1-1-2000

Data mining approaches for detecting intrusion using UNIX process execution traces

Xiaoning Zhang
Iowa State University

Data mining approaches for detecting intrusion using UNIX process execution traces

by

Xiaoning Zhang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Business

Major Professor: Dan Zhu

Iowa State University

Ames, Iowa

2000

Graduate College

Iowa State University

This is to certify that the Master's thesis of

Xiaoning Zhang

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In this chapter, we give a broad overview of the field of network security and the motivation for employing various techniques to automate audit data analysis in the intrusion detection systems. In the following chapters, we will survey the intrusion detection methods in detail.

1.1 Overview

During the past few years, information technology has become a key component to support the critical infrastructure of the world. The public telephone network, banking and finance, vital human services, and other critical infrastructures are dependent upon information technology for their day-to-day operation. Every day, millions of people use computers to process or search for different kinds of information. While most of these users access data legitimately, some use illicit ways to access computers. Thus, computer security becomes a major concern among various industries, especially those organizations depending heavily on networks.

Because of the convenience, ease of use, and ability to take advantage of rapid advances in the commercial market, organizations are continuing to expand their enterprise networks. As they connect their local area networks into wide area enterprise networks, these networks become more complex (Chin, 1999). In an effort to share information and streamline operations, organizations open their networks to business partners, suppliers, and other outsiders. In addition, organizations connect their internal networks to the Internet to give employees access to Internet resources and external users access to internal web servers for conducting e-commerce transactions. Increasing network complexity, greater access, and a growing emphasis on the Internet are causing organizations to feel more and more insecure about their networks. These three trends result in significantly higher exposure to both internal and external attacks. Some of the attacks come from the inside the organization because of misuse within the system, while many attacks are carried out by outside hackers. According to a leading survey, the number of computer security breaches has risen 22% between 1996 to 1998, with $136 million associated losses. The U.S. General Accounting Office (GAO) disclosed that approximately 250,000 break-ins into Federal computer systems were attempted during the year of 1996 and 64% of these attacks were successful. Worse yet, the number of attacks is doubling every year. Based on previous studies, the GAO estimates that only 1- 4% of these attacks will be detected and only about 1% will be reported (Durst et al., 1999). In the beginning of February 2000, several major web sites including Yahoo, Amazon, eBay, Datek, and eTrade were shut down because of the denial-of-service attacks created on their web servers.

Since computer systems are increasingly vulnerable to attack, the network security is an important issue for most organizations. A variety of security devices and actions are being deployed to secure networks. There are mainly two types of techniques: protection and detection.

Protection techniques are designed to guard hardware, software and user data against threats from both outsiders as well as from malicious insiders. One type of security device receiving considerable amount of attention is the firewall. A firewall interposes a barrier at the point of connection between the Internet and the corporate internal networks to keep out attacks. A recent survey in InfoWeek shows that 20% of the respondents already have a firewall and over 40% plan to implement one. However, firewalls are not foolproof. They require the accurate configuration of numerous and confusing access control lists. These configurations must be continually updated to allow access to new network services and to keep up with changing security policies. Even properly configured firewalls have known weak spots. In many situations, hackers can circumvent firewalls entirely and enter the system without being noticed. Another disadvantage is that firewalls cannot prevent attacks from the inside the network, which is also a frequent source of break-ins.

In addition to firewalls, encryption and operating system access controls also provide protective functions. Operating systems provide user authentication through passwords and multi-level access control to information. Encryption techniques can hide the true information by converting it into an unrecognized format. Regardless of these strong protection measures, people cannot be completely successful in preventing problems. Hackers continually find new ways to break into or interfere with systems. Moreover, most mechanisms are powerless against misbehavior by legitimate users who perform an authorized actions.

Intrusion detection systems (IDS) and vulnerability assessment systems (also known as scanners) are a good complement of protective tools for network security. Organizations have eagerly adopted detective products in the last several years. In a report by the Internet Security Systems Inc, they have supplied more than 10,000 detection engines for companies and government sites. According to industry estimates, the market for intrusion detection products has grown from $40 million in 1997 to $100 million in 1998.

Vulnerability assessment products, such as SATAN and COPS (Farmer and Spafford, 1990), perform rigorous examinations of systems in order to determine weaknesses that might allow security violations. These products inspect system configuration files for problematic settings, system password files for weak passwords, and other system objects for security policy violations. Although these systems cannot reliably detect an attack in progress, they can determine the possible weak points for attack.

IDS help computer systems prepare for and deal with attacks. They collect information from a variety of systems and network sources, then analyze the information for signs of intrusion and misuse. The majority of functions performed by IDS include:

- monitoring and analyzing user and system activity,
- assessing the integrity of critical system and data files,
- recognizing activity patterns reflecting known attacks,
- responding automatically to detected activity, and
- reporting the outcome of the detection process.

The success of an intrusion detection system can be characterized by both false alarm rates and detection efficiency (Stillerman et al., 1999). Since the audit services record the occurrence of all security-relevant events, which result in an enormous quantity of audit data, using the correct method to analyze the proper set of system data becomes an important issue to keep false alarm rates to low level and have high detection efficiency.

1.2 Purpose of the Study

In recent years, artificial intelligence technologies are increasingly being incorporated in intrusion detection systems for analyzing audit data. AI technologies, such as rule induction, Rough Sets, and neural networks are used for classification and pattern recognition in many industries including finance, manufacturing, and biological sciences. Since intrusion detection involves identification of anomalies in audit data, these AI techniques can be effectively employed in this context.

The data we used comes from the web site of the computer immune system, a network security research project conducted by Professor Forrest at the University of New Mexico and her students. They and another research group at Columbia University (Lee and Stolfo, 1997, 1998) analyzed this set of data using different techniques. In our experiment, we used a binary representation method different from those in their studies, and tried to determine whether C4.5, Rough Sets and Backpropagation net can discover the patterns behind the sequence-related data as efficiently as their methods.

1.3 Structure of the Thesis

This thesis includes five parts. Following this introduction, there is a comprehensive survey of existing intrusion detection methods; the third chapter is a review of the main methodologies utilized in our experiment including C4.5, Backpropagation net and Rough Sets. The fourth chapter explains in detail the experiment and results for the comparison between the three methodologies. The last chapter presents the conclusions of our experiment and directions for future research.

CHAPTER 2

LITERATURE REVIEW

In this chapter, first we will have a thorough discussion about the intrusion detection system, especially the analysis methods and their advantages and disadvantages. Then, we will give an overview of the analysis data used in IDS and focus on research of the system calls.

2.1 IDS Analysis Approaches

IDS can be generally divided into two categories according to the analysis method. These categories are misuse detection system and anomaly detection system. Misuse detection system detects attacks based on well-known vulnerabilities and intrusions stored in a database, while anomaly detection system detects deviation of activity from normal profiles.

2.1.1 Misuse Detection

Misuse detection is signature-based detection. When the intrusion signatures are matched by new audit data, the detector flags intrusions and sets off an alarm.

Signatures are patterns corresponding to known attacks or misuses of systems. They may be simple (character string matching looking for a single term or command) or complex (security state transition written as a formal mathematical expression). In general, a signature can be concerned with a process (the execution of a particular command) or an outcome (the acquisition of a root shell). Most commercial intrusion detection products perform signature analysis against a vendor-supplied database of known attacks. Additional signatures specified by the customer can also be added as part of the intrusion detection system configuration process.

The techniques used for misuse detection are expert systems, model-based reasoning systems, state transition analysis, genetic algorithm, fuzzy logic, and keystroke monitoring. The discussions of various misuse detection approaches are in the following.

2.1.1.1 Expert Systems

Human experts encode the knowledge about vulnerabilities and past intrusions in "if-then" rules. Conditions causing intrusions are specified in the "if" part, and reactions to the relative attacks are encoded in the "then" part. When all these conditions are satisfied, the actions corresponding to them occur. Examples using this implementation are described in various studies (Snapp and Smaha, 1992; Porras and Valdes, 1998). In the EMERALD system developed by SRI International (Porras and Valdes, 1998), signature engines are used to analyze an event stream. If the stream is mapped to an abstract representation of abnormal event sequences, it indicates danger to the system. The signature engine is essentially an expert system whose rules indicate suspicious activities.

2.1.1.2 State Transition Analysis

State Transition Analysis Tool (STAT) is a method in which penetration is viewed as a sequence of actions that make the system from the initial state prior to an attack to the final compromised state after the attack occurs (Porras, 1992). The system converts penetration scenarios into state transition diagrams, which are special rules representing dangerous activities. States in the attack pattern correspond to system states and have boolean assertions associated with them that must be satisfied to transit to that state. Successive states are connected by arcs that represent the events required for changing state.

2.1.1.3 Keystroke Monitoring

This technique utilizes user keystrokes to determine the occurrence of an attack. Primarily, there should be a pattern match for specific keystroke sequences that indicate an attack. The disadvantage of this approach is the lack of reliable mechanisms for user keystroke capture without the operating system support.

2.1.1.4 Model-Based Reasoning Intrusion Detection

In this approach, a database of attack scenarios is specified in terms of sequence of user behavior (Garvey and Lunt, 1991). The system would reason about hypothesized intrusions by gathering evidence from audit trails and statistical profiles, and determining the likelihood of specific hypothesized intrusion scenarios. If one model of intrusion is discovered, the system continues to predict the steps expected to occur in the intrusion scenario, until enough evidence is above a threshold value confirms the intrusion.

2.1.1.5 Genetic Algorithms

Genetic algorithms, proposed by Holland (1975), are optimum search algorithms based on the mechanism of natural selection in a population. The potential solutions to the problem are represented in strings. The population is randomly initialized; in every generation, mutation and crossover are applied to the strings to obtain new strings; then a set of fittest solution is chosen from the combined population for the next generation. The *fitness* of each individual is simply the value of the function to be optimized (the fitness function).

Ludovic Mé (1998) proposed an approach based on predefined attack scenarios and used a genetic algorithm. The goal was to determine, among all the possible attack subsets and according to the events recorded in the audit trail, the one presenting the greatest risk to the system. They represented a string in an N_a length of individuals with values of 1 or 0, where N_a is the number of potential attacks. The fitness function is given as:

$$\text{Fitness} = \alpha + (\sum_{i=1}^{N_a} W_i I_i - \beta T_e^2),$$

where I is an individual, W_i is the weight associated with the attack i, and βT_e^2 is a penalty function. The β parameter makes it possible to modify the slope of the penalty function, and α sets a threshold to create a positive fitness function making the fitness positive. The three basic operators they used were proportionate selection, one-point crossover, and simple mutation.

2.1.1.6 Time-Based Inductively Machine (TIM)

This approach was proposed by Teng et al. (1990). The main difference between TIM and other rule-based detection systems is that TIM focuses on patterns of event sequences, whereas the others focus on individual events. It is an approach using a time-based inductive engine to generate rule-based symbolic sequential patterns from observations of a temporal process. These patterns are used to predict the next possible events with satisfactory accuracy. When the subsequent events deviate significantly from the established patterns, it will be considered a violation of their rules, indicating the possibility of an intrusion.

2.1.1.7 Fuzzy Logic

RETISS is a real time security system for threat detection developed by F. Carrettoni et al. (1991). It is based on the assumption that there exists a correlation between anomalous user behavior and threats. There are security rules to be enforced to express this correlation. Levels of danger of the corresponding anomalies in term of its occurrences and of the subject and object involved are defined in a weight table for each rule. The weight table is a 7-column table with the form

< Subj_class, Obj_class, Action, Anomaly, Occurrence, Weight, Threshold>.

These levels are combined using fuzzy logic to express the probability of a given threat.

There are several components in this system: the Audit Record File (ADF), the Data Base (DB), the Knowledge Base (KB), the Audit Record Analyzer (ARA), the Threat Detection Module (TDM), the Profile Updating Module (PUM), and the Interface Module (IM). Among them, ARA, TDM, and KB are the key components enforcing the detection control. ARA is used to examine the ADF to evaluate if there are audit records corresponding to suspicious actions in the KB. Upon reception of an anomaly record by the ARA, the TDM checks the KB, firing every rule with a trigger matching the anomaly record under consideration. When a rule is fired, its antecedent is evaluated. If the antecedent is not satisfied, then no threat is pointed out, otherwise the system must evaluate how dangerous the threat is. Evaluation of the level of danger of anomalous behavior is enforced using the weight table associated to the rule. Tuple's weight is multiplied by the user_weight of the user specified in the anomaly record under consideration to get the evaluation_ record_weight. All these values are combined replacing AND/OR operators in the rule antecedent. The result is a numerical value expressing the level of danger of a given threat. If the value is greater than the specified threshold, then an alarm is generated.

2.1.1.8 Summary of Misuse Approach

One advantage of misuse detection is that it allows sensors to collect a more tightly targeted set of system data, thereby reducing the system overhead. It also has some limitations. An obvious one is that all the rules and models are set up based on human knowledge of attacks. Since there are attacks which have never been tried, or vulnerabilities yet to be discovered, we cannot always make a complete list of rules for the detection system. Furthermore, not every attack can be perfectly represented by the "expert system" or some kinds of rules. If an intrusion scenario does not trigger a rule or triggers an incomplete rule, it will not be detected by this rule-based approach. Last, because new attacks or new versions of attacks appear continuously to replace those old scenarios, the complexity of the databases grows as the number of well-known attacks grows, introducing problems of scale. It is difficult to keep them updated as the catalog of attacks grows.

2.1.2 Anomaly Detection

The goal of the anomaly detection system is to recognize behaviors that deviate from normal activities. In different anomaly detection approaches, the normal profiles include various kinds of measuring variables of the system objects, and they are represented in different styles. However, the basic working theory is to compare the observed activities with the normal profiles and find the deviations. There are always some thresholds to specify for the detection system. If the deviation is higher than the threshold, the computer system will be considered to be attacked, otherwise, the deviation will be neglected. The challenge is to define the threshold in a way that minimizes the false alarm rate and maximizes the detection efficiency.

2.1.2.1 Statistical Approach

Statistical profiles are created for system objects (e.g., users, files, directories, devices, etc.) by measuring various attributes of normal use (e.g., number of accesses, number of times an operation fails, time of day, etc.). Mean frequencies and measures of variability are calculated for each type of normal usage. Possible intrusions are signaled when observed values fall outside the normal range. For example, statistical analysis might signal an unusual event if an accountant who had never previously logged into the network outside the hours of 8 AM to 6 PM were to access the system at 2 AM.

NIDES (Lunt et al. 1992) is an example of a statistical approach. The anomaly detector observes the activity of subjects and generates profiles for them to represent their behaviors. There are several types of measures (M_1, M_2, M_3, ..., M_n) comprising a normal profile, every measure has a value (S_1, S_2, S_3, ..., S_n) presenting the abnormality of the profile. A higher value of S_i indicates a greater abnormality. A combining function of the individual S values may be the weighted sum of its squares, as in

$$a_1S_1^2 + a_2S_2^2 + a_3S_3^2 + \ldots + a_nS_n^2 \qquad a_i > 0,$$

where a_i reflects the weight of the metric M.

In the previous part of this chapter, there was an introduction of the expert system of EMERALD (Porras and Valdes 1998). This system also includes a statistical analysis tool. There are four classes of measures employed by statistical algorithms—categorical, continuous, intensity, and event distribution. The system maintains a description of a subject's behavior in a profile subdivided into short- and long- term elements. The long-term profile is adapted to changes in subject activity slowly, while the short- term profile represents recent activities and accumulates values between updates. The differences between short- and long- term profiles is compared to a historically adaptive, subject-specific deviation, then the distribution of this deviation is transformed to obtain a score for the event. An anomaly event has a higher score than certain thresholds.

The advantage of the statistical approach is that statistics has been well studied. It provides us with a complete theory base for research. However, there does exist some disadvantages. First, it is insensitive to the order of occurrence of events, which will miss an important measure for intrusion detection. Second, intruders can gradually train it so that a behavior once regarded as abnormal will be normal after intended training. Also, it is difficult to decide the threshold beyond which there will be an intrusion considered.

2.1.2.2 TIDE and STIDE

Hofmeyr et al. (1998,1997) and Forrest et al. (1996) at the University of New Mexico developed these two methods, time-delay embedding (TIDE) and sequence time-delay embedding (STIDE). They used *lookahead pairs* or fixed length sequences of system calls to set up the normal profile, compared them with certain process traces, and utilized some statistical indexes, such as local frame frequency and Hamming Distance to analyze the comparison results. We will discuss this method in detail later in this chapter.

2.1.2.3 Neural Networks

There are several neural network algorithms employed in intrusion detection research. Bonifacio et al. (1997) and Herve DeBar et al. (1992) used Backpropagation, QuickProp and Rprop. Kohonen's self-organizing feature map was used by Fox et al. (1990).

The basic idea of these methods is to train the neural net with both normal and abnormal system features at first. The weights of each neuron (or node) are adjusted in the process of training. When it is used in the real detection system, it can classify the audit data into normal and abnormal categories based on its well-trained weights.

Some advantages of this approach are:

- The success of this approach does not depend on any statistical assumptions about the nature of the underlying data.
- Neural nets cope well with noisy data.
- Neural nets can automatically account for correlation between the various measures that affect the output.

On the other hand, training a neural net is a time-consuming task. Parameters of the neural network topology and learning are determined only after considerable trial and error. Another disadvantage is that although the neural networks get a satisfied result, it is still hard to determine theory explanations about the results.

2.1.2.4 Machine Learning Approach

Rule induction technique is widely used in the classification, pattern recognition, and optimization problems. Lee and Stolfo (1997) used RIPPER (Cohen 1995), a rule learning method, to learn the normal and/or abnormal patterns from certain audit system features. Since they used the same set of data employed in our experiment, we will discuss their experiment in detail later in this chapter.

2.1.2.5 Immune System

Forrest et al. (1996, 1993) proposed an artificial immune system simulating the human immunology system to detect intrusions. Kim and Bentley (1999a, b, c) proposed a negative selection algorithm based on Forrest et al.'s immune system. Their primary theory is that a computer security system should protect itself from unauthorized intruders, which is similar to the human immune system protecting the body (self) from invasion by inimical microbes (non-self). In the experiment by Hofmeyr et al. (1999), they compressed each network connection into a single 49-bit binary string. Self is a set of normal occurring connections, while non-self is a set of connections not normally observed on the networks.

There are three learning mechanisms used by immune system. Negative selection is the first step, which means randomly created detectors (also 49-bit strings) would be killed when they match any one string in the "self." After this period, survived detectors become mature. The second step is maturation of naïve cells into memory cells. During some period after detector becomes mature, there will be new connection matches its pattern. If there are a sufficient number of packets detected (in another words, they match some patterns of the mature detectors), an alarm will be raised. But if a detector failed to match any connections, it will be killed too for its uselessness. This step can compress the detectors into a minimized set in order to save the system resource and speed up the detection process. The third step is some kind of genetic algorithm method called "affinity maturation." Since it is possible all redundant detectors to similar intrusions survived in a single "self" set, there will be a waste of system resource. So, the immune system is designed to make detectors compete against each other, the ones with the closest match (greatest fitness) will win and survive at last.

After these three steps, an efficient "self" detector set will be created and can be used to detect intrusion in the future. Since detectors can be randomly created continuously, this "self" set also can be updated from time to time.

2.1.2.6 Summary of Anomaly Approach

The advantage of anomaly detection is that it is best suited to detect unknown vulnerabilities. There are also some disadvantages of this analysis method. First, the good performance of anomaly detection system is based on the assumption that the system must have established an appropriate baseline for the user, it must have all the normal behavior profiles. If this cannot be accomplished, the false alarm rate will be high as the system alerts whenever some “anomaly” is detected, even though there is no intrusion at all. Another disadvantage is that it is relatively easy for an adversary to trick the detector into accepting attack activity as normal by gradually varying behavior over time. The last disadvantage is that anomaly detectors do not deal well with changes in user activities. This rigidity can be a problem in organizations where change is frequent and can result in both false positives (false alarms) and false negatives (missed attacks).

Both misuse and anomaly detections have advantages and disadvantages. There are some detection systems combining these two approaches. IDES (Lunt 1988) and EMERALD (Porras and Valdes 1998) are two examples of this kind of intrusion detection system.

2.2 Strategies for Data Collection and Analysis

No matter which approach is used in the intrusion detection system, collecting data on activities is the most critical task. There are several sources for data collection.

There is difficulty in selecting a proper set of features to represent a known intrusion pattern (in misuse detection) or a normal user’s profile (in anomaly detection). Determining the right measures is complicated, because there are so many features to be obtained directly or indirectly from the audit data. But since they are not designed for the purpose of intrusion detection originally, a subset is selected which can accurately predict or classify intrusions in real-time.

Some IDS monitor the network traffic packets on the TCP/IP level using a packet-capturing program, such as tcpdump. There is information in the packets that can be used for detection, such as type of connections, connection ID, source and destination address and ports, connection start time, the statistics of the connection (e.g., bytes sent in each direction, resent rate, etc.), and flag ("normal" or one of the recorded connection/termination errors), etc. Many studies have used this information in their research (Bonifacio et al., 1997, Carrettoni et al., 1991, Heberlein et al., 1990, Lunt, 1988, Lee and Stolfo, 1998, Lunt 1993b, and Teng et al., 1990).

IDS can either monitor network traffic packets, or monitor the audit trails on each host of the network and correlate the evidence from the hosts. There are different levels at which an IDS can monitor a host, including the keyboard level, system call level, command level, and application level. They are discussed below.

2.2.1 Keyboard Level

Scientists observed every keystroke of the user to determine whether there are any intrusions happening. Lin (1997) used neural networks to analyze the keystrokes' duration and latencies. It is efficient to discriminate an illegal user, but it is not capable of detecting a legal user's misuse actions.

2.2.2 Low Level System Calls

Anderson (1980) pointed out that if a user with direct programming access operates at a level of control, he or she could bypass the auditing and access controls. Kuhn (1986) recommends monitoring system service calls, because it is harder to circumvent auditing on this level. Forrest et al. (1996) at the University of New Mexico chose to monitor behaviors at the level of privileged processes, which is defined as "processes that have privileges over and above those granted normal users" and run at the root in a UNIX system. The reasons for choosing privileged processes can be found in their paper (Forrest et al., 1996). They discovered that short sequences of system calls made by a program during its normal execution are very consistent. In this way, the IDS can set up a profile for normal usage of a program and detect intrusion by examining anomaly sequences in system calls. Researchers at Columbia University (Lee and Stolfo, 1997, 1998) applied a machine learning method on the same data and obtained satisfactory results.

2.2.3 Command Level

On this level, IDS monitors a sequence of commands submitted by the user, with other related statistics such as quantity of CPU, memory used, and quantity of input-output performed (Debar et al., 1992). Since intrusion behaviors (rules) are easily defined by sequences of commands, this level of auditing makes it easy for a security officer to get a "feel" for what intrusion has occurred. There are quite a few IDS monitoring command levels, such as Carrettoni et al. (1991), Debar et al. (1992), Lunt (1993a), Smaha and Winslow (1995), and Tan (1995).

2.2.4 Application Level

Information on each application is generally available on computers, consequently there are many measures on this level used by IDS. Login time, name of terminal used, CPU, memory and input/output usage, etc., all can be found in the system log files. In these systems (Bonifacio et al., 1997, Debar et al., 1992, Fox et al., Lunt, 1993a, 1990, Petersen, 1992, Smaha and Winslow, 1995, and Tan, 1995), these researches mentioned some or all of these measures. However, as Kuhn (1986) argued, users can write programs to access files directly without leaving any trace in the application audit logs. Merely auditing at this level will not detect all user activity.

Since every level has its own advantages and drawbacks, some IDS monitor data at several levels at the same time and correlate them to get new measures. NIDES, developed by SRI (Lunt, 1988), monitored data at three different levels: application program level, interface between user and the operating system, and the operating system kernel calls. The specific measures it analyzed can be found in the paper by Lunt (1988, 1993a).

2.3 Methods to Analyze System Calls

With the rapid growth in network systems, intrusions have become more common, their patterns more diverse, and their damage more severe. As a result, much effort has been devoted to the problem of detecting intrusions as quickly as possible, thus "dynamic" methods in Forrest et al. (1996).

Even if the auditing measures are selected correctly, there should be proper methods to analyze these data to get the results we want. As mentioned above, scientists have tried statistical methods, machine learning, genetic algorithm, neural network, and expert systems. Some are good at classifications, while others are good at learning patterns. It is difficult to say which method is best because good comparisons between them on a variety of data are needed.

2.3.1 TIDE and STIDE

As discovered by Forrest et al. (1996), in a realistic computing environment, short sequences of system calls executed by privileged processes in a networked operating system are used to define self. The strategy is to build up a database of normal behavior for each program of interest. Each database is specific to a particular architecture, software version, local administrative policies, and usage patterns. One host would have many different databases defining self, which means that the patterns comprising self are not uniformly distributed throughout the protected system. Once a stable database is constructed for a given program in a particular environment, the database can be used to monitor the program's behavior. The sequences of system calls form the set of normal patterns for the database and sequences not found in the database indicate anomalies.

After simulation, the results show that shot sequences of system calls do provide a compact signature for normal behavior and that the signature has a high probability of being perturbed during intrusions. Another appealing feature is code that runs frequently will be checked frequently. Thus, system resources are devoted to protection of the most relevant code segments. Also, privileged system calls provide a behavioral signature for a computer that is much harder to falsify than, for example, an IP address.

Forrest et al. (1996) compared their sequence time-delay embedding method with Hidden Markov Model (HMM) to analyze the short sequences of system calls. The result indicates that HMM gives the best accuracy on average, although at high computational costs. The much simpler statistical method provides results that are comparable with HMM.

2.3.2 Machine Learning Approach

Lee et al. (1998) employed a type of machine learning – RIPPER (Cohen 1995) to examine the same data set of system calls from the University of New Mexico. They first trained the system with both normal and abnormal sequences. After the system developed a couple of rules based on training, they used the rules to justify every sequence in a privileged program trace. Next, they used a sliding window with length 2L+1, moved forward by L, to examine whether there are more than L abnormal sequences in each window. For example, the following is a list of sequence results classified by the rules:

…normal, abnormal, abnormal, normal, normal, normal, abnormal,…

If L equals 3, then 7 will be the size of a window. In this window, there are 3 abnormal sequences. Thus, this region is a type of normal region. After examining the entire trace, there will be some regions normal, some abnormal. If the percentage of abnormal regions is higher than a threshold, let's say, 20%, the entire trace will be flagged as abnormal. They obtained a more significant threshold than Forrest et al. (1996). Also, since they did not have to collect all the normal sequences of system calls of a privileged process, they concluded that machine learning was more efficient for intrusion detection.

2.4 Summary

In the first part of this chapter, we surveyed various approaches to intrusion detection to provide the reader with a basis for evaluating the benefits, as well as the costs, associated with these approaches. The feature selection problem was raised after the survey because it is one of the key components of efficient detection systems. After conducting a comparison among the different levels of audit data and referring to previous research, we concluded the low-level system calls are appropriate audit features for intrusion detection systems. Then, we explained two studies on the system calls using different learning methods in detail.

CHAPTER 3

INTRUSION DETECTION TECHNOLOGY

In this chapter, we will review three artificial intelligence algorithms related to this experiment: Backpropagation net, C4.5, and Rough Sets. Backpropagation networks (Rumelhart et al. 1986) is a multi-layered, feed-forward neural networks; C4.5 is an inductive learning algorithm which is an extension to ID3 (Quinlan 1983); Rough Sets (Pawlak 1982) were proposed as a mathematical method to deal with uncertain and noisy data.

3.1 Neural Networks and Backpropagation Net

3.1.1 Overview

The foundation of the neural network paradigm was laid in the 1950s (Rosenblatt, 1962). Also called artificial neural networks, connectionist systems, or neuron-computers, neural network is "a parallel, distributed, dynamic information processing structure," designed to simulate the information processing in the human brain and obtain knowledge from examples feed to the network.

A neural network is composed of a number of simple arithmetic computing elements (or nodes, units) connected by links to form a network. Each link has a numeric weight associated with it. The nodes correspond to neurons—the cells that perform information processing in the brain—and the network as a whole corresponds to a collection of interconnected neurons.

To build a neural network to perform some tasks, one must first decide how many units should be used, what type of units are appropriate, and how the units should be connected to form a network. One then initializes the weights of the network and trains the weights using a learning algorithm applied to a set of training examples for the task.

Neural networks could be used for modeling mathematical relationships between input and output variables, predicting output values, classification, pattern recognition, and optimization. Neural networks have many desirable properties including strong mathematical foundation, fault tolerance, and adaptation to noisy data (Mooney et al., 1989). Hence, it has been successfully applied in many disciplines, including finance, marketing, operation management, image and sound processing.

3.1.2 Backpropagation Net

There are two broad categories of neural network structures, feed-forward and recurrent networks. In a feed-forward network, links are unidirectional. In a recurrent network, the links can form arbitrary topologies. Backpropagation networks is a popular multi-layered, feed-forward network that consists of input layer, output layer, and at least one hidden layer. Thus, some units are not directly connected with an outside environment. A typical network structure for Backpropagation network is illustrated in Figure3.1.

Backpropagation network is a supervised learning neural network; it receives both the raw data as input and the desired corresponding output. Typically, it starts out with a random set of weights and then updates its weights to make it consistent with the examples. This is done by making small adjustments in the weights to reduce the difference between the observed and predicted values. In this way, Backpropagation network can minimize prediction and classification errors incrementally until the network stabilizes. The updating process is divided into epochs (or cycles). Each epoch involves updating all the weights in the networks, starting with the output layer and propagating back to the previous layer until the earliest hidden layer is reached.

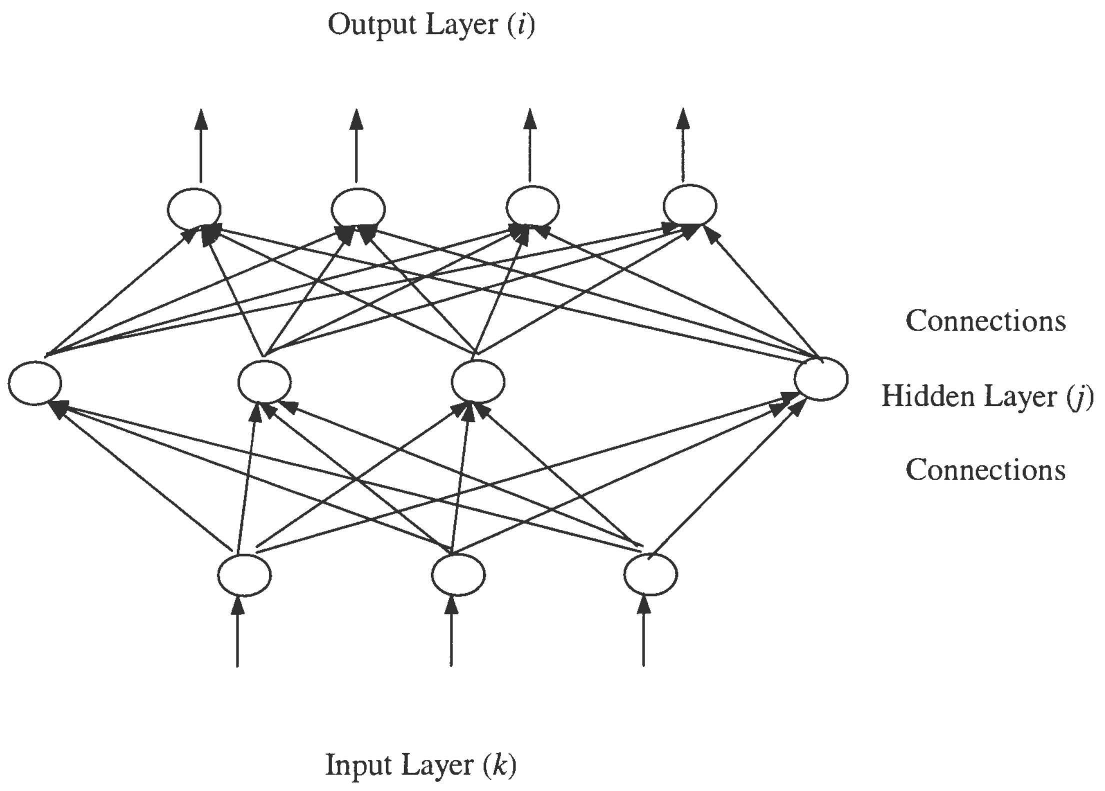

Figure 3.1: A Backpropagation network

Backpropagation operates in two phases. Each unit in the neural network has a set of input links from other units, a set of output links to other units, a current activation level, and a means of computing the activation level at the next step in time, given its inputs and weights. In the first operation called *forward-pass*, each unit performs a simple computation. It receives signals from its input links and computes a new activation level that it sends along each of its output links. The computation of the activation level is based on the values of each input signal and the weights on each input link. There are two components in the computation. The first is called input function, in_i, that computes the weighted sum of the unit's input values:

$$in_i = \sum_j W_{j,i} a_j ,$$

where $W_{j,i}$ is the weight on link *j* into node *i;* a_j is the input value of node *j.*

The second component is called the activation function, *g*, that transforms the weighted sum into the final value serving as the unit's activation value, a_i :

$$a_i \leftarrow g\,(in_i).$$

Usually all nodes in the network apply the same activation function.

In the second operation called *backward-pass*, differences between the desired and the actual outputs are propagated back through the network and are used to successively adjust the layers of the network, beginning with its output layer. If the predicted output for the single output unit *i* is O_i, and the correct output is T_i, then the error is given by $Err_i = T_i - O_i$. For Backpropagation networks, there are many weights connecting each input to an output, and each of these weights contributes to more than one output. Therefore, the trick is how to divide the error among the contributing weights. The weight update rule for the link from hidden unit *j* to output unit *i* is

$$W_{j,i} \leftarrow \quad W_{j,i} + \eta \times \ a_j \times Err_i \times g'\ (in_i),$$

where η is a constant called the leaning rate, and g' is the derivative of the activation function. $Err_i \times g'\ (in_i)$ can be simplified as Δ_i, which is called the error term of unit *i*.

The hidden node *j* is responsible for some fraction of the error Δ_i; this Δ_i value is divided according to the strength of the connection between the hidden node and the output node, and propagated back to provide the Δ_j value for the hidden layer. The propagation rule for the Δ_j value is the following:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i .$$

Now the weight updated rule for the weights between the inputs and the hidden layer is almost identical to the update rule for the output layer:

$$W_{k,j} \leftarrow W_{k,j} + \eta \times I_k \times \Delta_j ,$$

where $W_{k,j}$ is the weight of the link between input node *k* and hidden node *j*, and I_k is the value of the input node *k*. Therefore, the algorithm of updating the weights in each epoch can be summarized as follows:

- Compute the Δ values for the output units using the observed error.
- Start with the output layer and repeat the following for each layer in the network, until the earliest hidden layer is reached:

  1) propagate the Δ values back to the previous layer, and
  2) update the weights between the two layers.

In the real application of Backpropagation networks, through adapting the learning rate η to the local properties of the error surface, a quick and straightforward convergence of the gradient descent can be achieved. Oscillations in the proximity of a minimum can be avoided by introducing a *momentum factor* α, which accelerates the convergence in wide plateaus where the gradient is very small. By adjusting other network topology and learning parameters, such as the number of hidden layers and hidden neurons in each layer, the learning capability can be achieved to a very high level.

3.2 Rule Induction: ID3 and C4.5

3.2.1 Overview

Inductive learning is another popular machine learning technique for classification domain problems. It is simple, yet powerful, for the problem of classifying a set of variables into several discrete groups. Unlike neural networks, this method is able to extract a collection of decision rules from examples expressed on the IF-THEN-ELSE form. These kinds of rules are more meaningful and accessible to the users. Therefore, it is easier to maintain and to revise the database. It is also more convenient combine it with human experts' knowledge.

Iterative Dichotomizer 3 (ID3) algorithm has at its core a procedure based on Hunt's Concept Learning System (CLS) which, given a set of situations each described in terms of attributes and a class value, induces a situation classification rule (Quinlan, 1983). Quinlan adapted an information theory approach in the inductive algorithm where the measure of the information uncertainty associated with a certain system and the selection of dominant attributes are perceived as statistical processes. A detailed outline of the ID3 algorithm can be found in Quinlan (1983).

The ID3 algorithm constructs classification rules in the form of a decision tree recursively starting at the root. At each node, attribute A is selected to split the training data into those examples where $A = 0$ and where $A = 1$. This algorithm is then invoked recursively to these two subsets of training data, until all examples in one node belong to the same class. At this point, a leaf node is created and labeled as the expected value of the categorical attributes for the records described by the path from the root to that leaf.

The scheme used in decision tree learning for selecting attributes is designed to minimize the depth of the final tree. The idea is to pick the attribute that goes as far as possible toward providing an exact classification of the examples. All we need is a formal measure of "fairly good" and "really useless." The measure should have its maximum value when the attribute is perfect and its minimum value when the attribute is of no use at all. One suitable measure is the expected amount of information provided by the attribute. Information theory is given as follows: "The information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to the base 2 of the probability." Let S be a set of cases, and $freq(C_i,S)$ stands for the number of cases belonging to class C_i. The number of cases in the set S is denoted by |S|. If a case is selected at random from a set S of cases and is announce as belonging to some class C_i, this message will have the probability

$$\frac{freq(C_i, S)}{|S|}$$

and the information it passes on will be

$$-\log_2\left(\frac{freq(C_i, S)}{|S|}\right) \text{ bits.}$$

In general, if we are given a probability distribution S = $(p_1, p_2, ..., p_n)$, where

$$p_i = \frac{freq(C_i, S)}{|S|},$$

then the information conveyed by this distribution, also called the entropy of P, is

$$\text{Info (S)} = -\sum_{i=1}^{k} \frac{freq(C_i, S)}{|S|} * \log_2\left(\frac{freq(C_i, S)}{|S|}\right) \text{ bits.}$$

For example, if S is (0.5, 0.5) then Info (S) is 1, if S is (0.67, 0.33) then Info (S) is 0.92, if S is (1, 0) then Info (S) is 0. The higher the entropy of an attribute the more uncertainty there is with respect to its values.

When this is applied to a set T of training cases, info (T) gives the measure of the average amount of information needed to identify the class of a case in T. Considering a similar measurement after T has been partitioned in accordance with *n* outcomes of a decision test *X*, the expected information requirement can be found as the weighted sum over the subsets as

$$\text{info}_x(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \text{info}(T_i).$$

The quantity,

$$gain(X) = info(T) - info_x(T),$$

measures the information gained by partitioning *T* in accordance with the test *X*. This represents the difference between the information needed to identify an element of *T* and the information needed to identify an element of *T* after the value of attribute *X* has been obtained, that is, this is the gain in information due to attribute *X*.

The higher the information gain, the more informative the attribute *X*. We can use this notion of gain to rank attributes and to build decision trees where at each node is located the attribute with the greatest gain among the attributes not yet considered in the path from the root. Therefore, when a decision tree is being built, the root node of the tree would correspond to the attribute with the lowest entropy value and the highest gain value.

The intent of this ordering is two-fold:

1) To create small decision trees so that records can be identified after only a few questions.

2) To match a hopeful minimal process represented by the records being considered.

Although the gain criterion gives good results, it has a serious problem: there is a strong bias in favor of decision tests with a greater number of outcomes. This bias inherent in the gain criterion is rectified by a kind of normalization in which the apparent gain attributable to tests with a greater number of outcomes is adjusted. Consider the information content of a message with regard to a case that gives the outcome of the test rather than the class to which the case belongs. Analogous to the definition of *info (S),* it can be defined as

$$Split\ info\ (X) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2 \left(\frac{|T_i|}{|T|}\right),$$

where *split info (X)* represents the potential information generated by dividing *T* into *n* subsets, whereas the information *gain* measures the information relevant to the classification that arises from the same division. Then

$$gain\ ration\ (X) = \frac{gain(X)}{split\quad \inf o(X)}$$

expresses the proportion of information generated by the split that is useful, i.e., that appears helpful for classification.

3.2.2 C4.5

C4.5 is the direct descendent of ID3 and was developed by Quinlan as well. It accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on. Similar to ID3, C4.5 is able to create a classifier in the form of a decision tree consisting of leaves and decision nodes. The leaf indicates a class and the decision node contains some test to be carried out on a single attribute value. This will result in one branch and subtree for each possible outcome of the decision test. The program also contains heuristic methods for simplifying (pruning) decision trees, with the aim of producing more comprehensible structures without compromising accuracy on unseen cases.

In building a decision tree, C4.5 can deal with training sets that have records with unknown attribute values by evaluating the gain ratio for an attribute by considering only the records where that attribute is defined. In using a decision tree, C4.5 can classify records that have unknown attribute values by estimating the probability of the various possible results.

C4.5 can deal with the case of attributes with continuous ranges as follows. Say that attribute C_i has a continuous range. The training cases T are first sorted on the values of the attribute C_i. There are only a finite number of these values denoted in increasing order, A_1, A_2, ..., A_m. Then for each value A_j, j=1, 2,..., m, we partition the records into those that C_i have values up to and including A_j (e.g., A_1 to A_j) and those that have values greater than A_j (e.g., A_{j+1} to A_m). For each of these partitions we compute the gain ratio and choose the partition that maximizes the *gain.*

There is a very general phenomenon called overfitting for the decision tree learning algorithm. It happens when the algorithm uses the irrelevant attributes to make spurious distinctions among the examples, while the vital information is missing. Pruning of the decision tree is a way to treat the overfitting done by replacing a whole subtree by a leaf node. The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf. Start from the bottom of the tree and examine each nonleaf subtree. If the replacement of this subtree with a leaf, or with its most frequently used branch, were to lead to a lower predicted error rate, then the tree would be pruned accordingly, keeping in mind that the predicted error rate for this tree will be affected.

Since the error rate for the whole tree decreases as the error rate of any of its subtree is reduced, this process will lead to a tree whose predicted error rate is minimal with respect to the allowable forms of pruning. The additional computation invested in building parts of the tree that are subsequently discarded can be substantial, but this cost is offset against benefits due to a more thorough exploration of possible partitions. Growing and pruning trees is slower, but more reliable.

3.3 Rough Sets

3.3.1 Overview

Rough set, proposed by Pawlak in the early 1980s (Pawlak, 1982, Pawlak, 1991), is a mathematical approach used to deal with data analysis and knowledge discovery from imprecise and uncertain data. The starting point of this theory is that objects may be indiscernible in terms of the value of attributes. A rough set is a set of objects which cannot be precisely characterized based on the set of available attributes. In this case, any vague concept in the rough set is replaced by a pair of lower and upper approximations. These two approximations are two basic operations in the rough set theory.

3.3.2 Basic Concepts

3.3.2.1 Knowledge Representation System (KRS)

As data representation framework employed in the rough set theory, *knowledge representation system* is a finite table, with rows labeled as *objects*, columns labeled as *attributes,* and entries of the table called *attribute values*. It is also known as *information system*. A formal definition of KRS is given below.

Let S = < U, Q, V, f > be a KRS, where U is a non-empty, finite set of objects, the universal of data, e.g., $U = \{u_1, u_2, \ldots, u_n\}$. Q is the set of attributes, including a non-empty set of condition attributes C and a non-empty set of decision attributes D. We have $Q = C \cup D$ and $C \cap D = \Phi$. $V = \bigcup_{q \in Q} V_q$, where for each $q \in Q$, V_q is the domain of attribute q and the elements of V_q are called values of the attribute of q. f is the information function that assigns a unique value of the attribute q to each object $u_i \in U$.

3.3.2.2 Indiscernible Relation

Suppose P is a non-empty subset of Q, u_i and u_j are members of U. We can associate an approximation space in S by defining a binary indiscernible relation as follows:

$$\text{IND}(P) = \{ (u_i, u_j) \in U: \forall q \in P \quad f(u_i, q) = f(u_j, q) \}.$$

We say that u_i and u_j are indiscernible or equivalent by a set of condition attributes P in S IFF $\forall q \in P$, $f(u_i, q) = f(u_j, q)$. This indiscernible relation partitions U into several *elementary sets*. Each elementary set in IND (P) consists of a group of objects which has the same value of attributes; thus u_i and u_j are in one elementary set in terms of the attribute subset P.

3.3.2.3 Approximation of Set

Based on the concept of indiscernible relation, a universe U can be divided into several elementary sets by any subset of the attribute Q.

Suppose X is a non-empty subset of C. U is divided into $A = \{A_1, A_2, \ldots, A_i\}$ in terms of X and divided into $D = \{D_1, D_2, \ldots, D_j\}$ in terms of D. For each class D_n (n=1, 2, …, *j*) in D:

- If all objects in A_m (m=1, 2, …, *i*) are contained within D_n, we say A_m is in the positive region of D_n, that is POS (D_n).

- If no object in A_m (m=1, 2, ..., *i*) is contained within D_n, we say A_m is in the negative region of D_n, that is NEG (D_n).
- If some objects in A_m (m=1, 2, ..., *i*) are contained within D_n, we say A_m is in the boundary region of D_n, that is BND (D_n).

The *lower approximation* of set D_n, denoted by $\underline{X} D_n$, is the union of all A_m in the positive regions. The *upper approximation* of set D_n, denoted by $\overline{X} D_n$, is the union of all A_m in the positive regions and boundary regions.

We say that the lower approximation consists of all objects belonging to the concept D_n and the upper approximation consists all objects which possibly belong to the concept. Any set D_n (n=1, 2, ..., *j*) that has a non-empty boundary region is called rough, since it can be characterized only approximately.

3.3.2.4 Dependency of Attributes

The dependency of attributes is the relationship between condition attributes C and decision attributes D. Analysis of dependency is used to determine whether D can be characterized by the value of C. It is of primary importance in the rough set approach to discover data regularities for deriving rules.

The dependency of the decision attribute (D) on the condition attributes (C) equals the ratio of the number of objects in the positive regions to the number of objects in the universe U. It can range from 0 to 1:

- It is 1 when D can be uniquely discriminated with attributes in C.
- It is 0 when there is no relationship between C and D.

- It is between 0 and 1 when there is a partial dependency between C and D, i.e., only some objects can be classified into some categories of D based on the attributes in C.

3.3.2.5 Reduction of Attributes

Another important issue is the identification and elimination of redundant conditions. The objective is to find a subset of attributes that have the same discriminating power as the set of original attributes without losing any essential information. After all redundant attributes have been eliminated, the remaining subset of attributes is called a *minimal subset* or *reduct*.

Reduct R is defined as a minimal subset of attributes remaining the same degree of dependency between C and D, and none of the attributes in R can be further removed without affecting the degree of discerning power. For each reduct, we can derive a reduct table from the original knowledge representation system by removing those redundant attributes.

3.3.2.6 Decision Rule

3.3.2.6.1 Decision Rule Generation

Decision rules are generalized based on the non-redundant attributes contained in the chosen reduct. Values for these attributes are then analyzed to identify patterns in the data. The patterns are then expressed as logical statements which link the value of specific conditions with an outcome. A typical rule is:

If *conjunction of elementary conditions (Cond_C),*

Then *disjunction of elementary decisions (Dec_D).*

The elementary condition formulae over subset $X \subseteq C$ and domain V_{xi} of attribute $x_i \in X$ are defined as: $x_i = v_i$, where $v_i \in V_{xi}$. By Cond_C we denote a conjunction of elementary condition formulae, i.e. $(x_1 = v_1) \wedge (x_2 = v_2) \wedge \ldots \wedge (x_r = v_r)$ for all $x_i \in X$.

Similarly, we define elementary decision formulae $d = v_j$, where $v_j \in V_D$ by Dec_D. We denote a disjunction of elementary decision formulae, i.e., $(d = v_1) \wedge (d = v_2) \wedge \ldots \wedge (d = v_s)$. If s equals 1, there is only one elementary decision in Dec_D and the rule is certain rule. Otherwise, there are several elementary decisions in Dec_D, and the rule is a possible rule. This means available conditions Cond_C cannot exactly discriminate between elementary decisions.

The strength of decision rule is the number of objects satisfying Cond_C and belonging to one of the decisions in Dec_D. In possible rules, strength is calculated for each decision elementary separately.

The decision rules can be employed to analyzed new objects and partition them into different classes. If new object matches one possible rule, strength for all suggested decision classes in Dec_D in this rule will be assessed and the new object will be included into the class with the most strength.

3.3.2.6.2 Evaluation of Decision Rules

There are several measures used to evaluate the performance of decision rules.

1) Accuracy of decision rule

The accuracy of a rule with respect to a specific decision value can be defined as the ratio of the number of cases matching rule conditions and the decision to the number of cases matching rule conditions. Rules with high accuracy are usually desirable because the probability of an error is smaller.

2) Decision coverage

The decision coverage for a certain class equals the percentage of all cases with a matching rule decision and covered by rules for this decision to the number of cases with the decision value. An optimal set of rules is one which has acceptable values of rule strength, rule accuracy and decision coverage. However, the process of optimization may involve some trade off. To increase rule strength or decision coverage, one may have to decrease the accuracy of rules.

3.3.3 Advantages of Rough Set

The advantages and potential applications of the Rough Sets theory have been presented by Dimitras et al. (1999), Hashemi et al. (1998), Nagaraja (1997), Pawlak (1997), Pawlak et al. (1995), Slowinski et al. (1997), and Szladow and Ziarko (1993). It has been successfully applied for knowledge acquisition, forecasting and predictive modeling, decision support, etc.

- The main advantage is that it does not require any preliminary or additional information about the data.
- The method allows work with incomplete data without replacing missing values, to switch between different reducts, to use less expensive or alternative sets of measurements.
- It offers the ability to handle large amounts and various types of data.
- The knowledge of data is represented in an easy-to-understand logical format.

- Since the rules generated and attributes used are non-redundant, the patterns are very compact, strong, and robust.
- The ability to model highly non-linear or discontinuous functional relationships provides a powerful method for characterizing complex, multidimensional patterns.
- Rough sets can identify and characterize non-deterministic systems and incorporate probabilistic information and decision—making.

3.4 Summary

In this chapter, we reviewed three artificial intelligence techniques we would use in the experiment. In the past several years, there have been several studies, which compared the performances of either two of these three technologies or combined them as a hybrid system. Mooney et al. (1989) found that ID3 was faster than a Backpropagation net, but the Backpropagation net has stronger power to noisy data. Because of the rules created in the forms of decision trees, it is easier to explain the output values created by using rule induction methods. While to the Backpropagation net, it is relatively difficult to explain the output based on those weights between neurons. Dietterich et al. (1995) stated that the performance of these two techniques depends on the decoding method employed to the real-life data. They also discovered that Backpropagation net is rather time consuming to train.

CHAPTER 4

COMPARISON OF PERFORMANCES OF THREE METHODS

4.1 Preparation and Experiment

4.1.1 Data Description

The system call data set used in this experiment was from the immune system web site. It is for one privileged program – *sendmail*. The details of generating the *sendmail* traces used in this experiment were given in (Hofmeyr 1998) and on the immune system web site. The data includes several traces:

- Normal traces: a trace of the *sendmail* daemon and several invocations of the *sendmail* programs. During the period of collecting these traces, there were no intrusions or any suspicious activities happening.
- Abnormal traces: several traces either including intrusions or dangerous attempts. There are five error conditions of *forwarding loops*, three *sunsendmailcp* attacks, two traces of the *syslog-remote* attacks, two traces of the *syslog-local* attacks, two traces of the *decode* attacks, 1 trace of the *sm5x* attack attempt and one trace of the *sm565a* attack attempt. The descriptions of these intrusions can be found on the web site.

Each trace has one or more processes of *sendmail* programs, so each trace has two columns of data, the first one is the process ID indicating which process the system call belongs to and the second one is the system call value. In previous research, system calls were converted from string to integer values for ease of computing. Because there are totally 182 kinds of system calls considered, the values of system calls ranged from 0 to 181. The index about the numeric values and their related system call names can be also found on the web site. Table 4.1 shows a section of one *sendmail* trace. All the seven system calls belong

Table 4.1: An example of the *Sendmail* Trace.

	Values						
Process ID	1041	1041	1041	1041	1041	1041	1041
System Call Number	4	2	66	66	4	138	66
System Call [a]	write	fork	sstk	sstk	write	sethostid	sstk

[a] The system calls in last row will not appear in the data set.

to one process. In this sequence, “write” is the first system call, and “sstk” is the last one.

4.1.2 Data Preparation

Based on previous research, short sequences of system calls made by a program during its normal executions are very consistent and can be used for anomaly detection. Our purpose is to recognize the different patterns of normal and abnormal behaviors by using various learning algorithms. First, we need to set up these sequences from the original data sets. We have one system call and N-1 subsequent system calls in the same process compose one sequence of length N.

We suppose that all sequences of system calls from normal traces should be normal sequences, while those from suspicious traces which cannot be found in the normal traces are abnormal sequences. Using a sliding window of length N, we can search through all the traces and set up two data sets, one consists of normal sequences and another consists of abnormal ones. In previous research (Hofmeyr et al., 1998), the proper length of sequence tested varied from 2 to 30. The conclusion of the research was that the length of the sequence should be 6 or slightly larger than 6 so as to allow detection of anomalies while minimizing computation. The length of the sliding window in our experiment is 7 for ease of comparison with previous results.

- Normal Sequence Sets: all the sequences are selected from the normal traces and unique from each other.
- Abnormal Sequence Sets: whenever a unique sequence is obtained from the abnormal traces, it will be compared with all the sequences in the normal data set first. If it does not match any of those normal sequences, it will be added to the abnormal sequence set.

Finally, we got 1112 normal sequences (patterns) and 1576 abnormal sequences. The detailed number of abnormal sequences for each kind of trace is listed in Table 4.2.

4.1.3 Training and Testing Data Structure

In this part of the experiment, there are two sets of training and testing data sets listed in Table 3. One has the same structure as that used by Lee and Stolfo (1998). The training data consist of abnormal sequences from four suspicious traces, e.g., 2 *sunsendmailcp*, 1 *syslog-remote, and* 1 *syslog-local,* and 80% of the normal sequences. The testing data include the remaining 20% of the normal sequences and all the abnormal sequences which are not in the training data set except the *forwarding loop* traces (e.g., 1 *sunsendmailcp*, 1 *syslog-remote*, 1 *syslog-local*, 2 *decode*, 1 *sm565a*, and 1 *sm5x).*

For the second kind of data set, we used the same amount of normal and abnormal sequences in the training data set and made a three_fold cross validation. First, we randomly split the normal data set into three parts. Each time, we put two parts in the training set and combined them with the same number of abnormal sequences. The remaining normal data were placed in the testing set, combined with 300 abnormal sequences different from those in the training set.

Table 4.2: Amounts of the sequences of system calls in different traces.

Traces	# of Sequences	
Normal	1112	
Total		1112
Abnormal		
Decode	16	
Forwarding loops	258	
Sunsendmailcp	219	
Syslog-local	359	
Syslog-remote	439	
Sm565a	23	
Sm5x	262	
Total		1576

Table 4.3: Summary of the structure of the data sets.

No. of Combination	Training		Testing		Total
	Normal	Abnormal	Normal	Abnormal	
1	902	594	210	724	2430
2	742	742	370	300	2154

4.1.4 BP Experiment

4.1.4.1 Software Package

PCNeuron, a neural network development shell developed by Professor I-Cheng Yeh, was used in this experiment. It runs in the DOS environment. There are several neural networks learning algorithms included in this package. Backpropagation nets (BP) is the learning method employed in our research.

4.1.4.2 Input Neuron Representation

In the lookup table of the system call names, a number ranging from 0 to 181 represented each system call. So, in each sequence of system calls, the value of each unit ranged from 0 to 181. Considering that in BP, the range of individual input neuron value affects the prediction errors greatly, all the integer values of system calls were converted to binary format as a sequence of 0 and 1. There were only 53 kinds of system calls appearing in the *sendmail* program traces, the length of the sequence for each system call was 6 and each pattern used in BP had 42 bits in length. The lookup table for the conversion of integer values to binary bits is found in Appendix.

4.1.4.3 Output Neuron Representation

Because all the patterns are classified into two categories, e.g., normal and abnormal, the number of output neurons is 2. Normal sequence is represented in “0 1” and abnormal is “1 0.”

4.1.4.4 Hidden Layer and Hidden Neuron

The number of hidden units in the hidden layers as well as the number of hidden layers in BP is harder to determine. They provide the added power of internal representation, which could capture the non-linear relationships between the input and the output vectors.

Researchers have developed different heuristics for topology selection (Rumelhart et al. 1986). They generally agree that a larger hidden layer results in a more powerful network, but too much power obtained from the training data may be undesirable for predicting new, unknown data. In addition, the computation time needed for BP is directly proportional to the number of hidden units and the number of hidden layers in a network.

As a general rule, one hidden layer is effective enough for the simple classification domain problem. In our experiment, one and two hidden layers were both tested. The proper number of hidden neurons was usually obtained through extensive trial-and-error tests. In most cases, it equaled half of the sum of the input and output neurons, e.g., 22 in our case. Alternatively, the number could be 2/3 of the input neurons, e.g., 28 in our case. We used various numbers of hidden units ranging from 12 to 38 in our experiment.

4.1.4.5 Learning Parameters

4.1.4.5.1 Learning Rate

This parameter determines how far to move in the direction of the gradient of the surface over the weight space defined by an error function. A small learning rate will lead to slower learning; while a large one may cause a move through weight space that ignores the solution vector. In the package, the learning rate ranges from 0.1 to 10. We found that the default value worked very well.

4.1.4.5.2 Momentum Factor

This parameter is used to keep the weight changes moving in the same direction and allow the algorithm to skip over the small local minimum. In addition, it can speed up the learning step. However, a large momentum factor will cause the side effect of skipping too much and quickly as the large learning rate does. In past studies, the value of momentum factor was between 0 and 1. We tested various values ranging from 0.1 to 1 and found that the default value in most of the experiments worked very well.

4.1.4.5.3 Learning Cycles (epochs)

A large number of epochs usually brings the network closer to a convergent state and improves the recall accuracy for the training data set. For different applications, the required number of epochs can be very different, ranging from a few hundred to several thousand, or even more. The learning should stop at a proper number of epochs when the network stabilizes for a fairly long time and converges to a lowest error level. Too many epochs will cause the error rate to bump up. For this application, 2000 cycles were far enough to get a satisfactory result. We employed different cycles ranging from 200 to 10,000 in the experiment.

4.1.4.6 Affects of Parameters

Among the parameters we tested, the number of hidden neurons affected the learning performance the most. Figure 4.1 is a summary of the learning error rates when different numbers of hidden neurons were used with a learning rate 2, the momentum factor 0.5, and the hidden layer 1. This time the appropriate number of hidden neurons does not go with the general rule mentioned above. The lowest training and testing error rates were obtained when there are 35 hidden neurons.

4.1.5 C4.5 Experiment

4.1.5.1 Software Package

SIPINA-W version v2.5, a software for knowledge discovery in databases on windows platform, was used in this experiment. This version was developed by Zighed et al. (1992).

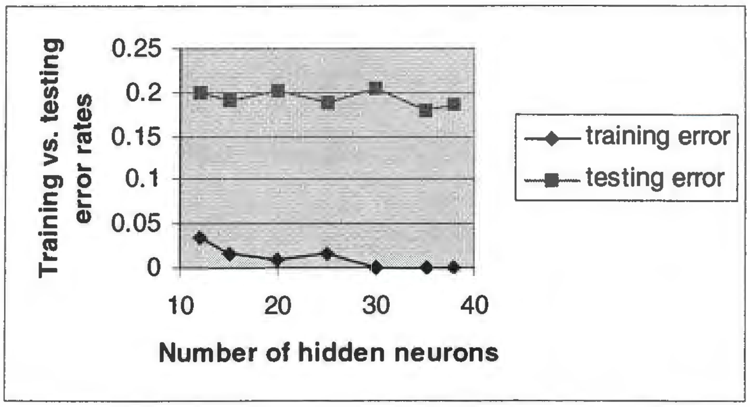

Figure 4.1: Effect of the number of hidden neuron on training and testing error rates.

It contained several learning methods: CART, ID3, C4.5, ELISEE, Chi2Aid, and Sipina. C4.5 was the learning method employed in our research. Both quantitative and qualitative data were accepted by this package. In this program, decision graphs and trees can be transformed into productions rules and stored in a Knowledge Base System (KBS), and non-learned examples can be tested using this knowledge.

4.1.5.2 Input Unit Representation

There are two kinds of representation methods in this part.

1) The original integer values of the system calls are used directly in the training system. Due to the limitation of this package, qualitative variables can have 10 values at most, all the input units are defined as continuous variables with the minimum at 1 and maximum at 167. The number of input units is 7.
2) Same as the representation method used in BP, each system call is changed to 6 units valued at 0 or 1. The number of input units is 42 instead of 7.

4.1.5.3 Output Unit Representation

Normal sequence is classified as 1. Abnormal sequence is classified as 2.

4.1.6 Rough Sets Experiment

4.1.6.1 Software Package

ROSETTA is a toolkit for analyzing tabular data within the framework of Rough Sets theory. It was designed and implemented in the Department of Computer and Information Science at the Norwegian University of Science and Technology. ROSETTA is designed to support the overall data mining and knowledge discovery process from initial browsing and preprocessing of the data, via computation of minimal attribute sets and generation of if-then rules or descriptive patterns, to validation and analysis of the induced rules or patterns.

4.1.6.2 Input Units Representation

Similar to the situation in C4.5, there are two kinds of representation methods in this part.

1) The original integer values of the system calls are used directly in the training system, but they are considered as qualitative attributes instead of real quantitative numbers. The number of input units in each pattern is 7.
2) Same as representation method in BP, each system call is changed to 6 bits valued at 0 or 1. The number of input units is 42 instead of 7.

4.1.6.3 Output Unit Representation

Normal sequence is classified as 1. Abnormal sequence is classified as 2.

4.1.6.4 Reduction Algorithm

We used the Genetic Algorithm as the deduction algorithm.

4.2 Result and Discussion

To understand how well the detection methods perform on a variety of data, we first averaged the results across all the data sets. Table 4.4 shows the average accuracy rates for each combination of data encoding method and structure of data sets. The columns represent

Table 4.4: Summary of the accuracy rates for different learning methods.

	Binary Encoding			Integer Encoding	
	BP	C4.5	Rough Sets	C4.5	Rough Sets
1484 training vs. 670 testing					
Training Accuracy					
	0.9962	0.84	1.0000	0.80	0.9900
Testing Accuracy					
	0.8106	0.65	0.8483	0.59	0.8050
1496 training vs. 934 testing					
Training Accuracy					
	0.9800	0.79	0.9900	0.76	0.9800
Testing Accuracy					
	0.6720	0.62	0.7092	0.55	0.6513

the different representation methods. Each representation method is further subdivided into three or two columns depending on the methods employed for this kind of encoding data. The rows represent the different combinations of normal and abnormal sequences. Then, each row is further subdivided into two rows for the training and testing accuracy rates. This table is only a rough outline of the results. In the following parts, we will compare the relationship between the accuracy and data combination or data encoding method in more detail.

4.2.1 Accuracy Rates vs. Data Combinations

In Figures 4.2 and 4.3, the combination of 1484 training data and 670 testing data has a better accuracy rate in all the three methods. There are two differences in the data compositions between the two combinations, which could explain the differences in accuracy rate. In the first data set, we used equal amounts of normal and abnormal sequences in the training set, while they are different from the second set. Therefore, we can conclude that the percentage of different classes of data in the training set has an impact on the learning result. The more data of one class in the training set, the more easily it is recognized correctly by the classifier.

The second difference is the source of abnormal sequences in the training set. In the first training set, the source of abnormal sequences includes all the intrusion traces available on the web. On the other hand, there are only four traces of abnormal sequences in the second

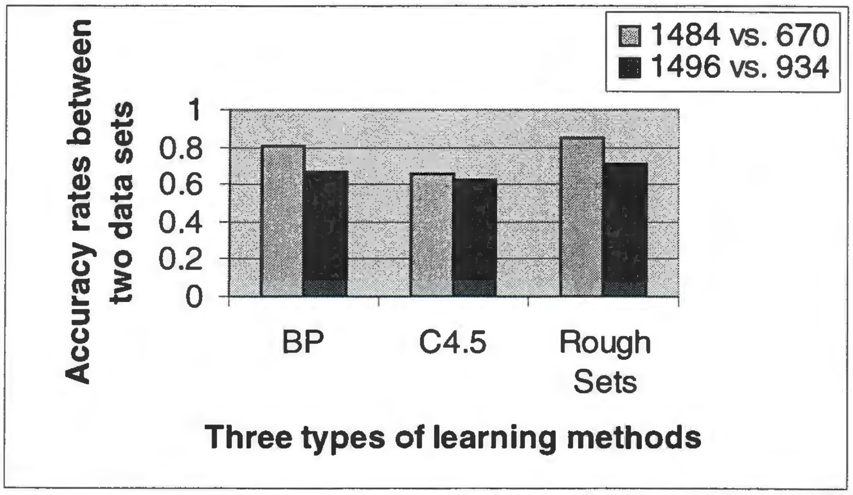

Figure 4.2: Comparison of the testing accuracy rates of three learning methods between two combinations of data. The input units are represented in binary format.

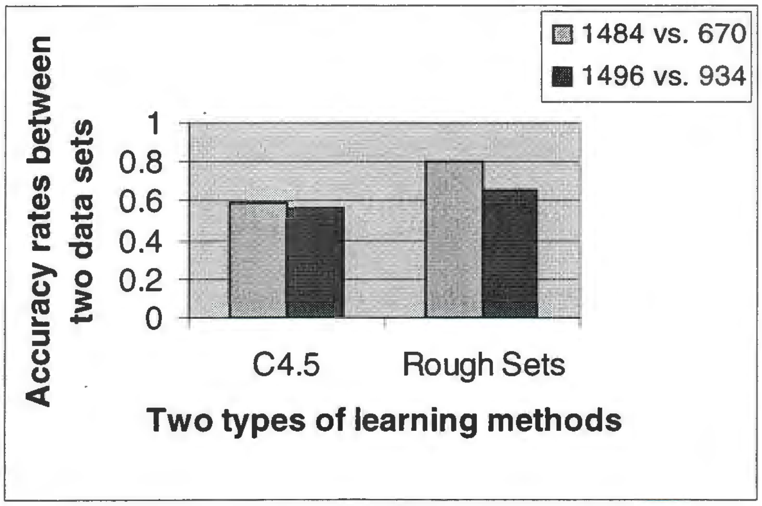

Figure 4.3: Comparison of the testing accuracy rates of three learning methods between two combinations of data. The input units are represented in integer format.

training set. For all the classifiers, an adequate amount of training samples is required to get satisfactory performance. The lower accuracy rate in the second training set could be attributed to the possibility of losing some important abnormal patterns.

4.2.2 Accuracy Rates vs. Data Representation

Now we compare the accuracy rates of the first data set, e.g., 1484 training data and 670 testing data between the two input representation methods.

From Figure 4.4, we find that no matter what kind of learning method we use, higher accuracy is always obtained when the input units are in a binary format. Although the integer system calls are declared as qualitative variables in Rough Sets and Rough Sets are capable of handling large numbers of discrete data, the testing accuracy rate is lower than that in the binary format. In this regard, the binary format is a better representation method for qualitative attributes.

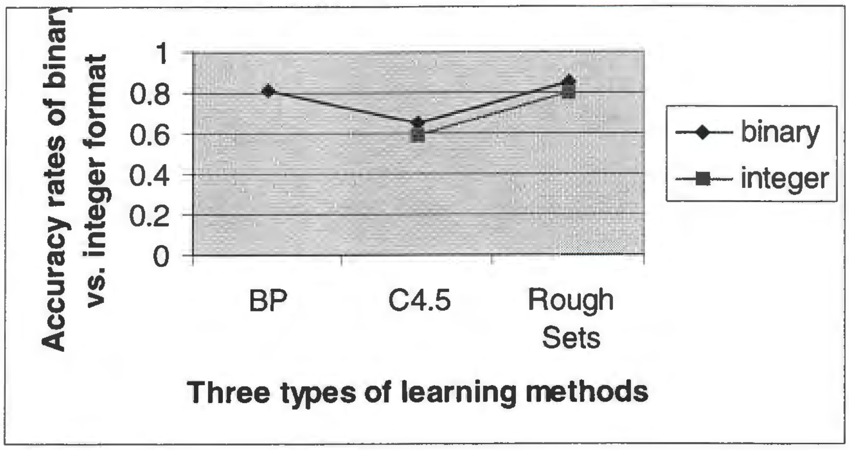

Figure 4.4: Comparison of the testing accuracy rates of three learning methods between two types of input representation methods.

4.2.3 Overall Prediction Performance

After evaluating the effect of training data combination and representation methods, we compared the prediction performance for each classification method and found that Rough Sets was generally better than the other two methods in terms of testing accuracy. Backpropagation obtained the second highest accuracy, and C4.5 was the worst. These results are further supported by statistical analysis in Table 4.5. We used the training data set with 1484 patterns in binary format because it had the highest accuracy rate in all the three methods. The testing data set included 14 whole *sendmail* traces— 3 normal traces, 2 *decode*, 3 *sunsendmailcp*, 2 *syslocal*, 2 *sysremote*, 1 *sm5x*, and 1 *sm565a* trace.

Selecting the best accuracy rates from these three methods, the time needed to train the learning systems is very different from each other. The system calls represented in binary format take more time for training than integer format. For the binary encoding method, Rough Sets took the longest time to complete the rule generation process, two hours and 12

Table 4.5 : Statistic testing among BP, C4.5, and Rough Sets [a]

(a) Paired T-test of testing accuracy between BP and C4.5

	BP	C4.5
Mean	0.841699	0.781429
Variance	0.000638	0.00069
Observations	14	14
Pooled Variance	0.000664	
Hypothesized Mean Difference	0	
Df	26	
t Stat	6.187322	
P(T<=t) two-tail	1.52E-06	

(b) Paired T-test of testing accuracy between Rough Sets and C4.5

	C4.5	Rough Sets
Mean	0.781429	0.90485
Variance	0.00069	0.000425
Observations	14	14
Pooled Variance	0.000557	
Hypothesized Mean Difference	0	
Df	26	
t Stat	-13.83105	
P(T<=t) two-tail	1.69354E-13	

(c) Paired T-test of testing accuracy between BP and Rough Sets

	BP	Rough Sets
Mean	0.841699	0.90485
Variance	0.000638	0.000425
Observations	14	14
Pooled Variance	0.000531	
Hypothesized Mean Difference	0	
Df	26	
t Stat	-7.24741	
P(T<=t) two-tail	1.07E-07	

[a] α=0.05

minutes compared to 30 minutes for BP, and 5 minutes for C4.5. For integer encoding method, Rough Sets are still the most time-consuming method, taking about 10 minutes to complete the training, while C4.5 needed only one minute and 20 seconds.

The size of the rule sets are quite different for C4.5 and Rough Sets as well. While there are more than ten thousand rules in Rough Sets, there are a dozen rules for C4.5. Considering the prediction performances of the two methods, we can hypothesize that a larger rule set leads to better performance.

BP has a similar accuracy rate as Rough Sets, but achieves it in much less time. This is an advantage to Rough Sets. On the other hand, since BP learns the patterns by updating the weights of neurons and the weights were initialized randomly, it is hard to tell the sequence patterns in a readable format after training.

4.2.4 True Positive and False Positive

When intrusion detection system is used in anomaly detection, its object is to differentiate that suspicious phenomenon from normal user behavior. Users do not care about individual sequence's class, but the whole trace's class (Lee and Stolfo, 1998). Therefore, besides the testing accuracy for each sequence, we used the concepts of true positive and false positive to measure the performance of different learning methods. Low false alarm and high efficiency are two desirable properties for a good intrusion detection system. Low false positive shows the ability to keep a low false alarm, and true positive rate measures the efficiency, the overall capability of detecting anomalies. The system has to minimize the false positives and maximize the true positives.

To detect an intrusion, the system has a locality frame count (LFC), which checkes whether the current sequence is anomalous, and keeps track of the number of anomalous ones in the last 20 sequences (Warrender et al., 1999). There is a preset threshold, valued from 1 to 20 on the LFC, below which traces are still considered to be normal. Any time the LFC reaches or exceeds the threshold, an anomaly is recorded, and the trace is considered as an intrusion. The true positive rate is either 1 or 0 according to the result of this process. For an intrusion trace, if it is recorded as an anomaly, the true positive rate is 1, otherwise it is 0.

The false positive rates are reported for normal traces. As we discussed in a previous chapter, all sequences in a normal trace should be normal. After testing with the classifier, a few sequences are mis-classified as suspicious. For each sequence in this category, we checked the last 20 sequences and counted the LFC. There is also a threshold in LFC ranging from 1 to 20, below which the current sequence was still considered as good. After scanning the whole trace, we calculated the false positive rate as a fraction of the total number of sequences. The false positive rate should be between 0 and 1.

We used the training data set with 1484 patterns in binary format because it had the highest accuracy rate in all the three methods. The testing data set included 14 whole *sendmail* traces— 3 normal traces, 2 *decode*, 3 *sunsendmailcp*, 2 *syslocal*, 2 *sysremote*, 1 *sm5x*, and 1 *sm565a* trace.

To get a picture of how well the detection methods perform on a variety of data sets, we first average the results across all the data sets. Figure 4.5 shows the true positives for each combination of learning method and sensitivity threshold. A different symbol is used to denote each method, and each point shows performance at a particular threshold. We find that Backpropagation net has an overall higher true positive across all the thresholds.

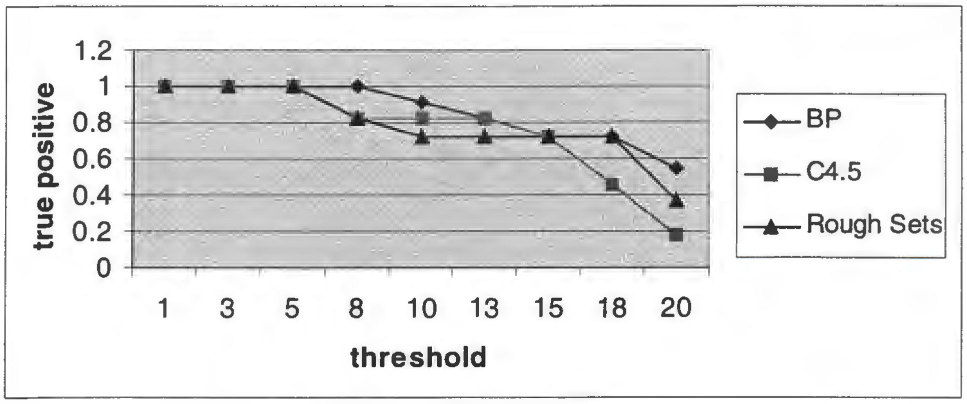

Figure 4.5: Average true positives vs. threshold for each method.

Figure 4.6 shows the false positives for each combination of learning method and sensitivity threshold. This time Rough Sets get much less false positives across all the thresholds.

False negatives can be reduced by additional layers of defense since the probability that an intruder will penetrate all layers undetected is $(P_n)^L$, where L is the number of layers and P_n is the probability of escape for each layer (Hofmeyr et al., 1997). Since false negatives are

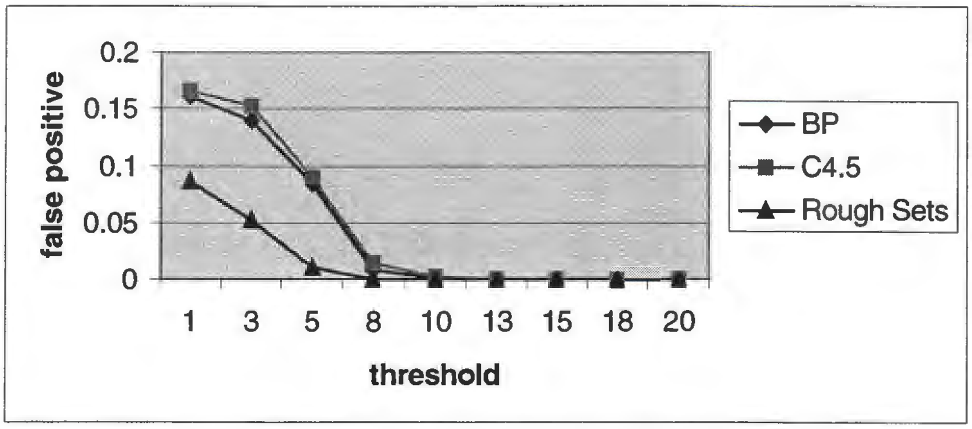

Figure 4.6: Average false positives vs. threshold for each method.

opposite to true positives, we can improve the true positive by adding layers of defense. By contrast, more layering will compound the false positives in the rate of $(P_f)L$, where P_f is the probability of generating a false positive in each layer. Therefore, low false positive is more critical for a successful intrusion detection system. Rough Sets have the lowest false positives and comparable true positives. In this case, it appears Rough Sets have the best performance for our data.

These two figures also show the relationship between the thresholds and true or false positives. As the threshold increases, fewer sequences are identified as anomalous in all traces, which affect both true and false positives. Lowering threshold improves the true positives and increases the opportunity of false alarms as well. Thus, the appropriate threshold is another element we need to adjust for balancing between maximized true positives and minimized false positives. From this result, we can determine that 5 is a proper threshold for Rough Sets, and 8 is appropriate for Backpropagation and C4.5. However, more testing traces would improve our confidences because the number of data sets we used is relatively small. Each individual data set carries too much weight in the final outcome, and adding one trace might change the results enough to favor other methods.

4.3 Summary

In this chapter we compared three methods for characterizing normal behavior and detecting intrusions based on system calls in *sendmail* privileged processes. There are two kinds of encoding methods and two combinations of the training data sets used for each learning method. On the test suite, Rough Sets gave the best accuracy on the average, although at high computational costs. Also, the binary encoding method better represents the system calls for the higher accuracy. However, the time consumed for training increases along with an increase in the number of attributes in each record. Besides the prediction accuracy for each sequence, we care more about what extent the learning method we used can detect the intrusion as well as keeping a low false alarm. Therefore, we employed the concepts of true positives and false positives after the classification. The performances of different methods not only rely on the prediction accuracy, but also on the threshold value we used.

CHAPTER 5

CONCLUSIONS

5.1 Summary

In this thesis, we first surveyed the existing approaches and research to intrusion detection problems. We discussed in detail about two issues. One issue is the technique used to analyze the audit data, mainly in anomaly and misuse ways. The other issue is the appropriate set of features selected for analysis. Then, we briefly summarized three techniques we used in this experiment. After that, we conducted a series of experiments to evaluate the classification performances between C4.5, Rough Sets, and Backpropagation net to the sequences of system calls. The measures include the prediction accuracy, false positive and negative rates, and the cost of prediction. We did not attempt to measure performance in terms of system usage or cost of time, although we do make some general observations about computational effort. After comparing the results, we conclude that Rough Sets have the best performance among these three methods. The key issues considered in this experiment are the data representation, the topology, and the parameters selected in C4.5, Rough Sets, and Backpropagation.

5.2 Contribution

We generalize our findings and contributions as the following:

1) Rough Sets has a better performance than the Backpropagation net and C4.5 for the system call data. It gets a more accurate rate in classification and pattern recognition. The result that Backpropagation net classifies more accurately than C4.5 verifies previous studies, which compared the performances between these two techniques with other applications. Although the time for training the neural network is longer than that needed

for C4.5, the testing time is almost the same. Because people only care about how fast a well-trained intrusion detection system can analyze new data in real life, longer training time is not a big issue at this point.

2) We determine that binary format is a better way to represent the qualitative variables.

5.3 Directions of Future Works

After the experiment, we got some expected results. However, there is still work needed to improve the classifier's performance.

1) Studies can always be conducted more thoroughly. As with training data, what we used is obtained from the web site. For all the learning algorithms, a large amount of training data is a fundamental component to high performance. Each method has a number of parameters that affect classification accuracy, but only a few representative variations were investigated here.

2) The hybrid system is a possible solution to improve the classification accuracy. The rule sets created by Rough Sets are good sources for feature selection. First, we can select some meaningful rules from the rule sets, determine the important attributes, remove the non-relevant attributes from the data sets, then use the modified data to train the classifiers. The classifiers are not restricted to Rough Sets; we can still use C4.5 or neural nets at this step.

3) Although short sequences of system calls proved to be an efficient way to represent the normal behavior, there are still many other features available in the audit trails or other resources. Previous researchers using those features also gave us some promising results. We should consider including other features into our data set as well.

4) The data used in this experiment were collected at the SunOS 4.1.1 and 4.1.4 workstations. As long as the development of the Linux operating system, there will be more people caring about its security problems. Therefore, if we can analyze the data collected from this operating system, it will be more helpful to future researchers.

APPENDIX

Binary representation of system calls in *sendmail* program [a]

System calls	Integer value	Binary value	System calls	Integer value	Binary value
fork	1	000001	getgroups	74	011100
read	2	000010	setgroups	75	011101
write	3	000011	setitimer	78	011110
open	4	000100	gethostname	83	011111
close	5	000101	getdtablesize	85	100000
wait4	6	000110	fcntl	88	100001
create	7	000111	select	89	100010
link	8	001000	socket	93	100011
unlink	9	001001	connect	94	100100
chdir	11	001010	accept	95	100101
chmod	14	001011	bind	100	100110
stat	17	001100	setsockopt	101	100111
lseek	18	001101	listen	102	101000
getpid	19	001110	sigvec	104	101001
getuid	23	001111	sigblock	105	101010
fstat	27	010000	sigsetmask	106	101011
access	32	010001	sigstack	108	101100
setpgrp	38	010010	gettimeofday	112	101101
dup	40	010011	recvfrom	121	101110
pipe	41	010100	setreuid	122	101111
getgid	45	010101	setregid	123	110000
ioctl	50	010110	rename	124	110001
readlink	54	010111	sendto	128	110010
umask	56	011000	getrlimit	138	110011
getpagesize	59	011001	getdomainname	155	110100
vfork	61	011010	getdents	167	110101
mmap	66	011011			

[a] The system call names and related integer values are collected from the web page of computer immune system developed by Forrest et al. The address is http://www.cs.unm.edu/~immsec/data/synth-sm.html

REFERENCES

Anderson J. P., *Computer Security Threat Monitoring and Surveillance*, Technical Report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.

Bonifacio J. M. Jr., Cansian A. M., Carvalho A. C. P. L. F., and Moreira E. S., Neural Networks Applied in Intrusion Detection Systems. *Proceedings of the International Conference on Computational Intelligence and Multimedia Application.* Gold Coast, Australia, 276-280, February 1997.

Carrettoni F., Castano S., Martella G., and Samarati P., RETISS: A Real Time Security System for Threat Detection Using Fuzzy Logic. *Proceedings of 25th IEEE International Carnahan Conference on Security Technology,* Taipei, Taiwai ROC, October 1991.

Chin S. K., High-confidence Design for Security, *Communications of the ACM*, 33-37, 42, 7, July 1999.

Cohen W. W., Fast Effective Rule Induction, *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Tahoe, CA, 1995.

Debar H., Becker M., and Siboni D., A Neural Network Component for an Intrusion Detection System. *Proceeding of the 1992 IEEE Symposium on Research in Security and Privacy*. 240-250, Oakland, CA, May 1992.

Dietterich T. G., Hild H., and Bakiri G., A Comparison of ID3 and Backpropagation for English Text-to-speech Mapping. *Machine Learning*, 51-80, 18(1), 1995.

Dimitras A.I., Slowinski R., Susmaga R., and Zopounidis C., Business Failure Prediction Using Rough Sets. *European Journal of Operational Research*, 114, 263-280, 1999.

Durst R., Champion T., Witten B., Miller E., and Spagnuolo L., Testing and Evaluating Computer Intrusion Detection Systems, *Communications of the ACM*, 53-61, 42, 7, July 1999.

Farmer D. and Spafford E., The COPS Security Checker System. *Proceedings of the Summer USENIX Conference*, 165-190, 1990.

Fisher D. H. and McKusick K. B., An Empirical Comparison of ID3 and Back-propagation. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 788-793. Detroit, MI. August 20-25, 1989.

Forrest S., Hofmeyr S. A., Somayaji A., and Longstaff T. A., A Sense of Self for UNIX Processes. *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, 120-128. 1996.

Forrest S., Jovornik B., Smith R. E., and Perelson A. S., Using Genetic Algorithms to Explore Pattern Recognition in the Immune System. *Evolutionary Computation,* 1, 3, 191-211, 1993.

Fox K. L., Henning R. R., Reed J. H., and Simonian R. P., *A Neural Network Approach Towards Intrusion Detection.* Harris Corporation. July 1990.

Fürnkranz J. and Widmer G., Incremental Reduced Error Pruning. *Machine Learning: Proceedings of the Eleventh Annual Conference.* New Brunswick, New Jersey. 1994.

Garvey T. D. and Lunt T. F., Model Based Intrusion Detection. *Proceedings of the 14th National Computer Security Conference.* 372-385, October 1991.

Hashemi R.R., Le Blanc L.A., Rucks C.T., and Rajaratnam, A., A Hybrid Intelligent System for Predicting Bank Holding Structures. *European Journal of Operational Research.* 109, 390-402, 1998.

Heberlein L. T., Dias G. V., Levitt K. N., Mukherjee B., Wood J., and Wolber D., A Network Security Monitor. *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy.* 296-304, May 1990.

Hofmeyr S. A. and Forrest S., Immunity by Design: An Artificial Immune System. *Proceedings of Genetic and Evolutionary Computation Conference (GECCO '99).* July 1999.

Hofmeyr S. A., Forrest S., and Somayaji A., Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security,* 6, 151-180, 1998.

Hofmeyr S. A., Forrest S., and Somayaji A., Lightweight Intrusion Detection for Networked Operating Systems. *http: //www.cs.unm.edu/ ~forrest/ papers.html.* 1997.

Holland J. H., *Adaptation in Natural and Artificial Systems*, Technical Report, University of Michigan, Ann Arbor, 1975.

Ilgun K., USTAT: A Real-time Intrusion Detection System for UNIX. *Proceedings of IEEE Symposium of Research in Security and Privacy.* 1993.

Keiss S. M., and Kulilowski C. A., *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning, and Expert Systems.* Morgan Kaufmann Publishers, Inc. San Mateo, CA, 1991.

Kim J. and Bentley P., The Human Immune System and Network Intrusion Detection. *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT '99).* Aachen, Germany, September 13- 19, 1999a.

Kim J. and Bentley P., The Artificial Immune Model for Network Intrusion Detection. *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, September 13- 19, 1999b.

Kim J. and Bentley P., Negative Selection and Niching by an Artificial Immune System for Network Intrusion Detection. *A late-breaking paper submitted to Genetic and Evolutionary Computation Conference (GECCO '99).* July 1999c.

Kuhn J. D., Research toward Intrusion Detection through the Automated Abstraction of Audit Data. *Proceedings of the 9th National Computer Security Conference.* September 1986.

Lee W. and Stolfo S. J., Learning Patterns from Unix Process Execution Traces for Intrusion Detection. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.

Lee W. and Stolfo S. J., Data Mining Approaches for Intrusion Detection. *Proceedings of the Seventh USENIX Security Symposium (SECURITY '98).* San Antonio, TX, January 1998.

Lin D. W., Computer Access Authentication with Neural Network Based Keystroke Identity Verification. *Proceedings of IEEE Symposium of Research on Security and Privacy.* 1997.

Lunt T. F., Tamaru A., Gilham F., Jagannathan R., Neumann P. G., Javitz H. S., Valdes A., and Garvey T. D., *A Real-time Intrusion Detection Expert System (IDES).* Technical Report, Computer Science Laboratory, SRI International. Menlo Park, CA. February 1992.

Lunt T. F., Automated Audit Trail Analysis and Intrusion Detection: A Survey. *Proceedings of 11th National Computer Security Conference*. 1988.

Lunt T. F., A Survey of Intrusion Detection Techniques. *Computer & Security*. 12, 1993a.

Lunt T. F., Detecting Intrusion in Computer Systems. *Proceedings of Conference on Auditing and Computer Technology*. 1993b.

Mé L., GASSATA—a Genetic Algorithm as an Alternative Tool For Security Audit Trails Analysis, *First international workshop on the Recent Advances in Intrusion Detection (RAID98)*. Louvain-la-Neuve, Belgium, September 14-16, 1998.

Mooney R., Shavlik J., Towell G., and Gove A., An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. 775-780. Detroit, MI. August 1989.

Nagaraja S., Bridge Deck Rebar-corrosion Knowledge Based Decision System Development Using Machine Learning Techniques. *Thesis*, Kansas State University. 1997.

Pawlak Z., Rough Set Approach to Knowledge-based Decision Support. *European Journal of Operational Research*, 99, 48-57, 1997.

Pawlak Z., Grzymala-Busse J., Slowinski R., and Ziarko W., Rough Sets. *Communications of the ACM*. Vol. 38, No. 11, 89-95, 1995.

Pawlak Z., Rough Sets. *Theoretical Aspects of Reasoning about Data*. Kluwer, Netherlands, 1991.

Pawlak Z., Rough Sets. *International Journal of Computation and Information Sciences*. 11, 341-356, 1982.

Petersen K. L., IDA-Intrusion Detection Alert. *Proceedings of the 16th Annual International Computer Software and Application Conference*, 306-311, September 1992.

Porras P. A., STAT: A State Transition Analysis Tool For Intrusion Detection. *Thesis*. University of California, Santa Barbara. July 1992.

Porras P. A. and Valdes A., Live Traffic Analysis of TCP/IP Gateways. *Proceedings of the Network and Distributed System Security Symposium,* Internet Society, San Diego, CA, March 11-13, 1998.

Quinlan J. R., Learning Efficient Classification Procedures and Their Application to Chess End Games. *Machine Learning, An Artificial Intelligence Approach.* 463-482. Tioga Publishing Company, Palo Alto, CA, 1983.

Rosenblatt F., *Principles of Perceptrons.* Sparton, Washington, DC, 1962.

Rumelhart D. E., Hinton G. E., and Williams R. J., Learning Internal Representations by Error Propagation. *Parallel Distributed Processing.* 318-362. The MIT Press, Cambridge, MA, 1986.

Slowinski R., Zopounidis C., and Dimitras A.I., Prediction of Company Acquisition in Greece by Means of the Rough Set Approach. *European Journal of Operational Research.* 100, 1-15, 1997.

Smaha S. E. and Winslow J., Misuse Detection Tools, *Computer Security Journal,* Vol. X, 1, 39-49, 1995.

Snapp S. R. and Smaha S. E., Signature Analysis Model Definition and Formalism, *Proceedings of the Fourth Workshop on Computer Security Incident Handling.* Denver, Colorado. August 1992.

Stillerman M., Marceau C., and Stillman M., Intrusion Detection on Distributed Systems, *Communications of the ACM*, 62-69, 42, 7, July 1999.

Szladow A. and Ziarko W., Rough sets: Working with Imperfect Data. *AI Expert*. 7, 36-41, 1993

Tan K., The Application of Neural Networks to UNIX Computer Security. *Proceedings of IEEE Symposium of Research on Security and Privacy*. 1995.

Teng H. S., Chen K., and Lu S. C., Security Audit Trail Analysis Using Inductively Generated Predictive Rules. *Proceedings of the 11th National Conference on Artificial Intelligence Applications*. 24-29, March 1990.

Warrender C., Forrest S., and Pearlmutter B., Detecting Intrusion Using System Calls: Alternative Data Models. *Proceedings of 1999 IEEE Symposium on Security and Privacy*. 133-145, May 1999.

Zighed D. A., Auray J. P., and Duru G., *SIPINA : Méthode et logiciel*. Editions A. Lacassagne Lyon, France, 1992.

ACKNOWLEDGEMENTS

I wish to express my deep appreciation to my major professor, Dr. Dan Zhu, for her consistent support and valuable advice during this thesis effort. I would also like to thank my committee members, Dr. Premkumar and Dr. Davis, for their understanding and support during this project.

My special thanks are overdue to Dr. Chao-Hsien Chu for his valuable advice and guidance during my course of study at ISU. I would also like to acknowledge the contributions of Dr. Stephanie Forrest in University of New Mexico and her student, Christina Warrender, in providing me with the experimental data and very helpful discussions and suggestions. Finally, thank you to my husband, Peng, and family. Without their help, this thesis would not have been possible.