Intrusion Detection Techniques

Peng Ning, North Carolina State University
Sushil Jajodia, George Mason University

In an information system, *intrusions* are the activities that violate the security policy of the system, and *intrusion detection* is the process used to identify intrusions. Intrusion detection techniques have been traditionally classified into one of two methodologies: *anomaly detection* or *misuse detection*. This chapter gives an overview of the existing intrusion detection techniques, including anomaly detection and misuse detection models, and identifies techniques related to intrusion detection in distributed systems. Topics covered include statistical models, machine learning and data mining approaches, computer immunological approach, specification-based approach, information theoretic measures for anomaly detection, rule-based languages, state transition analysis toolkit, colored Petri automata (CPA), abstraction-based approach, distributed intrusion detection systems, network-based intrusion detection systems, and information sharing among intrusion detection systems.

INTRODUCTION

Intuitively, *intrusions* in an information system are the activities that violate the security policy of the system, and *intrusion detection* is the process used to identify intrusions. Intrusion detection has been studied for approximately 20 years. It is based on the beliefs that an intruder's behavior will be noticeably different from that of a legitimate user and that many unauthorized actions will be detectable.

Intrusion detection systems (IDSs) are usually deployed along with other preventive security mechanisms, such as access control and authentication, as a second line of defense that protects information systems. There are several reasons that make intrusion detection a necessary part of the entire defense system. First, many traditional systems and applications were developed without security in mind. In other cases, systems and applications were developed to work in a different environment and may become vulnerable when deployed in the current environment. (For example, a system may be perfectly secure when it is isolated but become vulnerable when it is connected to the Internet.) Intrusion detection provides a way to identify and thus allow responses to, attacks against these systems. Second, due to the limitations of information security and software engineering practice, computer systems and applications may have design flaws or bugs that could be used by an intruder to attack the systems or applications. As a result, certain preventive mechanisms (e.g., firewalls) may not be as effective as expected.

Intrusion detection complements these protective mechanisms to improve the system security. Moreover, even if the preventive security mechanisms can protect information systems successfully, it is still desirable to know what intrusions have happened or are happening, so that we can understand the security threats and risks and thus be better prepared for future attacks.

In spite of their importance, IDSs are not replacements for preventive security mechanisms, such as access control and authentication. Indeed, IDSs themselves cannot provide sufficient protection for information systems. As an extreme example, if an attacker erases all the data in an information system, detecting the attacks cannot reduce the damage at all. Thus, IDSs should be deployed along with other preventive security mechanisms as a part of a comprehensive defense system.

Intrusion detection techniques are traditionally categorized into two methodologies: *anomaly detection* and *misuse detection*. Anomaly detection is based on the normal behavior of a subject (e.g., a user or a system); any action that significantly deviates from the normal behavior is considered intrusive. Misuse detection catches intrusions in terms of the characteristics of known attacks or system vulnerabilities; any action that conforms to the pattern of a known attack or vulnerability is considered intrusive.

Alternatively, IDSs may be classified into host-based IDSs, distributed IDSs, and network-based IDSs according to the sources of the audit information used by each IDS. Host-based IDSs get audit data from host audit trails and usually aim at detecting attacks against a single host; distributed IDSs gather audit data from multiple hosts and possibly the network that connects the hosts, aiming at detecting attacks involving multiple hosts. Network-based IDSs use network traffic as the audit data source, relieving the burden on the hosts that usually provide normal computing services.

This chapter starts with an overview of current intrusion detection techniques. Next, it reviews the various types of anomaly detection methods, such as statistical models and machine learning methods, followed by an overview of various types of misuse detection methods, including rule-based languages, the colored Petri-net-based method, and the abstraction-based method. The section following that discusses additional techniques for intrusion detection in distributed systems, including distributed IDSs, network-based IDSs, and interoperation between (heterogeneous) IDSs.

ANOMALY DETECTION

Statistical Models

Statistical modeling is among the earliest methods used for detecting intrusions in electronic information systems. It is assumed that an intruder's behavior is noticeably different from that of a normal user, and statistical models are used to aggregate the user's behavior and distinguish an attacker from a normal user. The techniques are applicable to other subjects, such as user groups and programs. Here, we discuss two statistical models that have been proposed for anomaly detection: NIDES/STAT and Haystack.

NIDES/STAT

The Stanford Research Institute's next-generation real-time intrusion detection expert system statistical component (NIDES/STAT) observes behaviors of subjects on a monitored computer system and adaptively learns what is normal for individual subjects, such as users and groups (Axelsson, 1999). The observed behavior of a subject is flagged as a potential intrusion if it deviates significantly from the subject's expected behavior.

The expected behavior of a subject is stored in the profile of the subject. Different measures are used to measure different aspects of a subject's behavior. When audit records are processed, the system periodically generates an overall statistic, T^2, that reflects the abnormality of the subject. This value is a function of the abnormality values of all the measures comprising the profile. Suppose that n measures M_1, M_2, ..., M_n are used to model a subject's behavior. If S_1, S_2, …, S_n represent the abnormality values of M_1 through M_n, then the overall statistic T^2 is evaluated as follows, assuming that the n measures are independent of each other:

$$T^2 = S_1^{\,2} + S_2^{\,2} + \ldots + S_n^{\,2}.$$

The profile of a subject is updated to reflect the changes of the subject's behavior. To have the most recently observed behaviors influence the profile more strongly, NIDES/STAT multiplies the frequency table in each profile by an exponential decay factor before incorporating the new audit data. Thus, NIDES/STAT adaptively learns a subject's behavior patterns. This keeps human users from having to manually adjust the profiles; however, it also introduces the possibility of an attacker gradually "training" the profile to consider his/her intrusive activities as normal behavior.

Haystack

Haystack used a different statistical anomaly detection algorithm, which was adopted as the core of the host monitor in the distributed intrusion detection system (DIDS) (Axelsson, 1999). This algorithm analyzes a user's activities according to a four-step process.

First, the algorithm generates a session vector to represent the activities of the user for a particular session. The session vector **X** = <x_1, x_2, …, x_n > represents the counts for various attributes used to represent a user's activities for a single session. Examples of the attributes include session duration and number of files opened for reading.

Second, the algorithm generates a Bernoulli vector to represent the attributes that are out of range for a particular session. A threshold vector **T** = <t_1, t_2, …, t_n>, where t_i is a tuple of the form <$t_{i,\,\min}$, $t_{i,\,\max}$>, is used to assist this step. The threshold vector is stored in a user's profile. The Bernoulli vector **B** = <b_1, b_2, ..., b_n> is generated so that b_i is set to 1 if x_i falls outside the range t_i, and b_i is set to 0 otherwise.

Third, the algorithm generates a weighted intrusion score, for a particular intrusion type, from the Bernoulli vector and a weighted intrusion vector. Each group and intrusion type pair has a weighted intrusion vector **W** = <w_1, w_2, …, w_n>, in which each w_i relates the importance of the ith attribute in the Bernoulli vector to detecting the particular intrusion type. The weight intrusion score is simply the sum of all weights, w_i, where the ith attribute falls outside the range t_i. That is,

$$\text{the weighted intrusion score} = \sum_{i=1}^{n} b_i \cdot w_i \,.$$

Finally, the algorithm generates a suspicion quotient to represent how suspicious this session is compared with all other sessions for a particular intrusion type. Specifically, the suspicion quotient is the probability that a random session's weighted intrusion score is less than or equal to the weighted intrusion score computed in the previous step.

Unlike NIDES/STAT, the Haystack algorithm has a step that determines resemblance to known attacks. The advantages are that more knowledge about the possible attacks can be derived from this step and better responses can follow the alarms. However, extra knowledge about possible intrusion types is required: We need to understand the impact of the intrusion types on the attributes of the session vectors and assign appropriate weights to these attributes to reflect the impact. In reality, the process of generating the weighted intrusion vectors is time consuming and error prone.

Machine Learning and Data Mining Techniques

Time-Based Inductive Machine

Teng, Chen, and Lu (1990) proposed the use of a time-based inductive machine (TIM) to capture a user's behavior pattern. As a general-purpose tool, TIM discovers temporal sequential patterns in a sequence of events. The sequential patterns represent highly repetitive activities and are expected to provide predication. The temporal patterns, which are represented in the form of rules, are generated and modified from the input data using a logical inference called *inductive generalization*. When applied to intrusion detection, the rules describe the behavior patterns of either a user or a group of users based on past audit history. Each rule describes a sequential event pattern that predicts the next event from a given sequence of events. An example of a simplified rule produced in TIM is

$$E1 - E2 - E3 \rightarrow (E4 = 95\%;\ E5 = 5\%),$$

where $E1$, $E2$, $E3$, $E4$, and $E5$ are security events.

This rule says that if $E1$ is followed by $E2$, and $E2$ is followed by $E3$, then there is a 95% chance (based on the previous observation) that $E4$ will follow, and a 5% chance that $E5$ will follow. TIM can produce more generalized rules than the above. For example, it may produce a rule in the form

$$E1 - * \rightarrow (E2 = 100\%),$$

where an asterisk matches any single event. Any number of asterisks is allowed in a rule.

The limitation of TIM is that it only considers the *immediately following* relationship between the observed events. That is, the rules only represent the event patterns in which events are adjacent to each other. However, a user may perform multiple tasks at the same time. For example, a user may check his/her e-mail during the editing of a document. The events involved in one application, which tend to have strong patterns embedded in the sequence of events, may be interleaved with events from other applications. As a result, it is very possible that the rules generated by TIM cannot precisely capture the user's behavior pattern. Nevertheless, TIM may be suitable for capturing the behavior patterns of such entities as programs that usually focus on single tasks.

Instance Based Learning

Lane and Brodley (1998) applied *instance based learning* (IBL) to learn entities' (e.g., users) normal behavior from temporal sequence data. IBL represents a concept of interest with a set of instances that exemplify the concept. The set of instances is called the instance dictionary. A new instance is classified according to its relation to stored instances. IBL requires a notion of "distance" between the instances so that the similarity of different instances can be measured and used to classify the instances.

Lane and Brodley did several things to adapt IBL to anomaly detection. First, they transformed the observed sequential data into fixed-length vectors (called *feature vectors*). Specifically, they segmented a sequence of events (e.g., a sequence of user commands) into all possible overlapping sequences of length l, where l is an empirical parameter. (Thus, each event is considered the starting point of a feature vector, and each event is replicated l times.) Second, they defined a similarity measure between the feature vectors. For a length l, the similarity between feature vectors $\mathbf{X} = (x_0, x_1, \ldots, x_{l-1})$ and $\mathbf{Y} = (y_0, y_1, \ldots, y_{l-1})$ is defined by the functions

$$w(X,Y,i) = \begin{cases} 0, & \text{if } i < 0 \text{ or } x_i \neq y_i \\ 1 + w(X,Y,i-1), & \textit{if } x_i = y_i \end{cases}$$

and

$$Sim(X,Y) = \sum_{i=0}^{l-1} w(X,Y,i).$$

The converse measure, distance, is defined as $\text{Dist}(\mathbf{X},\mathbf{Y}) = \text{Sim}_{\max} - \text{Sim}(\mathbf{X},\mathbf{Y})$, where $\text{Sim}_{\max} = \text{Sim}(\mathbf{X},\mathbf{X})$. Intuitively, the function $w(\mathbf{X}, \mathbf{Y}, i)$ accumulates weights from the most recently consecutively matched subsequences between $\mathbf{X}$ and $\mathbf{Y}$ at position i, whereas $\text{Sim}(\mathbf{X},\mathbf{Y})$ is the integral of total weights.

A user profile is built to contain a collection of sequences, $\mathbf{D}$, selected from a user's observed actions (e.g., commands). The similarity between the profile and a newly observed sequence, $\mathbf{X}$, is defined as $\text{Sim}_{\mathbf{D}}(\mathbf{X}) = \max_{\mathbf{Y}\in\mathbf{D}}\{\text{Sim}(\mathbf{Y}, \mathbf{X})\}$. That is, the similarity between $\mathbf{X}$ and $\mathbf{D}$ is defined as the similarity between $\mathbf{X}$ and a vector in $\mathbf{D}$ that is most similar to $\mathbf{X}$. Then a threshold r is chosen. If the similarity between an observed sequence $\mathbf{X}$ and the profile $\mathbf{D}$ is greater than r, $\mathbf{X}$ is considered normal; otherwise, $\mathbf{X}$ is abnormal.

To reduce the storage required by the profile, Lane and Brodley used the least-recently-used pruning strategy to keep the profile at a manageable size. As new instances are acquired and classification is performed, the profile instance selected as most similar is time stamped. Least-recently-used instances are removed when the profile is constrained to the desired size. In addition, they applied a clustering technique to group the instances in the profile, and they used a representative instance for each cluster.

This attempt shares a problem similar to that of TIM, that is, it tries to find patterns from sequences of consecutive events. As the authors have noted, a user may interrupt his/her normal work (e.g., programming) and do something different (e.g., answer an urgent e-mail) and thus yield a different sequence of actions from his/her profile. Their solution (Lane & Brodley, 1998) is to use a time average of the similarity signals; however, such a solution may make real anomalies unnoticeable. In addition, the least-recently-used pruning strategy gives an attacker a chance to train the profile slowly, so that intrusive activities are considered normal ones.

Neural Network

Fox, Henning, Reed, and Simmonian (1990) were the first to attempt modeling system and user behaviors using neural networks. Their choice of neural network is Kohonen's *self-organizing map* (SOM), which is a type of unsupervised learning technique that can discover underlying structures of the data without prior examples of intrusive and nonintrusive activities.

They used SOM as a real-time background monitor that alerts a more complex expert system. In their prototype system, 11 system parameters accessible from the system's statistical performance data are identified as the input to the SOM model. These parameters include (a) central processing unit (CPU) usage, (b) paging activity, (c) mailer activity, (d) disk accesses, (e) memory usage, (f) average session time, (g) number of users, (h) absentee jobs, (i) reads of "help" files, (j) failed log-ins, and (k) multiple log-ins. However, their study showed the results of only one simulated virus attack, which are not sufficient to draw serious conclusions.

In another attempt to apply neural network to anomaly detection, Ghosh, Wanken, and Charron (1998) proposed using a *back-propagation* network to monitor running programs. A back-propagation network is developed for supervised learning. That is, it needs examples of intrusive and nonintrusive activities (called *training data*) to build the intrusion detection model. Such a network consists of an input layer, at least one hidden layer (neurons that are not directly connected to the input or output nodes), and an output layer. Typically, there are no connections between neurons in the same layer or between those in one layer and those in a previous layer.

The training cycle of a back-propagation network works in two phases. In the first phase, the input is submitted to the network and propagated to the output through the network. In the second phase, the desired output is compared with the network's output. If the vectors do not agree, the network updates the weights starting at the output neurons. Then the changes in weights are calculated for the previous layer and cascade through the layers of neurons toward the input neurons.

Ghosh et al. proposed using program input and the program internal state as the input to the back-propagation network. One interesting result is that they improved the performance of detection by using randomly generated data as anomalous input. By considering randomly generated data as anomalous, the network gets more training data that is complementary to the actual training data.

Similar to statistical anomaly detection models, deciding the input parameters for neural network anomaly detectors is a difficult problem. In addition, assigning the initial weights to the neural networks is also an unresolved question. The experiments by Ghosh et al. (1998) showed that different initial weights could lead to anomaly detectors with different performance. Nevertheless, research on applying neural networks to anomaly detection is still preliminary; more work is needed to explore the capability of neural networks.

Audit Data Analysis and Mining

Audit data analysis and mining (ADAM) proposes applying data mining techniques to discover abnormal patterns in large amounts of audit data (e.g., network traffic collected by TCPdump, which is a program used to sniff and store packets transmitted in the network) (Barbara, Couto, Jajodia, & Wu, 2001; Barbara, Couto;, Jajodia, & Wu, 2002). In particular, the existing research focuses on the analysis of network audit data, such as the transmission control protocol (TCP) connections. Using data mining techniques, ADAM has the potential to provide a flexible representation of the network traffic pattern, uncover some unknown patterns of attacks that cannot be detected by other techniques, and accommodate the large amount of network audit data that keeps growing in size.

ADAM uses several data-mining-related techniques to help detect abnormal network activities. The first technique ADAM uses is inspired by association rules. Given a set I of items, an association rule is a rule of the form $X \rightarrow Y$, where X and Y are subsets (called item sets) of I and $X \cap Y = \phi$. Association rules are usually discovered from a set T of transactions, where each transaction is a subset of I. The rule $X \rightarrow Y$ has a *support s* in the transaction set T if s% of the transactions in T contain $X \cup Y$, and it has a *confidence c* if c% of the transactions in T that contain X also contain Y.

However, ADAM doesn't use association rules directly; instead, it adopts the item sets that have large enough support (called *large item sets*) to represent the pattern of network traffic. Specifically, it assumes that each network event (e.g., a TCP connection) is described by a set of attribute values and considers each event a transaction. The large item sets discovered from the network traffic then represent the frequent events in the network traffic. The power of such a mechanism lies in the flexible representation of events.

ADAM builds a profile of normal network activities in which the frequent events (represented by large item sets) are stored. During the detection time, it adopts a sliding-window method to incrementally examine the network events. Within each window, ADAM looks for the large item sets that do not appear in the profile and considers them suspicious.

The second technique ADAM uses is called *domain-level mining*. Intuitively, it tries to generalize the event attribute values used to describe a network event. For example, an IP address that belongs to the subnet *ise.gmu.edu* can be generalized to *ise.gmu.edu*, *gmu.edu*. and *edu*. Then it discovers large item sets using the generalized attribute values. An advantage of this approach is that it provides a way to aggregate the events that share some commonality and may discover more attacks. However, the scheme used to generalize the attribute values is ad hoc; only generalization from IP addresses to subnets and from smaller subnets to larger subnets are studied.

The third technique ADAM uses is classification. ADAM is innovative in that classification is used to classify the output of the mining of large item sets. Four classification algorithms have been studied to date: C4.5 decision tree, naive Bayes,cascading classifier (which uses decision tree followed by naive Bayes and vice versa), and inductive rule learner. The results show that classification is quite effective in reducing false alarms.

Finally, ADAM uses the pseudo-Bayes estimator to accommodate unknown attacks. It is assumed that unknown attacks are the attacks that have not been observed. The training data is represented as a set of vectors, each of which corresponds to an event and is labeled as normal or as a known class of attacks. An additional class is then considered to represent the unknown attacks. Because the unknown attacks haven't been observed in the training data, the probability $P(\mathbf{x}|\text{class = unknown})$, where $\mathbf{x}$ is a training vector, is zero. The pseudo-Bayes estimator is used to smooth all the conditional probabilities $P(\mathbf{x}|\text{class})$ so that $P(\mathbf{x}|\text{class = unknown})$ is assigned a (small) probability. These conditional probabilities are then used to build a naive Bayes classifier.

The limitation of ADAM is that it cannot detect stealthy attacks. In other words, it can detect an attack only when it involves a relatively large number of events during a short period of time. This limitation occurs because ADAM raises an alarm only when the support of an unexpected rule (i.e., association of event attributes) exceeds a threshold. Indeed, this limitation is not unique to ADAM; most of the anomaly detection models suffer from the same problem.

Computer Immunological Approach

The computer immunological approach is based on an analogy of the immune system's capability of distinguishing self from non-self (Hofmeyr, Forrest, & Somayaji, 1998). This approach represents self as a collection of strings of length l, where l is a system-wide parameter. A string of length l is considered non-self if it does not match any string belonging to self. To generate detectors that can distinguish non-self from self, a naive approach is to randomly generate a string of length l and check whether it matches any self-string. If yes, the generated string is discarded; otherwise, it is used as a detector. However, the naive approach takes time exponential to the number of self-strings. To address this problem, Forrest et al. proposed a "r-contiguous-bits" matching rule to distinguish self from non-self: two l-bit strings match each other if they are identical in at least r contiguous positions. As a result, detectors can be generated more efficiently for this particular matching rule.

Hofmeyr et al. proposed using short sequences of system calls to distinguish self from non-self for anomaly detection. Given a program in a particular installation, the immunological approach collects a database of all unique system call sequences of a certain length made by the program over a period of normal operation. During the detection time, it monitors the system call sequences and compares them with the sequences in the aforementioned database. For an observed sequence of system calls, this approach extracts all subsequences of length l and computes the distance $d_{\min}(i)$ between each subsequence i and the normal database as $d_{\min}(i) = \min\{d(i,j)$ for all sequences j in the normal database$\}$, where $d(i,j)$ is the Hamming distance between sequences i and j (i.e., the number of different bits in sequences i and j). The anomaly score of the observed sequence of system calls is then the maximum $d_{\min}(i)$ normalized by dividing the length of the sequence. This approach raises an alarm if the anomaly score is above a certain threshold.

The advantage of this approach is that it has a high probability of detecting anomalies using a small set of self-strings based on short sequences of system calls. In addition, it does not require any prior knowledge about attacks. The disadvantage is that it requires self to be well understood. That is, it requires a complete set of self-strings in order not to mistake self for non-self. This requirement may be trivial for such applications as virus detection, but it is very difficult for intrusion detection, where some of normal behaviors cannot be foreseen when the detectors are being generated.

Sekar, Bendre, Dhurjati, and Bollineni (2001) further improve the computer immunological method by using an automaton to represent a program's normal behavior.The program counter is used as the state of the automaton, and the system calls made by the program are used as the events that cause the state transitions. As a result, the automaton representation can accommodate more information about the programs' normal behavior and, thus, reduce false alert rate and improve detection rate. In addition, the automaton representation is more compact than the previous alternative. Consequently, such automata are easier to build and more efficient to use for intrusion detection.

Specification-Based Methods

Ko, Ruschitzka, & K. Levitt (1997) proposed a specification-based approach for intrusion detection. The idea is to use *traces*, ordered sequences of execution events, to specify the intended behaviors of concurrent programs in distributed system. A specification describes valid operation sequences of the execution of one or more programs, collectively called a (*monitored*) *subject*. A sequence of operations performed by the subject that does not conform to the specification is considered a security violation. Each specification is called a *trace policy*. A grammar called *parallel environment grammars* (PE-grammars) was developed for specifying trace policies.

The advantage of this approach is that in theory, it should be able to detect some new types of attacks that intruders will invent in the future. The drawback of this approach is that substantial work is required to specify accurately the behavior of the many privileged system programs, and these specifications will be operating-system specific. To address this issue, Ko (2000) proposed the use of *inductive logic programming* to synthesize specifications from valid traces. The automatically generated specifications may be combined with manual rules to reduce the work involved in specification of valid program behaviors.

Wagner and Dean (2001) further advanced the specification-based approach. The basic idea is to automatically generate the specification of a program by deriving an abstract model of the programs from the source or binary code. Wagner and Dean studied several alternative models, including the call-graph model and the abstract stack model. Central to these models is the control flow graph of a program; these models adopt different ways to represent the possible system call traces according to the control flow graph. Attractive features of this approach are that it has the potential to detect unknown patterns of attacks and it has no false alerts, although it may miss some attacks.

Information-Theoretic Measures

Lee and Xiang (2001) proposed the use of information-theoretic measures to help understand the characteristics of audit data and build anomaly detection models. The well-known concept *entropy* is the first information-theoretic measure. Given a set of classes C_X and a data set X, where each data item belongs to a class $x \in C_X$, the entropy of X relative to C_X is defined as

$$H(X) = \sum_{x \in C_X} P(x) \log \frac{1}{P(x)},$$

where $P(x)$ is the probability of x in X. For anomaly detection, entropy reveals the regularity of audit data with respect to some given classes.

The second information-theoretic measure is *conditional entropy*. The conditional entropy of X given Y is the entropy of the probability distribution $P(x|y)$; that is,

$$H(X \mid Y) = \sum_{x,y \in C_X, C_Y} P(x,y) \log \frac{1}{P(x \mid y)},$$

where $P(x,y)$ is the joint probability of x and y and $P(x|y)$ is the conditional probability of x given y. The conditional entropy is proposed to measure the temporal or sequential characteristics of audit data. Let X be a collection of sequences where each is a sequence of n audit events and where Y is the collection of prefixes of the sequences in X that are of length k. Then $H(X|Y)$ indicates the uncertainty that remains for the rest of the audit events in a sequence x after we have seen the first k events of x. For anomaly detection, conditional entropy can be used as a measure of regularity of sequential dependencies.

The limitation of conditional entropy is that it only measures the sequential regularity of contiguous events. For example, a program may generate highly regular sequences of events; however, if these events are interleaved with other sequences of events, the conditional entropy will be very high, failing to reflect the regularity embedded in the interleaved sequences.

The third information-theoretic measure, *relative entropy*, measures the distance of the regularities between two data sets. The relative entropy between two probability distributions $p(x)$ and $q(x)$ that are defined over the same $x \in C_X$ is

$$\text{relEntropy}(p \mid q) = \sum_{x \in C_X} p(x) \log \frac{p(x)}{q(x)}.$$

The fourth information-theoretic measure, *relative conditional entropy*, measures the distance of the regularities with respect to the sequential dependency between two data sets. The relative entropy between two probability distributions $p(x|y)$ and $q(x|y)$ that are defined over the same $x \in C_X$ and $y \in C_Y$ is

$$\text{relCondEntropy}(p \mid q) = \sum_{x,y \in C_X, C_Y} p(x,y) \log \frac{p(x \mid y)}{q(x \mid y)}.$$

Viewing intrusion detection as a classification problem, Lee and Xiang proposed the fifth information-theoretic measure, *information gain*, to measure the performance of using some features for classification. The information gain of attribute (i.e., feature) A on data set X is

$$\text{Gain}(X, A) = H(X) - \sum_{v \in \text{Values}(A)} \frac{|X_v|}{|X|} H(X_v),$$

where $\text{Values}(A)$ is the set of values of A and X_v is the subset of X where A has value v. This measure can help choose the right features (i.e., the features that have high information gain) to build intrusion detection models. The limitation of information gain is that it requires a relatively complete data set to help choose the right features for the classification model. Nevertheless, an intrusion detection model cannot be better than the data set from which it is built.

Limitation of Anomaly Detection

Although anomaly detection can accommodate unknown patterns of attacks, it also suffers from several drawbacks. A common problem of all anomaly detection approaches, with the exception of the specification-based approach, is that the subject's normal behavior is modeled on the basis of the (audit) data collected over a period of normal operation. If undiscovered intrusive activities occur during this period, they will be considered normal activities. In addition, because a subject's normal behavior usually changes over time (for example, a user's behavior may change when he moves from one project to another), the IDSs that use the above approaches usually allow the subject's profile to gradually change. This gives an intruder the chance to gradually train the IDS and trick it into accepting intrusive activities as normal. Also, because these approaches are all based on summarized information, they are insensitive to stealthy attacks. Finally, because of some technical reasons, the current anomaly detection approaches usually suffer from a high false-alarm rate.

Another difficult problem in building anomaly detection models is how to decide the features to be used as the input of the models (e.g., the statistical models). In the existing models, the input parameters are usually decided by domain experts (e.g., network security experts) in ad hoc ways. It is not guaranteed that all and only the features related to intrusion detection will be selected as input parameters. Although missing important intrusion-related features makes it difficult to distinguish attacks from normal activities, having non-intrusion-related features could introduce "noise" into the models and thus affect the detection performance.

MISUSE DETECTION

Misuse detection is considered complementary to anomaly detection. The rationale is that known attack patterns can be detected more effectively and efficiently by using explicit knowledge of them. Thus, misuse detection systems look for well-defined patterns of known attacks or vulnerabilities; they can catch an intrusive activity even if it is so negligible that the anomaly detection approaches tend to ignore it.

The major problem in misuse detection is how to represent known patterns of attacks. The detection algorithms usually follow directly from the representation mechanisms. In this section, we discuss the typical ways to represent attacks.

Rule-Based Languages

The rule-based expert system is the most widely used approach to misuse detection. The patterns of known attacks are specified as rule sets, and a forward-chaining expert system is usually used to look for signs of intrusions. Here we discuss two rule-based languages, rule-based sequence evaluation language (RUSSEL) (Mounji, Charlier, Zampunieris, & Habris, 1995) and production-based expert system tool set (P-BEST) (Lindqvist & Porras, 1999). Other rule-based languages exist, but they are all similar in the sense that they all specify known attack patterns as event patterns.

RUSSEL

RUSSEL is the language used in the advanced security audit trail analysis on UNIX (ASAX) project (Mounji, Charlier, Zampunieris, & Habris, 1995). It is a language specifically tailored to the problem of searching arbitrary patterns of records in sequential files. The language provides common control structures, such as conditional, repetitive, and compound actions. Primitive actions include assignment, external routine call, and rule triggering. A RUSSEL program simply consists of a set of rule declarations that are made of a rule name, a list of formal parameters and local variables, and an action part. RUSSEL also supports modules sharing global variables and exported rule declarations.

When intrusion detection is being enforced, the system analyzes the audit records one by one. For each audit record, the system executes all the active rules. The execution of an active rule may trigger (activate) new rules, raise alarms, write report messages, or alter global variables, for example. A rule can be triggered to be active for the current or the next record. In general, a rule is active for the current record because a prefix of a particular sequence of audit records has been detected. When all the rules active for the current record have been executed, the next record is read and the rules triggered for it in the previous step are executed in turn. User-defined and built-in C-routines can be called from a rule body.

RUSSEL is quite flexible in describing sequential event patterns and corresponding actions. The ability to work with user-defined C-routines gives the users the power to describe almost anything that can be specified in a programming language. The disadvantage is that it is a low-level language. Specifying an attack pattern is similar to writing a program, although it provides a general condition-trigger framework and is declarative in nature. The feature that rules can share global variables introduces the possibility of bugs along with the convenience of sharing information among different rules.

P-BEST

P-BEST was developed for the multiplexed information and computing service (Multics) intrusion detection and alerting system (MIDAS) and later employed by the intrusion detection expert system (IDES), NIDES, and the event monitoring enabling responses to anomalous live disturbances (EMERALD) (Lindqvist Y Porras, 1999). The P-BEST toolset consists of a rule translator, a library of runtime routines, and a set of garbage collection routines. Rules and facts in P-BEST are written in production rule specification language. The rule translator is then used to translate the specification into an expert system program in C language, which can then be compiled into either a stand-alone, self-contained executable program or a set of library routines that can be linked to a larger software framework.

The P-BEST language is quite small and intuitive. In P-BEST, the user specifies the structure of a fact (e.g., an audit record) through a template definition referred to as a *pattern type*. For example, an event consisting of four fields—*event_type* (an integer), *return_code* (an integer), *username* (a string), and *hostname* (a string)—can be defined as ptype[event *event_type*: int, *return_code*: int, *username*: string, *hostname*: string].

Thus, P-BEST does not depend on the structure of the input data. One important advantage of P-BEST is that it is a language preprocessor (i.e., it generates a precompiled expert system) and can extend its ability by invoking external C functions. However, it shares a similar problem with RUSSEL: It is a low-level language. Specification of attack patterns in P-BEST is time consuming. When many related rules are included in a system, correctness of the rules is difficult to check due to the interaction of these rules.

State Transition Analysis Tool Kit

Though rule-based languages are flexible and expressive in describing attack patterns for misuse detection, in practice, they are usually difficult to use. As observed in (Ilgun, Kemmerer, & Porras, 1995), “in general, expert rule-bases tend to be non-intuitive, requiring the skills of experienced rule-base programmers to update them.” STAT was developed to address this problem.

In STAT, state transition analysis technique was adopted to facilitate the specification of the patterns of known attacks (Ilgun, Kemmerer, & Porras, 1995). It is based on the assumption that all penetrations share two common features. First, penetrations require the attacker to possess some minimum prerequisite access to the target system. Second, all penetrations lead to the acquisition of some ability that the attacker does not have prior to the attacks. Thus, STAT views an attack as a sequence of actions performed by an attacker that leads from some initial state on a system to a target-compromised state, where a *state* is a snapshot of the system representing the values of all memory locations on the system. Accordingly, STAT models attacks as a series of state changes that lead from an initial secure state to a target-compromised state.

To represent attacks, STAT requires some critical actions, called *signature actions*, to be identified. Signature actions refer to the actions that, if omitted from the execution of an attack scenario, would prevent the attack from successful completion. With the series of state changes and the signature actions that cause the state changes, an attack scenario is then represented as a state transition diagram, where the states in the diagram are specified by assertions of certain conditions and the signature actions are events observable from, for example, audit data.

STAT has been applied for misuse detection in UNIX systems, distributed systems, and networks. USTAT is the first prototype of STAT, which is aimed at misuse detection in UNIX systems (Ilgun, Kemmerer, & Porras, 1995). It relies on Sun Microsystems' C2-Basic Security Module (BSM) to collect audit records. In addition to detecting attacks, USTAT is designed to be a real-time system that can preempt an attack before any damage can be done. USTAT was later extended to process audit data collected on multiple UNIX hosts. The resulting system is called *NSTAT*. NSTAT runs multiple daemon programs on the hosts being protected to read and forward audit data to a centralized server, which performs STAT analysis on all data.

A later application of STAT to network-based misuse detection resulted in another system, named *NetSTAT* (Vigna & Kemmerer, 1999). In this work, the network topology is further modeled as a hypergraph. Network interfaces, hosts, and links are considered the constituent elements of hypergraphs, with interfaces as the nodes, and hosts and links as hyperedges. Using the network topology model and the state transition description of network-based attacks, NetSTAT can map intrusion scenarios to specific network configurations and generate and distribute the activities to be monitored at certain places in a network.

STAT was intended to be a high-level tool to help specify attack patterns. Using STAT, the task of describing an attack scenario is much easier than using rule-based languages, although the analysis required to understand the nature of attacks remains the same. In the implementations of STAT techniques (i.e., USTAT, NSTAT, and NetSTAT), the attack scenarios are transformed into rule bases, which are enforced by a forward-chaining inference engine.

Colored Petri Automata

Kumar and Spafford (1994) and Kumar (1995) viewed misuse detection as a pattern-matching process. They proposed an abstract hierarchy for classifying intrusion signatures (i.e., attack patterns) based on the structural interrelationships among the events that compose the signature. Events in such a hierarchy are high-level events that can be defined in terms of low-level audit trail events and used to instantiate the abstract hierarchy into a concrete one. A benefit of this classification scheme is that it clarifies the complexity of detecting the signatures in each level of the hierarchy. In addition, it also identifies the requirements that patterns in all categories of the classification must meet to represent the full range of commonly occurring intrusions (i.e., the specification of context, actions, and invariants in intrusion patterns).

Kumar and Spafford adopted colored Petri nets to represent attack signatures, with guards to represent signature contexts and vertices to represent system states. User-specified actions (e.g., assignments to variables) may be associated with such patterns and then executed when patterns are matched. The adapted colored Petri nets are called *colored Petri automata* (CPA). A CPA represents the transition of system states along paths that lead to intruded states. A CPA is also associated with pre- and postconditions that must be satisfied before and after the match, as well as invariants (i.e., conditions) that must be satisfied while the pattern is being matched. CPA has been implemented in a prototype misuse detection system called Intrusion Detection In Our Time (IDIOT).

CPA is quite expressive; it provides the ability to specify partial orders, which in turn subsume sequences and regular expressions. However, if improperly used, the expressiveness may lead to potential problems: If the intrusions are described in every detail, the attacker may be able to change his/her attacking strategy and bypass the IDSs. Nevertheless, CPA is not the root the problem.

Automatically Build Misuse Detection Models

Lee and Stolfo (2000) looked at intrusion detection as a data analysis process and applied several data mining techniques to build misuse detection models. The research efforts were conducted under a project entitled Jam, the Java Agent for Meta-learning (meta-learning is a general strategy that provides the means of learning how to combine and integrate a number of separately learned classifiers or models). In particular, association rules and frequent episodes are used to automatically discover features that should be used to model a subject’s behavior, and the meta classification is used to combine the results of different classifiers to get better classification results.

Lee and Stolfo extended the original association rules to take into account the “order of importance” relations among the system features (the notion of association rule was discussed above regarding ADAM). *Axis* refers to the features that are important to intrusion detection; only the association rules involving axis features are considered. For example, in the shell command records, the command is likely to reflect the intrusive activities and thus be identified as an axis feature. In some sense, axis features incorporate expert knowledge into the system and thus improve the effectiveness of association rules.

To represent frequent sequential patterns of network events, Lee et al. extended frequent episodes, which was originally proposed in (Mannila, Toivonen, & Verkamo, 1995), to represent the sequential interaudit record patterns. Their algorithm finds frequent sequential patterns in two phases. First, it finds the frequent associations among event attributes using the axis attributes, and then it generates the frequent sequential patterns from these associations. The algorithm also takes advantage of the “reference” relations among the system features. That is, when forming an episode, the event records covered by the constituent item sets of the episode share the same value for a given feature attribute. The mined frequent episodes are used to construct temporal statistical features, which are used to build classification models. Thus, new features can be derived from the training data set and then used for generating better intrusion detection models.

Another innovation of the JAM project is meta classification, which combines the output of several base classifiers and generates the best results out of them. Specifically, from the predictions of base classifiers and the correct classes, a meta classifier learns which base classifier can make the best prediction for each type of input. It then chooses the prediction of the best base classifier for different input and combines the powers of the base classifiers.

Lee & Stolfo (2000) advanced state-of-the-art knowledge of intrusion detection by introducing a framework that helps generate intrusion detection models automatically. The limitation is that the framework depends on the volume of the evidences. That is, intrusive activities must generate a relatively noticeable set of events so the association of event attributes or frequent episodes can reflect them. Thus, the generated models must work with some other complementary systems, such as STAT.

Abstraction-Based Intrusion Detection

The implementation of many misuse detection approaches shares a common problem: Each system is written for a single environment and has proved difficult to use in other environments that may have similar policies and concerns. The primary goal of abstraction-based intrusion detection is to address this problem.

The initial attempt of the abstraction-based approach is a misuse detection system named the adaptable real-time misuse detection system (ARMD) (Lin, Wang, & Jajodia, 1998). ARMD provides a high-level language for abstract misuse signatures, called MuSigs, and a mechanism to translate MuSigs into a monitoring program. With the notion of abstract events, the high-level language specifies a MuSig as a pattern over a sequence of abstract events, which is described as conditions that the abstract event attributes must satisfy. The gap between abstract events and audit records is bridged by an audit subsystem, which transforms the actual audit records into abstract events. In addition, on the basis of MuSigs, the available audit trail, and the strategy costs, ARMD uses a strategy generator to automatically generate monitoring strategies to govern the misuse detection process.

ARMD is a host-based misuse detection system. In addition to the features mentioned above, it also employs database query optimization techniques to speed up the processing of audit events. The experiences with ARMD show that knowing the characteristics of the audit trail helps estimate the cost of performing misuse detection and gives the security officers the opportunity to tune the misuse detection system.

A limitation of ARMD is that it requires users to have a precise understanding of the attacks and to make careful plans for the abstraction of events. This planning is not an easy job, especially when a user does not know how his/her MuSigs may be used. In particular, unforeseen attacks may invalidate previously defined abstract events and MuSigs, thus forcing the redevelopment of some/all of the MuSigs.

The work by Ning, Jajodia, and Wang (2001) further extends the result in ARMD to address the aforementioned limitation. It provides a framework for distributed attack specification and event abstraction. In this framework, abstraction is considered an ongoing process. The structures of abstract events are represented as system views, and attack signatures are represented as generic patterns on the basis of system views. This new approach allows the semantics of a system view to be modified by defining new signatures and view definitions without changing the specifications of the views or the signatures specified on the basis of the system views. As a result, signatures in this model can potentially accommodate unknown variants of known attack patterns. Although the specification of attack signatures and the choice of right abstraction still partially depend on the users' skill, this framework provides guidance and alleviates the burden of writing and maintaining signatures.

Limitation of Misuse Detection

Current misuse detection systems usually work better than anomaly detection systems for known attacks. That is, misuse detection systems detect patterns of known attacks more accurately and generate much fewer false alarms. This better performance occurs because misuse detection systems take advantage of explicit knowledge of the attacks.

The limitation of misuse detection is that it cannot detect novel or unknown attacks. As a result, the computer systems protected solely by misuse detection systems face the risk of being comprised without detecting the attacks. In addition, due to the requirement of explicit representation of attacks, misuse detection requires the nature of the attacks to be well understood. This implies that human experts must work on the analysis and representation of attacks, which is usually time consuming and error prone. Lee and Stolfo (2000) have improved this process by automatically building the intrusion detection model; however, identification of attacks in past events is still required.

INTRUSION DETECTION IN DISTRIBUTED SYSTEMS

The rapid growth of the Internet not only provides the means for resource and information sharing, but it also brings new challenges to the intrusion detection community. Due to the complexity and the amount of audit data generated by large-scale systems, traditional IDSs, which were designed for individual hosts and small-scale networked systems, cannot be directly applied to large-scale systems.

Research on intrusion detection in distributed systems is currently focusing on two essential issues: *scalability* and *heterogeneity*. The IDSs in large distributed systems need to be scalable to accommodate the large amount of audit data in such systems. In addition, such IDSs must be able to deal with heterogeneous information from component systems of different types and that constitute large distributed systems and can cooperate with other types of IDSs.

Research on distributed intrusion detection is being conducted in three main areas. First, people are building scalable, distributed IDSs or are extending existing IDSs to make them capable of being scaled up to large systems. Second, network-based IDSs are being developed to take advantage of the standard network protocols to avoid heterogeneous audit data from different platforms. Third, standards and techniques are being developed to facilitate information sharing among different, possibly heterogeneous IDSs.

Distributed Intrusion Detection Systems

Early distributed IDSs collect audit data in a distributed manner but analyze the data in a centralized place, for example DIDS (Snapp et al., 1991) and ASAX(Mounji, Charlier, Zampunieris, & Habra, 1995). Although audit data is usually reduced before being sent to the central analysis unit, the scalability of such systems is still limited. When the size of the distributed system grows large, not only might audit data have to travel long distances before arriving at the central place, but the central analysis component of the IDS may be overwhelmed by large amount of audit data being generated.

Recent systems, such as EMERALD (Lindqvist & Porras, 1999), GrIDS (graph-based intrusion detection systems; Axelsson, 1999), and AAFID (the autonomous agents for intrusion detection system; Spafford & Zamboni, 2000), pay more attention to the scalability issue. To scale up to large distributed systems, these systems place IDS components in various places in a distributed system. Each of these components receives audit data or alerts from a limited number of sources (e.g., hosts or other IDS components), so the system is not overwhelmed by large amounts of audit data. Different components are often organized hierarchically in a tree structure; lower level IDS components disseminate their detection results to higher level components, so the intrusion related information from different locations can be correlated together.

Although most of the recent distributed IDSs are designed to be scalable, they only provide a partial solution to the scalability problem. The IDS components are either coordinated in an ad hoc way or are organized hierarchically. Although coordinating the IDS components in an ad hoc way is certainly not a general solution, organizing the components hierarchically does not always provide an efficient solution, especially when the suspicious activities are spread in different, unpredictable locations in a large system. In a hierarchical system, when the activities involved in a distributed attack fall beyond the scope of one IDS component, the audit data possibly related to the attack will have to be forwarded to a higher level IDS component to be correlated with data from other places. In a worst-case scenario, the audit data may have to be forwarded several times before arriving at a place where the data can finally be correlated with the related information. This process not only wastes the network's bandwidth, it also limits the scalability of the detection of distributed attacks.

The abstraction-based intrusion detection (Ning, Jajodia, & Wang, 2001) addresses this problem by generating a hierarchy of IDS components dynamically rather than statically. Intuitively, this method defines attack signatures as generic patterns (called *generic signatures*) of abstract events that may be observed in different places in a distributed system. When a particular type of attack is to be detected in a distributed system, the corresponding generic signature is mapped to the specific systems. The resulting signature is called a *specific signature*. This method then decomposes the specific signature into components called *detection tasks*, each of which corresponds to the intrusion detection activities required to process a type of event involved in the attack. A coordination mechanism is developed to arrange the messages passing between the detection tasks, so the distributed detection of attacks is equivalent to having all events processed in a central place. The abstraction-based method is more flexible and more efficient than the previous methods; however, it is limited in that it is applicable only to misuse detection.

Network-Based Intrusion Detection Systems

Network-based IDSs collect audit data from the network traffic, as opposed to host-based IDSs, which usually collect audit data from host audit trails. Examples of network-based IDSs include NSM (Network Security Monitor; Axelsson, 1999), NetSTAT (Vigna & Kemmerer, 1999), and Bro (Axelsson, 1999).

One challenge that the network-based intrusion detection is facing is the speed of high-performance networks. The great speed of the high-performance networks makes it very difficult to capture the network traffic, let alone perform intrusion detection in real time.

Several efforts have addressed enabling intrusion detection in high-speed networks. Snort, an open source IDS that specializes in network intrusion detection (Snort, 2002), was developed by Roesch (1999). It employs a fast pattern-matching algorithm to detect network misuse. However, early versions of Snort detected attacks individually, and its performance degrades when the number of attack signatures (rules) increases. Since version 1.9, Snort has incorporated new pattern-matching algorithms to address this problem.

Sekar, Guang, Verma, and Shanbhag (1999) developed a high-performance network IDS based on efficient pattern-matching algorithms. A distinguishing feature is that the performance of the system is independent of the number of misuse rules (signatures).

Kruegel, Caleur, Vigna, and Kemmerer (2002) proposed a partition approach to intrusion detection that supports misuse detection on high-speed network links. This approach is based on a slicing mechanism that divides the overall network traffic into subsets of manageable size. Thus, each subset can be processed by one or several misuse detection systems. The traffic partitioning is done such that each subset of the network traffic contains all of the evidence necessary to detect a specific attack.

Network-based IDSs offer several advantages. First, network-based IDSs can take advantage of the standard structure of network protocols, such as TCP/IP. This is a good way to avoid the confusion resulting from heterogeneity in a distributed system. Second, network-based IDSs usually run on a separate (dedicated) computer; thus, they do not consume the resources of the computers that are being protected.

Conversely, network-based IDS are not silver bullets. First, because these IDSs do not use host-based information, they may miss the opportunity to detect some attacks. For example, network-based IDSs cannot detect an attack launched from a console. Second, the standard network protocols do not solve the entire problem related to the heterogeneity of distributed systems because of the variety of application protocols and systems that use these protocols. For example, network-based IDSs must understand the UNIX shell commands if their goal is to monitor intrusive remote log-ins. As another example, network-based IDSs usually describe the suspicious network activities using the structure of the packet that standard network protocols such as TCP/IP support, which makes the specification of the suspicious activities to be detected very difficult.

Network-based IDSs have the same scalability problem as do general distributed IDSs. For example, existing network-based IDSs analyze network traffic data in a centralized place, as was done by the early distributed IDSs, although they may collect data from various places in the network. This structure limits the scale of the distributed systems that such IDSs can protect.

Sharing Information Among Intrusion Detection Systems

With the deployment of so many commercial IDSs, these IDSs must be able to share information so that they can interact and thus achieve better performance than if operating in isolation. Research and development activities are currently under way to enable different, possibly heterogeneous, IDSs to share information.

The Common Intrusion Detection Framework (CIDF) was developed to enable different intrusion detection and response (IDR) components to interoperate and share information and resources(Porras, Schnackenberg, Staniford-Chen, Stillman, & Wu, 1999). It began as a part of the Defense Advanced Research Project Agency (DARPA) Information Survivability program, with a focus on allowing DARPA projects to work together. CIDF considers IDR systems as composed of four types of components that communicate via message passing: event generators (E-boxes), event analyzers (A-boxes), event databases (D-boxes), and response units (R-boxes). A communication framework and a common intrusion specification language are provided to assist the interoperation among CIDF components.

Researchers involved in CIDF started an Intrusion Detection Working Group (IDWG) in the Internet Engineering Task Force (IETF), trying to bring the impact of CIDF to a broader community. The IDWG has been working to develop data formats and exchange procedures for sharing information among IDSs, response systems, and management systems. The extensible markup language (XML) has been chosen to provide the common format, and an intrusion detection message exchange format (IDMEF) has been defined in an Internet draft. The IDWG uses the blocks extensible exchange protocol (BEEP) as the application protocol framework for exchanging intrusion detection messages between different systems; an intrusion detection exchange protocol (IDXP) is specified as a BEEP profile, and a tunnel profile is provided to enable different systems to exchange messages through firewalls. At the time this writing, these Internet drafts are under consideration for being adopted as IETF requests for comments (RFCs).

A negotiation protocol for CIDF components has been developed as part of the intrusion detection intercomponent adaptive negotiation (IDIAN) project(Feiertag, et al., 2000). It allows distributed IDS components to discover the intrusion detection services of other components, and to negotiate, manage, and adjust the use of those services. In particular, the notion of a filter was introduced to assist the negotiation process. A filter is essentially a pattern of CIDF messages, and the negotiation process helps determine one or several filters between two CIDF components. A CIDF component sends a CIDF message to another component only when the message matches the filters of the receiver. Similarly, a receiving CIDF component accepts an incoming CIDF message only if the message matches one of its filters. IDIAN partially addresses the information-sharing problem by enabling different IDS components to discover, negotiate, and adjust the use of services from each other.

Another effort in sharing information among different IDSs is the Hummer project (Frincke, Tobin, McConnell, Marconi, & Polla, 1998). In particular, the relationships among different IDSs (e.g., peer, friend, manager/subordinate) and policy issues (e.g., access control policy, cooperation policy) were studied, and a prototype system, HummingBird, was developed to address these issues. A limitation of the Hummer project is that it only addresses the general data-sharing issue; what information needs to be shared and how the information would be used are out of its scope. Thus, it should be used along with mechanisms such as IDIAN (Feiertag, et al, 2000) and the decentralized coordination mechanism in the abstraction-based approach (Ning, Jajodia, & Wang, 2001).

CONCLUSION

Intrusion detection continues to be an active research field. Even after 20 years of research, the intrusion detection community still faces several difficult problems. How to detect unknown patterns of attacks without generating too many false alerts remains an unresolved problem, although recently, several results have shown there is a potential resolution to this problem. The evaluating and benchmarking of IDSs is also an important problem, which, once solved, may provide useful guidance for organizational decision makers and end users. Moreover, reconstructing attack scenarios from intrusion alerts and integration of IDSs will improve both the usability and the performance of IDSs. Many researchers and practitioners are actively addressing these problems. We expect intrusion detection to become a practical and effective solution for protecting information systems.

GLOSSARY

Anomaly Detection One of the two methodologies of intrusion detection. Anomaly detection is based on the normal behavior of a subject (e.g., a user or a system); any action that significantly deviates from the normal behavior is considered intrusive. The other methodology is misuse detection.

Audit Trail Records a chronology of system resource usage. This includes user log-in, file access, other various activities, and whether any actual or attempted security violations occurred.

False Negative An actual intrusive action that the system allows to pass as nonintrusive behavior.

False Positive Classification of an action as anomalous (a possible intrusion) when it is legitimate.

IDS Intrusion detection system.

Intrusion Any activity that violates the security policy of an information system.

Intrusion Detection The process of identifying intrusions by observing security logs, audit data, or other information available in computer systems and/or networks.

Misuse Detection One of the two methodologies of intrusion detection. Catches intrusions in terms of the characteristics of known patterns of attacks or system vulnerabilities; any action that conforms to the pattern of a known attack or vulnerability is considered intrusive. The other methodology is anomaly detection.

Misuse Signature A known pattern or attack or vulnerability, usually specified in a certain attack specification language.

Profile A set of parameters used to capture the pattern of a subject's (e.g., a user or a program) normal behavior. It is normally used to conduct anomaly detection.

Security Policy A set of rules and procedures regulating the use of information, including its processing, storage, distribution, and presentation.

CROSS REFERENCES

REFERENCES

Axelsson, S. (1999). *Research in intrusion-detection systems: A survey. Technical report TR 98-17.* Göteborg, Sweden: Department of Computer Engineering, Chalmers University of Technology.

Barbara, D., Couto, J., Jajodia, S., & Wu, N. (2001). ADAM: A testbed for exploring the use of data mining in intrusion detection. *ACM SIGMOD Record*, *30* (4), 15--24.

Barbara, D., Couto, J., Jajodia, S., & Wu, N. (2002). An architecture for anomaly detection. In D. Barbara & S. Jajodia (Eds.), *Applications of Data Mining in Computer Security* (pp. 63--76). Boston: Kluwer Academic.

Feiertag, R., Rho, S., Benzinger, L., Wu, S., Redmond, T., Zhang, C., Levitt, K., Peticolas, D., Heckman, M., Staniford, S., & McAlerney, J. (2000). Intrusion detection inter-component adaptive negotiation. *Computer Networks*, *34*, 605--621.

Fox, K.L., Henning, R.R., Reed, J.H., & Simonian, R.P. (1990). A neural network approach towards intrusion detection. In NIST (Ed.), *Proceedings of 13th National Computer Security Conference* (pp. 125--134), National Institute of Standards and Technology (NIST), Baltimore, MD.

Fríncke, D., Tobin, D., McConnell, J., Marconi, J., & Polla, D. (1998). A framework for cooperative intrusion detection. In NIST (Ed), *Proceedings of the 21st national information systems security conference* (pp. 361-373), National Institute of Standards and Technology (NIST), Baltimore, MD.

Ghosh, A.K., Wanken, J., & Charron, F. (1998). Detecting anomalous and unknown intrusions against programs. In K. Keus (Ed), *Proceedings of the 14th annual computer security applications conference* (pp. 259--267). IEEE Computer Society, Los Alamitos, CA.

Hofmeyr, S., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security, 6*, 151--180.

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

Ko, C., Ruschitzka, M., & K. Levitt (1997). Execution monitoring of security-critical programs in distributed systems: a apecification-based approach. In G. Dinolt & P. Karger (Eds.), *Proceedings of 1997 IEEE symposium of security and privacy* (pp. 175-187), IEEE Computer Socitey, Los Alamitos, CA.

Ko, C. (2000). Logic induction of valid behavior specifications for intrusion detection. In M. Reiter & R. Needham (Eds.), *Proceedings of 2000 IEEE symposium of security and privacy* (pp. 142--153), IEEE Computer Socitey, Los Alamitos, CA.

Kruegel, C., Valeur, F., Vigna, G., & Kemmerer, R. (2002). Stateful intrusion detection for high-speed networks. In M. Abadi & S. Bellovin (Eds.), *Proceedings of 2002 IEEE symposium on security and privacy* (pp. 285--293) , IEEE Computer Socitey, Los Alamitos, CA.

Kumar, S, (1995). *Classification and detection of computer intrusions*. Unpublished doctoral dissertation, Purdue University, West Lafayette, IN.

Kumar, S., & Spafford, E.H. (1994). A pattern-matching model for misuse intrusion detection. In NIST (Ed.), *Proceedings of the 17th national computer security conference* (pp. 11--21), National Institute of Standards and Technology (NIST), Baltimore, MD.

Lane, T., & Brodley, C.E. (1998). Temporal sequence learning and data reduction for anomaly detection. In L. Gong & M. Reiter (Eds.), *Proceedings of 5th conference on computer and communications security* (pp. 150--158), ACM Press, New York, NY.

Lee, W., & Stolfo, S.J. (2000). A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, *3* (4) (pp. 227-261).

Lee, W., & Xiang, D. (2001). Information-theoretic measures for anomaly detection. In R. Needham & M. Abadi (Eds), *Proceedings of 2001 IEEE symposium on security and privacy* (pp. 130--143), IEEE Computer Society, Los Alamitos, CA.

Lin, J., Wang, X.S., & Jajodia, S. (1998). Abstraction-based misuse detection: High-level specifications and adaptable strategies. In S. Foley (Ed.), *Proceedings of the 11th computer security foundations workshop* (pp. 190--201), IEEE Computer Socitey, Los Alamitos, CA.

Lindqvist, U., & Porras, P.A. (1999). Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In L. Gong & M. Reiter (Eds.), *Proceedings of the 1999 IEEE symposium on security and privacy* (pp. 146--161), IEEE Computer Socitey, Los Alamitos, CA.

Mannila, H., Toivonen, H., & Verkamo, A.I. (1995). Discovering frequent episodes in sequences. In U. Fayyad & R. Uthurusamy (Eds.), *Proceedings of the 1st conference on knowledge discovery and data mining* (pp. 210--215), AAAI Press, Menlo Park, CA.

Mounji, A., Charlier, B.L., Zampuniéris, D., & Habra, N. (1995). Distributed audit trail analysis. In D. Balenson & R. Shirey (Eds.), *Proceedings of the ISOC'95 symposium on network and distributed system securit*y (pp. 102--112), IEEE Computer Society, Los Alamitos, CA.

Ning, P., Jajodia, S., & Wang, X.S. (2001). Abstraction-based intrusion detection in distributed environments. *ACM Transactions on Information and System Security*, *4* (4), 407--452.

Porras, P., Schnackenberg, D., Staniford-Chen, S., Stillman, M., & Wu, F. (1999). *Common intrusion detection framework architecture*. Retrieved on February 13, 2003 from http://www.isi.edu/gost/cidf/.

Roesch, M. (1999). Snort -- lightweight intrusion detection for networks. In D. Parter (Ed.), *Proceedings of the 13th Systems Administration Conference*, Retrieved on February 13, 2003 from http://www.usenix.org/publications/library/proceedings/lisa99/technical.html.

Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In R. Needham & M. Abadi (Eds), *Proceedings of 2001 IEEE symposium on security and privacy* (pp. 144--155), IEEE Computer Socitey, Los Alamitos, CA.

Sekar, R., Guang, Y., Verma, S., & Shanbhag, T. (1999). A high-performance network intrusion detection system. In J. Motiwalla & G. Tsudik (Eds.), *Proceedings of the 6*th *ACM conference on computer and communications security* (pp. 8--17), ACM Press, New York, NY.

Snapp, S.R., et al. (1991). DIDS (distributed intrusion detection system)---Motivation, architecture, and an early prototype. In *Proceedings of 14th national computer security conference* (pp. 167--176).

Snort---The open source intrusion detection system. (2002). Retrieved February 13, 2003, from http://www.snort.org.

Spafford, E.H., & Zamboni, D. (2000). Intrusion detection using autonomous agents. *Computer Networks 34*, 547--570.

Teng, H.S., Chen, K., & Lu, S.C. (1990). Adaptive real-time anomaly detection using inductively generated sequential patterns. In *Proceedings of 1990 IEEE symposium on security and privacy* (pp. 278--284), IEEE Computer Socitey, Los Alamitos, CA.

Vigna, G., & Kemmerer, R.A. (1999). NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, *7* (1), 37--71.

Wagner, D., & Dean, D. (2001). Intrusion detection via static analysis. In R. Needham & M. Abadi (Eds), *Proceedings of 2001 IEEE symposium on security and privacy* (pp. 156--168), IEEE Computer Socitey, Los Alamitos, CA.