} P_i(A|I)$. Likewise, the false alarm rate is $\prod_{i=1}^{N} P_i(A|\neg I)$. It is obvious that each detection module needs to have a very high detection rate. For example, if there are a total of five modules and each has a detection rate of 99%, then the overall detection rate is just about 95%. In general, a detector with a high detection rate will also have a high false alarm rate.



(a) Cascaded Detection Modules



(b) Layered IDS Architecture

Figure 3: (a) In a cascade of detection modules, only the "likely" intrusion data is passed from one (simpler) module to the next (more complex) module. (b) In a layered IDS, data is processed at the appropriate semantic levels.

However, when the detectors are cascaded together, the resulting overall false alarm rate can be very low. For example, if the five modules each has a false alarm rate of 10%, then the overall false alarm rate is 10^{-5}. In other words, by building each detection module in the cascade to have a very high detection rate at the expense of a high positive rate, which is not an impossible task, we can have a highly efficient intrusion detection model with high detection rate and low false alarm rate.

The key research issue for data mining is how to select the appropriate features for constructing a detection module that is more comprehensive than the one in the previous layer. We are in the process of designing and implementing cascaded intrusion detection modules for network data, and will report our findings in the near future.

4. THE QUEST FOR CREDIBILITY

Let I and $\neg I$ denote the intrusive or non-intrusive (or, normal) behavior, and A and $\neg A$ denote the presence or absence of an intrusion alert from the IDS. The most commonly used IDS performance metrics are the detection rate $P(A|I)$ and false alarm rate $P(A|\neg I)$. From a usability point of view, a critical performance measurement is the Bayesian detection rate [3] $P(I|A)$, i.e., the probability that an intrusion is present when the IDS produces an alert. It measures IDS *credibility* because it indicates how likely an intrusion is present when there is an alert. Using Bayes' theorem, it can be computed as [3]:

$$P(I|A) = \frac{P(I)P(A|I)}{P(I)P(A|I) + P(\neg I)P(A|\neg I)} \tag{1}$$

Although a higher detection rate and lower false alarm rate will lead to a higher Bayesian detection rate, as pointed out by Axelsson, if the base rate (i.e., the probability of intrusion data), P(I), is extremely low, say 2×10^{-5}, then even with a perfect detection rate, $P(A|I) = 1$, and a perhaps unattainably low false alarm rate, say $P(A|\neg I) = 1 \times 10^{-5}$, the Bayesian detection rate is only 66% [3]. Axelsson concluded that there is no way for an IDS to have an acceptable Bayesian detection rate because in a typical network the frequency of intrusion data (i.e., the base rate) in the huge amount of (raw) audit data is extremely low. It is also obvious that applying a new detection algorithm or a new model generation approach (e.g., data mining) alone is not likely to solve this problem, unless one can guarantee that $P(A|\neg I) = 0$.

The key to improve the Bayesian detection rate is to increase the base rate of the data stream analyzed by the complex detection module(s). This requires a layered IDS architecture as shown in Figure 3(b). Here, data processing and analysis tasks are carried out at the appropriate semantic levels where the base rates are much higher than in the raw audit data. At the lowest level, the network interface receives network traffic data. The data capturing and filtering unit then selects only a portion of the traffic according to the IDS configuration policy. The packet data is then pre-processed (e.g., re-assembled to connection data), and only the important events (e.g., a new connection is established) or suspicious events (e.g., an attempted connection to a closed port) are extracted. The event analysis engine then uses an intrusion detection algorithm(s) to piece the event data together and produces an alert when it believes that an intrusion is occurring. Alerts can trigger local response, and can be sent to a global correlator for cross-checking with other IDSs, and for analysis of distributed attacks and long-term attack plans (or scenarios).

The layered IDS architecture presents opportunities for data mining. For example, a research issue is developing *clustering* techniques for alert correlation or reduction. An IDS can generate a large number of alerts for a single attack. Multiple IDSs on the network (if so configured) can all report the same attack. Alert correlation can reduce the number of intrusion reports that an security staff needs to handle, thus improving the usability and credibility of the IDS. Another research problem is developing techniques similar to *link analysis* for analyzing attack plan, which is a sequence of related attacks that together accomplish an end-goal. Grouping the individual attacks into attack plans helps the security staff gain a better understanding of the network state and take proactive response actions, while significantly lowering their workload.

5. SUMMARY

Intrusion detection is a real-world application critical to the well-being of our society. In this paper, we have given an overview of our research in applying data mining techniques to intrusion detection. We have described a framework for mining patterns from audit data and constructing features based on analysis of intrusion patterns. We have discussed approaches for improving the run-time efficiency of detection models. We have also outlined techniques for improving IDS credibility.

We have pointed out throughout the paper that there are still many open problems and research opportunities. We as researchers must take up these challenges, and make contributions to both data mining and intrusion detection.

6. ACKNOWLEDGMENTS

This research has been supported in part by DARPA (F30602-96-1-0311 and F30602-00-1-0603) and NSF (CCR-0133629). Many thanks to Sal Stolfo and members of his Columbia research team for all the guidance and helps. We also wish to thank all the *open-minded* researchers in security and data mining who gave us encouragements and good suggestions at the early stage of our research.

7. REFERENCES

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

[2] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (NIDES): A summary. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, California, May 1995.

[3] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(3), 2000.

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

[5] Wei Fan, Wenke Lee, Sal Stolfo, and Matt Miller. A multiple model cost-sensitive approach for intrusion detection. In *Proceedings of The Eleventh European Conference on Machine Learning (ECML 2000), Lecture Notes in Artificial Intelligence No. 1810*, Barcelona, Spain, May 2000.

[6] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

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

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

[9] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, August 1999.

[10] W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, June 1999.

[11] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.

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

[13] W. Lee, S. J. Stolfo, and K. W. Mok. 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.

[14] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunninghan, and M. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, January 2000.

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

[16] S. McCanne, C. Leres, and V. Jacobson. `libpcap`. available via anonymous ftp to ftp.ee.lbl.gov, 1994.

[17] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. AAAI Press, August 1999.

[18] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., 2000.

[19] S.J. Stolfo, W. Lee, P.K. Chan, W. Fan, and E. Eskin. Data mining-based intrusion detectors: An overview of the Columbia IDS project. *ACM SIGMOD Record*, 30(4), December 2001.

[20] SunSoft. *SunSHIELD Basic Security Module Guide*. SunSoft, Mountain View, CA, 1995.

[21] P. Viola and M. Jones. Robust real-time object detection. In *Proceedings of the Second International Workshop on Statistical and Computational Theories of Vison - Modeling, Learning, Computing, and Sampling*, May 2002.

[22] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.