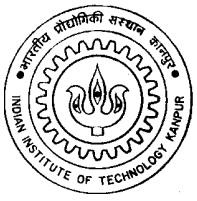
Design and Implementation of an Anomaly Detection Scheme in Sachet Intrusion Detection System

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

J.V.R.Murthy



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

June, 2004

Certificate

This is to certify that the work contained in the thesis entitled "*Design and Implementation of an Anomaly Detection Scheme in Sachet Intrusion Detection System*", by *J.V.R.Murthy*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

June, 2004

(Dr. Deepak Gupta)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

(Dr. Dheeraj Sanghi)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

Due to the widespread proliferation of computer networks, attacks on computer systems are increasing day by day. Preventive measures can stop these attacks to some extent, but they are not very effective due to various reasons. This lead to the development of intrusion detection as a second line of defense. Intrusion detection systems try to identify attacks or intrusions by analyzing network data (network-based systems) or operating system and application logs (host-based systems), possibly in real-time. These systems either search for patterns of well known attacks in the data (misuse detection) or try to find abnormalities in the data by first constructing the normal profile of the system under observation and then detecting deviations from this profile (anomaly detection). Anomaly detection is important due to the inability of misuse detection techniques in detecting unknown attacks.

In this thesis, we describe the design and implementation of an anomaly detection scheme for Sachet - A distributed, realtime, network-based intrusion detection system developed by us. In this scheme, the normal profile is constructed using learning techniques and stream handling techniques, from features extracted for each connection in the network traffic. Stream handling techniques are employed because the problem of constructing normal profile from feature vectors falls in the data stream class of problems. Several learning and stream handling techniques were tested on a benchmark data set and the best performing techniques were implemented in Sachet. The final system was tested on a benchmark dataset containing over 58 types of attacks.

Acknowledgements

I take this oppurtunity to express my sincere thanks to my thesis supervisors, Dr. Deepak Gupta and Dr. Dheeraj Sanghi, for providing me with many valuable ideas throughout the thesis. It was their constant support and encouragement that helped me complete this project in time. I also thank Prabhu Goel Reasearch Center for partially supporting my thesis. I would also like to express my sincere thanks to Dr. Pabitra Mitra for his valuable suggestions regarding the thesis. I also thank Vijaya Saradhi for his valuable suggestions and help regarding the learning techniques. He made my work easy by sharing his experience in machine learning. I also thank my project partner, Sachin Goel, for his cooperation and his timely and innovative suggestions regarding the project. It was a pleasure working with him. I also thank Sanjay Jain for his help and support at the beginning of my thesis.

I wish to express my sincere thanks to all the faculty members of the CSE department for imparting their knowledge and skill to me. I also thank all my classmates for the moments I shared with them. Being part of mtech2002 batch is a great experience; it made my stay at IITK a pleasant and unfogettable one.

Finally, I would like to thank my parents for their support and encouragement in all of my endeavours.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last few years there has been a tremendous increase in connectivity between systems which has brought about limitless possibilities and opportunities. Unfortunately, security related problems have also increased at the same rate. Computer systems are becoming increasingly vulnerable to attacks. These attacks or intrusions, based on flaws in operating system or application programs, usually read or modify confidential information or render the system useless. Formally, an intrusion is defined as any activity that violates the confidentiality, integrity or availability of the system.

Intrusion prevention is more desirable, but it cannot be fully achieved due to several reasons like unknown bugs in software, vast base of installed systems, abuse by insiders and human negligence. Many times it is difficult to have good access control while simultaneously making the system user friendly. Attacks are inevitable, but even after the attack has occurred, it is important to determine that the attack has happened, assess the extent of damage and track down the attacker. This helps in preventing future attacks. Due to these reasons, a detection system as a second line of defence is always desirable.

Intrusion detection systems (IDS) can be classified in two ways. The first one is based on the source of data being analyzed by the system. If the data is from operating system logs and application logs, it is called a 'host based' detection system; if the data is from network traffic, it is called a 'network based' detection

system. Each method has its own advantages and disadvantages. For example, an attack by a local user cannot be detected by a network based system, but a denial of service attack can be detected more efficiently by a network based system. Thus each method is more efficient in detecting a particular class of attacks than the other.

The other classification is based on the detection method being used irrespective of the source of data. The main types in this classification are misuse detection systems and anomaly detection systems. In misuse detection, well known intrusions are represented by signatures. Each signature is a pattern of activity which corresponds to the intrusion it represents. A detection system using such signatures is called a 'signature based' or a 'misuse detection' system. These detection systems search for patterns of intrusions in the data being analyzed. Thus misuse detection is basically a pattern matching process. Misuse detection systems are accurate and have a low false alarm rate, but they cannot detect unknown intrusions.

Anomaly detection systems assume that intrusions are anomalies or deviations from normal system activity. These detection systems try to capture the normal behaviour of the system (also called the normal profile), and then detect deviations from this normal behaviour. If this deviation is greater than a threshold, an alert is raised. Anomaly detection systems can detect unknown intrusions, but they have a high false alarm rate. There is generally a trade-off between detection rate and false alarm rate.

Several IDSs have been developed in the public and private domains using a variety of techniques and with varying features. Commercial IDSs mostly use signature based detection techniques. The features offered by them include scalability, real-time detection and a user friendly interface. Open source IDSs employ either misuse detection or anomaly detection or both. They offer features like scalability and real-time detection. For example, Snort [4], an open source IDS, employs misuse detection and is capable of doing real-time detection. Public domain research IDSs generally employ novel detection techniques. For example, ADAM [6] uses data mining techniques and IDES [17] uses statistical techniques.

Looking at the intrusion detection field from a research perspective, the research

in misuse detection is focused mainly on writing signatures which encompass all possible variations of an attack without matching normal activity, and on developing efficient methods of pattern matching. In anomaly detection, the main focus is on finding methods for representing the normal profile, selection of features used for constructing the profile and determining threshold levels so that most intrusions are detected while false alarms are minimized. In an overall system perspective, the focus of current research is on developing hybrid systems, *i.e* systems that are both network based and host based or that employ both anomaly detection and misuse detection.

1.1 Problem statement and Approach

In this thesis, we describe the design and implementation of a network based, real-time anomaly detection scheme for the *Sachet* IDS. *Sachet* is a network based, real-time, hybrid intrusion detection system developed at IIT Kanpur. *Sachet* employs both misuse detection and anomaly detection; hence it has the benefits of both the techniques, *i.e.* the accuracy of misuse detection systems in detecting known attacks, and the ability of anomaly detection systems in detecting unknown attacks. The *Sachet* IDS has agent based architecture with a central server. The detection is carried out at each agent and the results are aggregated at the server. The architecture is explained in more detail in Chapter 3. In the remaining part of this section, we describe the main issues involved in the thesis, followed by our approach.

The main task in anomaly detection is to construct the normal profile of the system under observation. This profile should adapt to the changes in the system over time. It should also be small enough so that real-time detection is possible. The profile is generally constructed from a set of measures or features extracted from the data being analyzed. In this case, the features are extracted from the network packets sniffed at appropriate points in the network being monitored. One of the main issues here is feature extraction in real-time.

The construction of profile from feature vectors follows the data stream model; we have a continuous stream of feature vectors and the profile at any point should

capture the information in the stream up to that point. If possible, the profile construction method should give more weight to newer data when compared with older data. Since the amount of network data is generally very large, any method used to construct the profile cannot obviously take the entire data seen in the stream so far, as input. Hence, efficiently dealing with the data stream is also a major issue here. Older data in the stream has to be discarded periodically, but the information in the discarded data has to be retained to some extent. Stream handling techniques have to be employed for this purpose. Finally, the detection technique has to be implemented in Sachet so that it requires minimal human intervention.

Our approach is as follows: the profile is learned from feature vectors using unsupervised learning (clustering) techniques. The features used for learning the profile are extracted for each connection in real-time, from the header and payload parts of network packets sniffed at various points in the network. Features corresponding to the payload part of the packet are extracted only for commonly used application layer protocols. These features are then aggregated at a single location, the Sachet learning agent, and the profile of the entire network is learned offline. Stream handling techniques are used to deal with the continuous stream of feature vectors. These techniques can be viewed as wrappers around the learning techniques. They construct a synopsis of the stream seen so far, with the possible option that newer data is given more weight in this synopsis. Learning is then applied on this synopsis and the resulting profile is distributed to the detection points where deviations are detected and alerts are raised.

Two different unsupervised learning techniques, support vector clustering [7] and a modified k-means technique [14] were considered for learning the profile. To handle the feature vector stream, three different techniques, Divide-and-conquer technique of clustering over data streams [15], reservoir sampling [25] and bootstrapping [16], were considered. The five valid combinations (a clustering technique and a stream handling technique) resulting from the above were tested on a benchmark data set. The combination that gave best results was implemented in the Sachet IDS. The implemented anomaly detection scheme was then tested on a benchmark data set of size 20GB, which contains over 50 attacks of various types.

1.2 Organization of report

Chapter 2 presents a brief overview of some of the techniques applied to anomaly detection and describes a few anomaly detection systems. Chapter 3 presents the architecture of Sachet and its components. Chapter 4 presents the results of evaluation of various learning techniques on the benchmark dataset. The results in this chapter form a justification for the choice of the methods used in the system. Chapter 5 describes the design and implementation of the anomaly detection system in Sachet. Chapter 6 presents the results of testing the system using the benchmark dataset. Chapter 7 presents conclusions and future work.

Chapter 2

Related work

In this chapter we present a brief review of the literature relevant to this thesis. We describe some of the techniques proposed for anomaly detection and a few actually implemented anomaly detection systems. We review some data mining techniques in section 2.1 and some machine learning techniques in section 2.2. In section 2.3, we review some stream handling techniques and their properties. Finally, in section 2.4, we look at some actually implemented anomaly detection systems.

2.1 Datamining techiniques

Data mining refers to the process of automatically extracting models from large stores of data [27]. Data mining techniques have been applied for both misuse and anomaly detection and for feature selection. In anomaly detection normal usage patterns are mined from audit data. In misuse detection encoded attack patterns are mined from audit data to detect intrusions. Thus, data mining techniques view intrusion detection as a data analysis process.

Data mining techniques like association rules [24], frequent episodes [20] and the RIPPER [9] algorithm are widely used for intrusion detection. Association rules are used to derive multi-feature correlations from a database table. Formally, an association rule is an expression of the form $X -> Y, confidence, support$, where X and Y are subsets of the feature set, *support* is the percentage of records in the table

that contain both X and Y and *confidence* is the ratio of *support* to the number of records that contain only X [24]. Association rules find intra-audit record patterns. On the other hand, frequent episodes, which are sets of events that occur together in a specified time window [20], are used to find inter-audit record patterns. The last of the above mentioned algorithms, RIPPER [9], is a rule learning algorithm. It generates a set of if-then rules using which one can classify test data.

A framework for constructing features and detection models using data mining techniques is proposed in [27]. The main idea is to use data mining techniques to identify useful patterns of user and program behaviour and use these patterns for detecting anomalies and known intrusions. As an example, RIPPER can be applied on normal and abnormal *sendmail* system call traces and the rules generated can be used to classify new traces as normal or abnormal. The problem of identifying useful features is also addressed in [27]. Association rules and frequent episode techniques are used to discover inter-audit and intra-audit record patterns. These patterns help the user in selecting relevant features.

2.2 Machine Learning Techiniques

Learning algorithms generally try to construct a classifier using training data, and later apply this classifier on test data. Two forms of learning, *supervised learning* and *unsupervised learning*, are generally applied for intrusion detection. In supervised learning, a cost metric or label is provided for each training pattern by a teacher. The goal here is to reduce the total cost for all training patterns. In unsupervised learning or clustering, the algorithm tries to form 'natural groupings' or clusters of the input patterns without the involvement of a teacher.

Among supervised learning techniques, neural networks have been widely used for intrusion detection [22], and recently support vector machines have also been used [21]. Neural networks are constructed from an interconnected set of units called neurons. Each neuron takes a number of real-valued inputs and produces a single output. Artificial neural networks are inspired from the biological learning system which is built of a complex web of interconnected neurons. On the other

hand, support vectors machines (SVMs) are derived from the statistical learning theory. SVMs are binary classifiers; they try to construct an optimum hyperplane after transforming the training points from the input space to a higher dimensional feature space. The optimality criterion for constructing the hyperplane is to maximize the margin of separation of the hyperplane from the two classes of training points. Intrusion detection using neural networks and support vector machines was described in [21]. Using supervised learning techniques is not very practical for intrusion detection because these techniques require both normal and attack data; but in practice, it is difficult to get real attack data.

Unsupervised learning techniques have also been applied to intrusion detection. The most popular unsupervised learning method is the k-means clustering algorithm. The main goal of the algorithm is to choose k centers in the input space so that the sum of the distances of the training points from their nearest cluster center is minimized. But the drawback of this algorithm is that the value of k has to be decided beforehand, which is difficult as k depends on the data. Many modifications of this algorithm have been proposed to overcome this drawback. Y-means [14] is one such algorithm which tries to bring out the actual number of clusters in the data given as input. A clustering technique based on the SVMs, calledsupport vector clustering, was proposed recently [7]. It tries to construct a sphere of minimal radius in the feature space that encloses all the training points.

2.3 Stream Processing Techniques

A data stream is a massive sequence of elements arriving at a rapid rate. The general data stream computation model contains a data stream, a stream processing engine and a synopsis in memory, along with the requirements that each record can be accessed only a finite number of times, the memory for storing synopsis is limited and the processing required to maintain the synopsis must be low. Data processing in network monitoring applications generally follow this model because these applications generate large streams of data.

There are several stream processing techniques available. One method of constructing the synopsis is by using sampling techniques like reservoir sampling [25] and concise sampling [13]. In reservoir sampling, a sample of a fixed size M is maintained and new elements are added to the sample with a probability M/n, where n is the total number of stream elements seen so far, by evicting random elements from the sample. In concise sampling also, a sample is maintained, with duplicates stored as (value, count) pairs. For each new element, count is incremented if the element is already present in the sample; otherwise the element is inserted into the sample with some probability.

Clustering under the data stream model is described in [15]. The approach is to divide the stream into disjoint windows, find k centers for each window weighted by the number of points assigned to them, and finally apply clustering on these weighted centers to obtain the clustering of the entire stream.

2.4 Anomaly detection systems

In this section we describe four anomaly detection systems that use four different techniques for learning the normal profile of the system under observation.

2.4.1 ADAM

ADAM (Audit Data Analysis and Mining) [6] uses several data mining techniques to discover abnormal patterns in large amounts of data like network audit data. It discovers frequent events in network traffic and uses them to build a profile of normal network activity. During detection time it employs a sliding window method and within each window it considers frequent events that do not appear in the profile as anomalous. The limitation of ADAM is that it cannot detect stealthy attacks which cause a relatively small number of events within a short period of time.

2.4.2 NNID

In [22], a method of applying neural networks for intrusion detection is proposed. It is based on the idea that every user leaves a 'print' when using the system and a neural network can be used to learn this print and identify each user. If the behaviour of a user does not match this print then an alert for a possible intrusion is raised. The system is called NNID (Neural Network Intrusion Detector). It is an offline anomaly detection system which uses a back-propagation neural network to identify users based on the distribution of commands used by them. It assumes that different users exhibit different behaviours based on their needs. The set of commands and their frequencies form the print of the user. The model is implemented in a UNIX environment where the audit logs for each user are collected for a period of several days. Command distribution vectors are extracted from these logs and the network is trained to identify the print of each user. The network is then used to identify the user for each new command vector and if the suggestion is different from the actual user or if the network does not have a clear suggestion then an anomaly is signaled.

2.4.3 IDES Statistical Anomaly Detector

The IDES statistical anomaly detector is part of the SRI International's host-based real-time intrusion detection expert system [17]. It is based on the general anomaly detection model proposed in [12]. It observes behaviour on a computer system and adaptively learns what is normal for users and groups. It also raises alert for a potential intrusion if the observed behaviour deviates significantly from expected behaviour. It uses multivariate methods to learn normal behaviour.

IDES maintains a statistical knowledge base consisting of normal profiles of subjects. The definition of a profile, as given in [17], is "a description of a subject's behaviour with respect to some intrusion-detection measures". Profiles are constructed from audit records and consist of statistics such as frequency tables, means and co-variances. Each audit record is a vector of intrusion-detection variables corresponding to the measures recorded in the profiles. It can be represented by a point in the n-dimensional space. If this point is sufficiently far from the point defined

by the values stored in the profile then it is considered anomalous. Thus the system takes into account the values of individual variables as well as the correlation between them.

The statistical knowledge base is updated daily using the observed behaviour of subjects. The means, frequency tables and co-variances in the profile are multiplied by an exponential decay factor periodically. This ensures that recent behaviour is given more weight than old behaviour resulting in a changing profile over time as the behaviour of the subject changes.

The IDES statistical anomaly detector uses a single point in n-dimensional space to represent the profile. But generally, normal data itself is very diverse and a single point cannot represent the entire spectrum of normal activity.

2.4.4 Defence using autonomous agents

In [11], an architecture is proposed in which programs use genetic programming to evolve and detect anomalies. These programs, called autonomous agents, run independently of each other and of the jobs already on the system. The agents learn normal and intrusive behaviour by observation and adapt to changing profiles. A prototype solution in which the agents monitor the network traffic on a system is described. In this solution the agents access the network data through a well defined set of primitives. They require the values of various fields in the network packet headers and a variety of aggregate values such as average packet size, inter-packet arrival times etc.

Each agent can be represented as a parse tree for a simple language. This language allows the agent to inspect contents of network packets and act accordingly. Before the agents are deployed for detection they are trained to identify intrusions and minimize false positives. This involves human interaction with the agents via the training module. The operator presents both normal and intrusive traffic to the agents and guides their learning through a feedback mechanism. The agents use genetic programming to actually learn.

Many agents are evolved at the same time with each agent monitoring a small aspect of the overall network traffic. The agents cooperate by communicating their

suspicions among themselves. Each agent makes a suspicion broadcast whenever it believes that the observed activity is suspicious. As successive agents analyze packet data and make such broadcasts, the level of suspicion rises above a predefined threshold and the system raises an alert, indicating a possible intrusion. The main drawback of this system is that it requires manual intervention during the training phase.

Chapter 3

Architecture of Sachet IDS

In this chapter, we briefly describe the architecture of the Sachet IDS. We begin by introducing the components in the system and the interactions between them at a high level. In the subsequent sections, we briefly describe the components in the system, as they were before anomaly detection was incorporated in Sachet. In the final section of this chapter, we describe the changes made to the architecture and the components for incorporating anomaly detection into Sachet.

The architecture of Sachet is shown in Figure 3.1. The components in the system, as can be seen from the figure, are multiple Sachet agents, a Sachet server and the Sachet console. The agents and the server communicate with each other using the Sachet protocol, which provides authentication, reliability, confidentiality and integrity.

Sachet agents are deployed at various points in the network, depending on its topology. Their main task is to monitor the network for intrusions. The Sachet server is deployed on a dedicated machine and is responsible for controlling the agents, collecting data from agents and interacting with the Sachet console. It uses a database to store configuration information and alerts. The user interacts with and controls the system using the Sachet console. Generally the server and the console are installed on the same machine.

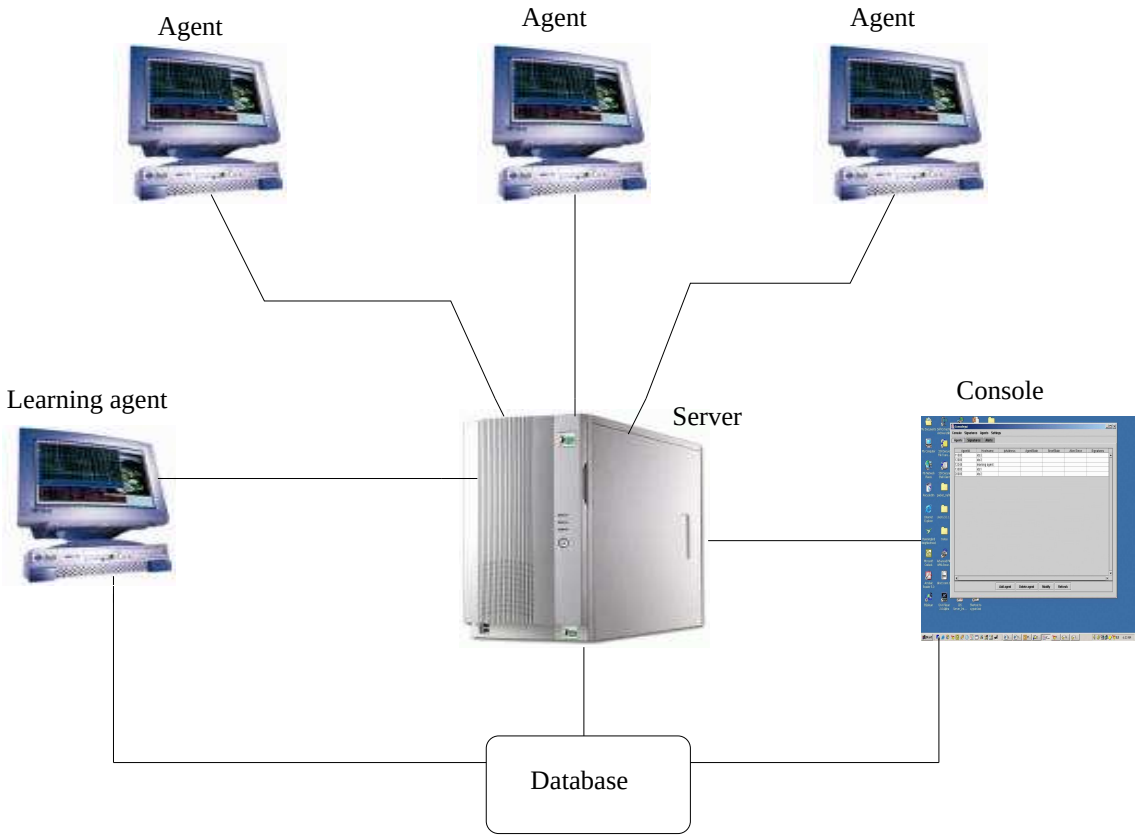


Figure 3.1: Architecture of Sachet IDS

3.1 The Sachet Protocol

The Sachet protocol, used for communication between agents and server, is implemented over UDP. It provides authentication, encryption and reliability to the communicating parties. Authentication is done using public key cryptography method. The server maintains the public keys of all the agents and similarly each agent maintains the public key of the server. During authentication, each side proves the ownership of its public key to the other side using a challenge-response mechanism. After authentication is completed in both directions, the server sends a random secret key to the corresponding agent. All the messages from this point are encrypted using the secret key. This key is changed periodically.

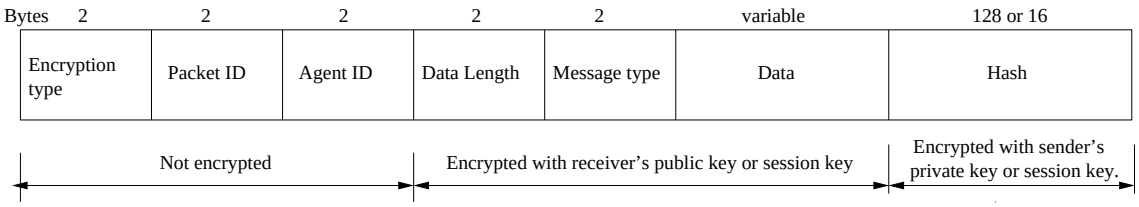


Figure 3.2: Message format

Reliability is achieved using acknowledgements, timeouts and retransmissions. Every message in the protocol is acknowledged by the recipient. If the sender does not receive the acknowledgement within a time period, either because the packet is lost or because the recipient is down, it retransmits the packet. The retransmission is done until either the sender gets an acknowledgement or the retransmission count exceeds a threshold. In the later case, the sender assumes that the other side is down and goes into the initial unauthenticated state. The retransmission timeout is updated using the round trip time of each message.

Every message in the protocol uses exactly one UDP packet. The general message format is shown in figure 3.2. Each message has a header of size 10 bytes, divided into 5 fields of 2 bytes each. The 'encryption type' field is used to specify the encryption method, used for the current message, to the receiver. It takes one of three possible values representing the following: encryption is not used, RSA encryption is used or symmetric encryption is used. The 'packet id' field contains a unique integer for each message with respect to the sender and is used to identify duplicates. The 'agent id' field is used to identify the sender of the message. Each agent is assigned a unique 2 byte non-zero integer for this purpose. The messages from server have 0 in this field. The 'data length' field gives the length of the data portion of the packet in bytes. The 'message type' field identifies the message present in the current packet. The data field is interpreted based on this value. The 'hash' field contains the encrypted MD5 checksum of the entire packet. The size of this field is 128 bytes if RSA encryption is used and 16 bytes if 3-DES is used.

3.2 The Sachet Server

The Sachet server is a console based application which can run in the background as a daemon or service. It does not have any user interface but it can interact with other programs acting as user interfaces. The main functions of the server are as follows: the server acts as a central point from which the entire system can be controlled by the administrator. The server aggregates data from agents and stores this data in the database. The data received from agents include alerts generated for possible intrusions. The server maintains information about agents in the database and retrieves it at the beginning of its execution. It also communicates with the Sachet console using a simple request-response protocol in which the console sends a request for some information or a command from user and the server responds with the appropriate information or result.

3.3 The Sachet Agent

The Sachet agent is also a console based application which can run in the background. It does not interact with the user; as the name indicates, it does work on behalf of the server. It consists of two main components: the control agent and the misuse detector. These two components run as separate processes on the target host. The main functions of the control agent are communicating with the server, executing commands from server locally and controlling the misuse detector.

The misuse detector analyzes network packets in real-time for finding possible intrusions. It has a database of attack signatures. It applies them on each connection and raises an alert on finding a match. Snort [4] is currently used as the misuse detector in Sachet. The agent can be deployed to monitor either an entire network segment or a single host.

3.4 The Sachet Console

The Sachet console is a Java based GUI application using which the administrator interacts with the system. It communicates with the server on a predefined port. It

also interacts with the database for extracting information requested by the administrator. The administrator uses the console to configure, monitor and control the system from a central location. For example, the administrator can add a new agent to the system using the console. The console presents important information about all the agents on a single screen and more detailed information about each agent on a separate screen. It can also show alerts from all the agents simultaneously.

3.5 Incorporating anomaly detection in the Sachet Architecture

In this section, we describe the changes made to the architecture and components of Sachet to incorporate anomaly detection into Sachet. The major changes to the architecture are the addition of a learning agent to Sachet and the inclusion of an anomaly detector at each agent in the system.

The anomaly detector at an agent processes network traffic and produces a compact representation called feature vector for each connection. Using this feature vector and the normal profile of the system, it generates an anomaly score and detects deviations from the normal profile. It raises an alert if this deviation is more than a threshold. The 'control agent' component at the agent receives alerts and feature vectors from the anomaly detector and sends them to the server.

The Sachet server is modified to receive both alerts and feature vectors from agents; these alerts and feature vectors are stored in the database by the server. The server periodically instructs the learning agent to learn the normal profile of the system. It collects the normal profile from learning agent and distributes this profile to agents in the system.

The Sachet learning agent uses the same protocol described earlier for communicating with the server. The main task of the learning agent is to learn the normal profile of the system. It authenticates with the server like other agents, and waits for commands from server. Upon receiving the 'start learning' command from server, it fetches the feature vectors from database and applies the learning algorithm on this data. When the learning is completed, it sends the result or profile generated

by the algorithm to the server.

The Sachet console was modified so that it gives information about the learning agent in a separate screen. The user can start the learning of profile at the learning agent from the console. Many new messages were added to the Sachet protocol to implement the new features of server and agent described above. Since the server has to deal with learning agent also, many learning agent specific messages were also added to the protocol.

Chapter 4

Application of Learning Techniques for Anomaly Detection

In this chapter, we describe the learning and stream handling techniques considered for implementing in the anomaly detection scheme, along with their application to intrusion detection, and give the results of testing these techniques on a benchmark dataset. We begin by giving a classification of attacks that is used throughout this report for presenting results. In the second section, we briefly describe the features used for testing. We describe some learning and stream handling techniques in the subsequent sections, and in the last section we present the results of our tests.

4.1 Classification of attacks

Classification of attacks into groups that share common properties will make the presentation and analysis of results easier. The classification given here was originally presented in [26]. It is based on the level of access of the attacker and the transitions thereof. In this taxonomy, there are four levels of access an attacker can have. They are remote access, local access, superuser access and physical access. In remote access, the attacker can send network packets to the victim machine but he does not have an account on that machine. In local access, the attacker has an

account on the victim machine and in superuser access, the attacker has root privileges on the victim machine. In each class the attacker either performs some action at a particular level or tries to obtain a higher level of access. The four classes in this classification are explained below.

4.1.1 Denial of Service Attacks

In a denial of service attack (DoS) the attacker tries to render a resource or system feature unusable by legitimate users by making it too busy with false requests. There are different kinds of denial of service attacks. Some attacks try to exploit bugs in network software and protocol stack by sending malformed packets. Others send valid requests at a very fast rate so that the victim machine cannot handle them. Remote access is generally sufficient to perform DoS attacks. Examples of DoS attacks are back, ping of death, smurf, neptune, teardrop etc. [2]

4.1.2 Probes

Probes do not cause any damage by themselves but they provide valuable information which can be used later to launch an attack. Remote access is sufficient to do probing. The attacker tries to search for valid IP addresses, services running on each machine or for known vulnerabilities. Examples of probes and probing tools are ipsweep, mscan, nmap, saint, satan etc. [2]

4.1.3 Remote to User

In a remote to user attack, the attacker has remote access to a system but not local access. He tries to exploit some vulnerability in the system to gain local access. The vulnerabilities include buffer overflows in network server software, weakly configured and misconfigured systems etc. Examples of remote to user attacks are dictionary attacks, guest login, ftpwrite, sshtrojan, httptunnel etc. [2]

4.1.4 User to Root

In a user-to-root attack, the attacker has local access on a system and by exploiting some vulnerability he gains superuser privileges on that system. The most common vulnerability is the buffer overflow vulnerability. Other vulnerabilities like bugs in management of temporary files and race conditions are also exploited in these attacks. Examples in this class are eject, loadmodule, casesen, anypw, yaga etc. [2]

4.2 Features

The features used in the evaluation can be divided into four main categories. They are general, time-based, host-based and content-based. The general category contains features like protocol, service, number of source bytes etc. Time-based features are derived features which are extracted by considering connections in a 2 second time window. Host-based features are extracted by considering the last 100 connections to the same host. Content based features are extracted from the data portion of the packet and require analysis of application layer protocols.

4.2.1 General features

Duration Length of the connection in number of seconds.

Protocol Transport layer protocol of the packet, such as TCP, UDP etc.

Service Network service on the destination such as FTP, HTTP etc. This information can be obtained from destination port number. e.g, FTP, HTTP etc.

Source bytes Number of data bytes from source to destination.

Destination bytes Number of data bytes from destination to source.

Flag Status of the connection. This feature indicates whether the connection is half closed, fully closed and whether there are any errors in the connection.

land This value is 1 if the source IP address or port number is equal to destination IP address or port number. Otherwise it is 0.

Wrong fragment Number of wrong fragments. A wrong fragment is an IP fragment whose length is not a multiple of 8.

Urgent Number of packets in which the urgent flag is set.

4.2.2 Content-based features

Hot Number of hot indicators like access to system directories, creation and execution of programs etc.

Number of failed logins Number of failed login attempts.

Logged in 1 if the login is successful. 0 otherwise.

Number of compromised conditions Count of "file path not found" error.

Root shell 1 if root shell is obtained. 0 otherwise.

Su attempted 1 if root access is attempted. 0 otherwise.

Num root Number of commands typed as root.

Num file creations Number of file creation operations.

Num shells Number of shell prompts.

Num access files Number of operations on access control files.

Num outbound commands Number of outbound commands in an FTP session.

Is hot login 1 if the login belongs to the 'hot' list; 0 otherwise.

Is guest login 1 if the login is a guest login; 0 otherwise.

4.2.3 Time-based features

Some of the features in this and in the next category use the concept of SYN error and REJ error. A connection which has less than 2 SYN packets is said to have a SYN error. A connection which is rejected by setting the RST flag is said to have a REJ error. All these features are extracted by considering connections in a 2 second time window.

Count Number of connections to the same host as the current connection.

Serror rate Percentage of connections that have SYN errors.

Rerror rate percentage of connections that have REJ errors.

Same srv rate Percentage of connections to the same service.

Diff srv rate Percentage of connections to services other than the current service.

Srv count Number of connections to the same service as the current connection.

Srv serror rate Percentage of connections to the same service as the current one having SYN errors.

Srv rerror rate Percentage of connections to the same service as the current one having REJ errors.

Srv diff host rate Percentage of connections to hosts other than the current host.

4.2.4 Host-based features

The features in this section are extracted by considering past 100 connections to the same host as the current host.

Dst host count Count of connections having the same destination host as the current one.

Dst host srv count Count of connections having the same destination host and same service as the current one.

Dst host same srv rate Percentage of connections having the same destination host and using the same service.

Dst host diff srv rate Percentage of connections to the same host as the current one and are to services other than the current one.

Dst host same src port rate Percentage of connections to the current host having the same source port.

Dst host srv diff host rate Percentage of connections to the same service as the current one but coming from hosts others than the current one.

Dst host serror rate Percentage of connections to the current host that have a SYN error.

Dst host srv serror rate Percentage of connections to the current host and current service that have an SYN error.

Dst host rerror rate Percentage of connections to the current host that have a REJ error.

Dst host srv rerror rate Percentage of connections to the current host and current service that have an REJ error.

4.3 Supervised Learning

In supervised learning, each point in the training data has a class label assigned to it, which is used by the learning algorithm during the training phase. The learning algorithm tries to construct a model which can classify the training data as accurately as possible. For example, in neural networks the model consists of the weights on the paths connecting the neurons. These weights are adjusted during the training phase using the class labels of the training data. Similarly in a support vector machine, which is a binary classifier, the model consists of a hyperplane separating the two classes. The position of hyperplane is adjusted in such a way that the margin of separation between this plane and the nearest data points of the two

classes on either side of this plane is maximized. Again, this optimization is carried out using the class labels of the training data set. Thus, these algorithms depend heavily on the labels of training data set and any error in this labeling will result in an inaccurate classifier.

When these learning techniques are applied to intrusion detection, they require accurately labeled attack and normal data during the training phase. But in any practical IDS, it is not possible to accurately label the data; some attacks are flagged as normal (false negatives) and some normal connections are flagged as attack (false positives). Generating training data manually is not a viable option because the amount of network data is generally very large and the training has to be done periodically to cope up with the changing patterns of network activity. Another important issue here is the relative size of normal and attack data in the training data. Generally, normal data will be overwhelmingly large when compared to attack data. Since supervised learning techniques try to reduce the error in classifying training data, by the output classifier, if one of the input classes has very few points, the learning algorithm may ignore this class. This is a big drawback in our case because if the attack data, which is present in relatively small numbers generally, is ignored either partially or completely by the learning algorithm, the detection rate will fall drastically.

While choosing the learning algorithm for the anomaly detection scheme we considered the following supervised learning techniques: support vector machines [10], supervised k-means algorithm [18] and soft linear vector quantization [23]. But due to the reasons mentioned above, we did not consider these algorithms during the testing phase in which different learning algorithms were compared (by testing them on a benchmark dataset) to choose the best one.

4.4 Unsupervised Learning

In unsupervised learning algorithms, the training data does not need to be labeled. These algorithms try to bring out 'natural groupings' or clusters from the training data, by looking at how close a point is from the rest of the points in the training

data. The degree of closeness is determined by using a metric; the most widely used metric is the Euclidean distance.

Unsupervised learning can be effectively applied to anomaly detection. In anomaly detection, the main goal is to learn the profile of the system under observation and then detect deviations from this observed profile. In the case of a network based IDS, the normal profile is learned from feature vectors extracted for normal connections. Unsupervised learning techniques do not have the drawbacks of the supervised learning techniques mentioned above, when applied to intrusion detection. Even in this case the decision regarding the normality of a connection is made by the IDS and hence the training data, from which normal profile is learnt, may not entirely be normal. However, the algorithm tries to bring out the differences among the input points, and hence we can always identify and discard most of the attack data, so that the normal profile is learnt from mostly normal data. Our approach is to discard 2 percent of the training data on the grounds that it is possibly anomalous. The normal profile retains information from the rest of the 98 percent training data.

We considered two unsupervised learning techniques for anomaly detection. They are y-means [14] and support vector clustering [7]. We briefly describe these two techniques and their application to anomaly detection in the following two subsections.

4.4.1 Y-means clustering

K-means is a popular clustering algorithm which partitions the input data into k groups based on a similarity metric. Its main drawback is that the result depends on the value of k and finding an optimal value of k is not easy. Many modifications have been proposed to k-means to overcome this drawback. The y-means clustering technique [14] defines three operations for this purpose: empty cluster removal, splitting and merging. Empty cluster removal simply removes zero sized clusters. Splitting is used to break up clusters with a large number of outlier points into multiple clusters. A cluster has outliers if the point farthest from the cluster center is not within a radius of (mean + r * standard deviation), where mean and standard deviation are calculated on the Euclidean distances of the points in the cluster from

the center, and r is an integer. For all the clusters with outliers, the farthest point in the cluster is taken as a new cluster center and the k-means algorithm is applied again. After each iteration, empty clusters are removed and new clusters are added by splitting clusters with outliers. This process is continued till outliers are not found in any cluster. In this clustering technique the final number of clusters does not depend on the value of k.

The output of this technique is a set of points representing the cluster centers. A threshold is calculated such that 98 percent of the training data points lie within this distance from their nearest cluster center. For classifying the test patterns the distance of the test pattern from the nearest cluster center is calculated and if this distance is greater that the threshold, the test pattern is classified as an attack. Otherwise it is classified as normal.

4.4.2 Support Vector Clustering

The main idea in support vector clustering [7] is to represent the normal data by a sphere of minimal radius in a higher dimensional space using a non-linear kernel. Here input data points are mapped to a higher dimensional feature space using a Gaussian kernel, and a sphere with minimal radius enclosing all these points is constructed. When this sphere is mapped back to the input space, it separates the data into several components or clusters. As the width of the Gaussian kernel is decreased, the number of clusters increases. The method allows outliers to be present by employing a soft margin in which not all points are required to be within the sphere in the feature space. The points that lie on the surface of the sphere are called support vectors and the points that lie outside the sphere are called bounded support vectors. In the input space, the support vectors form contours of clusters and the bounded support vectors form the outliers. The percentage of outliers is determined by the value of the soft margin constant C. If the value of C is 1 then no outliers are present.

In our case, we stop after constructing the sphere. This is because, we want only a representation of the normal data *i.e.* the regions in the n-dimensional space containing the normal data, but not the exact clusters. Since the sphere itself gives

us this information (whether a point is inside any normal cluster), we do not need to map it back to the input space and find the exact clusters. The normal profile is represented by the center and the radius of this sphere, which are in turn represented by the support vectors. We modified LIBSVM [8], a library for support vector machines, to find this sphere. We used an RBF kernel with its width parameter set to $1/k$, where k is the number of features. The soft margin constant C is set to 1.0 which means that outliers are not allowed. The radius is calculated so that only 98 percent of the training points are inside the sphere. During testing, the test pattern is mapped to the feature space and its distance from the sphere center is calculated. If this distance is greater than the radius of the sphere, the test pattern is considered anomalous.

4.5 Stream Processing Techniques

The problem of construction of the normal profile from feature vectors falls in the stream processing class of problems, since feature vectors form a data stream and the profile at any point must capture the information in the stream up to that point, with the possible option that newer data is given more weight than older data. Learning algorithms alone cannot achieve this because they cannot work in an incremental fashion and they cannot handle very large amounts of input data (if we want to create the profile from the entire stream). To overcome this problem stream handling techniques are required. These techniques maintain a synopsis of the stream such that processing the synopsis at any point is approximately the same as processing the entire stream up to that point. The major restrictions on stream processing techniques are: each stream element can be seen only a finite number of times, the size of synopsis is limited and the time taken to maintain the synopsis should be low. There are several stream processing techniques available out of which we considered three techniques: a divide-and-conquer technique for clustering data streams, reservoir sampling and bootstrapping. We briefly describe each of these techniques in the following subsections.

4.5.1 Clustering Data Streams

A method for clustering under the data stream model was described in [15]. The main objective in this method is to maintain a good clustering of the points observed so far, using small amounts of memory and time. A divide-and-conquer approach is used in which the stream is divided into disjoint parts, each part is clustered separately, and finally, the cluster centers obtained for individual parts are clustered to obtain a clustering for the entire stream. While clustering the centers, they are weighted by the number of points associated with them. This basic approach is then extended so that it fits under the data stream model. The main restriction here is the space required to store the intermediate cluster centers. To achieve clustering in a limited space, whenever the number of intermediate cluster centers in the memory reaches a threshold, they are clustered again to get much fewer second level cluster centers. In general, when the number of centers at level i reaches a threshold, they are clustered to obtain much fewer cluster centers at level $i + 1$. When the clustering of points observed so far in the stream is needed, the centers at all levels in the memory are clustered to obtain the final clustering.

This method can be applied on all clustering algorithms that output cluster centers, *i.e*, on all k-means class of algorithms. Since the support vector clustering method does not give cluster centers, this method for dealing with data streams cannot be applied along with support vector clustering. It can be applied along with the y-means algorithm described in the previous section, to generate the normal profile of the network traffic. Whenever the profile needs to be updated, the most recent points in the stream are first clustered to obtain centers and then these centers are merged along with the centers in the previous profile and clustered again to get the new profile. Weighted clustering is done in all the cases.

4.5.2 Reservoir Sampling

A random sampling technique for maintaining the synopsis in the data stream model was described in [25]. This method, called reservoir sampling, maintains a true random sample of the data seen so far in the stream, in a reservoir. All algorithms that maintain a true random sample of fixed size after processing each record in

the stream are called reservoir algorithms. Initially, these algorithms put the first few records of the stream into the reservoir till the reservoir is full. After this, each record in the stream is considered for inclusion into the reservoir, and if chosen, it replaces a randomly chosen sample from the reservoir.

This stream handling method can be applied along with both the unsupervised learning techniques described above. The learning algorithm is applied on the records in the reservoir. The size of the reservoir is fixed beforehand depending on the memory and processing power of the host on which learning is carried out. Whenever the profile needs to be generated/updated, a reservoir algorithm is applied on the most recent data in the stream and the reservoir is updated first. The learning algorithm is then applied on the reservoir to get the new profile.

4.5.3 Bootstrapping

Bootstrapping is a technique used to generate artificial training data set from original training data set [16]. The bootstrap method has also been successfully applied for error estimation and 1-NN classifier design [16]. We use the bootstrap method in the context of sampling and data reduction. The method of maintaining synopsis using bootstrap samples is as follows: every time after the profile is generated, a bootstrap sample of size $n/2$ is taken, where n is sum of the number of records in the current synopsis and the number of new records observed in the stream. This sample is set as the new synopsis. The profile is generated using the records in the synopsis and the most recent data from the stream.

4.6 Experimental Evaluation

An anomaly detection scheme requires a learning technique and a stream handling technique. To choose the best combination, the learning and stream handling techniques described above were evaluated using a benchmark data set. Since two learning techniques and three stream handling techniques were considered, there are six possible combinations out of which five are valid. The method of clustering in data streams cannot be applied for support vector clustering for the reasons mentioned

earlier. These five combinations were tested on a benchmark data set and the combination that performed the best was considered for implementation in the Sachet system.

4.6.1 Preparation of Datasets and Criteria for evaluation

We used the data provided in the 1999 KDD Cup [3], for testing the techniques mentioned above. Each point in this data corresponds to a network connection and contains values of the 41 features described in Section 4.2. This data itself was obtained by extracting features from the 1998 DARPA IDS evaluation data.

A sample of 1,50,000 feature vectors corresponding to normal connections was taken from the 1999 KDD cup data. The data stream was simulated using these feature vectors. The profile was generated/updated every time after 25,000 feature vectors were processed. Thus, for each combination, the profile was generated six times. The classifier obtained at the end was tested on three test datasets. The test datasets were prepared from KDD cup data using *stratified sampling*. Stratified sampling is a random sampling technique in which data points are first separated into mutually disjoint sets and then each set is sampled separately. This method is advantageous if the number of data points in each class vary drastically. In the KDD cup data some attack types have thousands of data points while some have as few as two or three data points. Hence stratified sampling was used to create the datasets.

Another important issue here is the normalization of data. Each feature in the data has its own range. Some features have very large ranges where as some take only binary values. Hence all features will not have equal weight during the learning process and features with bigger ranges exert greater influence than those with smaller ranges. To solve this problem data needs to be normalized. The *z-score normalization technique*, described below, was used for this purpose.

Z-Score normalization: The mean (μ) and the standard deviation (σ) of the data to be normalized are first calculated and the normalized instance is calculated as follows,

$$x_i' = \frac{x_i - \mu}{\sigma} \tag{1}$$

Training data is first normalized using this technique and the mean and standard deviation vectors are saved. Test data is then normalized using the saved mean and standard deviation vectors.

The evaluation criteria used for comparing the performances are *detection rate* and *false alarm rate*. Detection rate is equal to the number of intrusions detected, divided by the total number of intrusions present in the data set. False alarm rate is equal to the number of normal connections classified as intrusive by the algorithm divided by the total number of normal connections. The detection rate should be as high as possible and the false alarm rate should be as low as possible. Apart from these criteria, the training time and the output size of the algorithm were also considered. The training time is important as the profile has to be updated periodically. The size of the result is important as real-time detection is desired.

4.6.2 Results

The results of applying the five valid combinations on the benchmark data are shown in Table 4.1. The numbers given in this table are the averages of results over all three test datasets. Note that the results here indicate the performance of the stream handling technique and learning algorithm together, rather than the learning algorithm alone.

When we compare the detection rates among attack classes, all combinations gave the best results for probe class of attacks, and performed fairly well for user-to-root (U2R) and DoS class of attacks. The detection rate was lowest for the remote-to-login (R2L) class of attacks. The time-based and host-based features are designed for DoS and probe class of attacks and the host-based features are designed for the R2L and U2R class of attacks. The detection rate for probes and DoS attacks is expected to be high because they are basically network-based attacks. The detection rate of DoS attacks is lower than that of probes because the attacks 'back' and 'mailbomb', which are present in large numbers in the test data, were not detected by any of the methods. The detection rate of U2R attacks, which are inherently host-based attacks, shows that the content-based feature capture the general patterns of these attacks well enough. The low detection rate of R2L attacks shows that the features

Combination	Detection rate					False Alarm Rate
	DoS	Probe	R2L	U2R	Total	
BS-YM	59.14	99.21	57.07	90.35	72.46	2.19
BS-SVC	69.58	98.05	51.01	78.68	75.76	2.88
RS-YM	54.07	94.79	55.36	84.21	68.0	1.53
RS-SVC	68.96	92.69	47.12	74.59	72.93	1.87
DQ-YM	69.35	99.48	52.49	71.93	76.26	2.76

BS - bootstrap technique.
RS - reservoir sampling.
DQ - divide-and-conquer method of clustering data streams.
YM - y-means clustering.
SVC - support vector clustering.

Table 4.1: Comparison of detection rates and false alarm rates

used currently do not capture these attacks properly and hence there is a need to define more features appropriate to this class of attacks.

The performance of learning techniques alone can be compared by fixing the stream handling technique. Support vector clustering performed better than y-means when applied along with both reservoir sampling and bootstrapping stream handling techniques. This shows that the performance of these learning techniques is independent of the stream handling techniques. Among the stream handling techniques, the divide-and-conquer method of clustering data streams performed the best followed by bootstrapping and then the reservoir sampling. This can be observed by comparing the stream handling techniques for a given learning technique.

Another important criterion is the false alarm rate. From the table, it can be seen that the false alarm rate is lowest when the reservoir sampling technique is applied. Among the learning algorithms, the false alarm rate is lowest for y-means clustering for a given stream handling technique.

In terms of detection rate the best performing combination is the stream clustering and y-means combination, and hence it is the natural choice for the anomaly detection scheme. But the false alarm rate is a bit high for this combination and also for the second best combination. Since false alarm rate is also an important criterion, we may also choose a combination that performs fairly well with respect

to both detection rate and false alarm rate. From the table, it can be seen that the reservoir sampling and support vector clustering combination is exactly like this, *i.e,* it shows good detection rate and low false alarm rate simultaneously.

The other criteria considered are the training time and the profile size. The training time of y-means algorithm is around 15 minutes for an input size of 25000 and the training time of support vector clustering is around 2 minutes for the same size of data. If stream handling techniques are being used, then we can always fix the size of the synopsis such that the training time is reasonable enough for a given learning technique. The same is the case with profile size also.

We chose the reservoir sampling and support vector clustering combination for implementation in the Sachet system.

Chapter 5

Design and Implementation

In this chapter we describe the design and implementation of an anomaly detection scheme in Sachet. In Section 5.1, we describe the design, and in Section 5.2, we describe feature extraction followed by the implementation of this scheme at each of the components in Sachet.

5.1 Design of the anomaly detection scheme in Sachet

Any anomaly detection scheme involves the construction of a normal profile. Since Sachet is a network-based intrusion detection system, the normal profile is constructed by observing patterns in network traffic. More precisely, it is constructed by using certain metrics or features extracted from network traffic. Since the monitoring points in Sachet are the agents, feature extraction is done at the agents. These feature vectors are aggregated at the server for constructing the normal profile. The task of learning the profile from these features can be implemented at the server itself, but it can also be implemented in a special agent whose purpose is not to monitor the network but to learn the normal profile. In the later case, the learning process is transparent to the server and hence the learning technique can be changed without changing the server. This approach also reduces load on the server since learning is a computationally intensive task. The special agent which deals

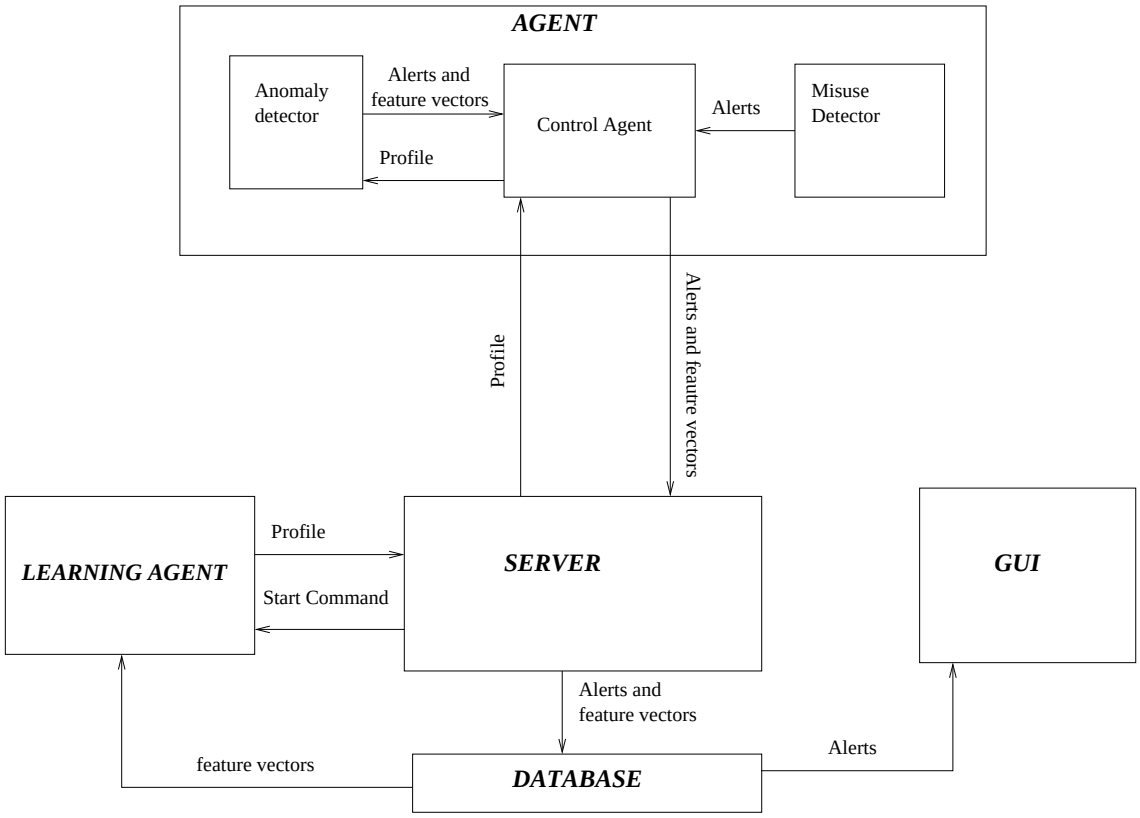


Figure 5.1: Anomaly detection process

with learning is called learning agent and is responsible for generating the normal profile.

The profile generated by the learning agent will be used for detecting deviations in observed patterns of network activity. This detection can be done either at the server or at the agents. Since detection requires considerable amount of processing, the later option is more desirable as it will lead to scalability.

Figure 5.1 shows the flow of data in Sachet. At the agent, the anomaly detector extracts features from network traffic. These features are the same as the ones described in Section 4.2 except for the hot_login feature which is not currently implemented. Since these features are extracted on a per-connection basis, the TCP packets needs to be reassembled into connections for extracting content based

features. The anomaly detector component at the agent reassembles the packets and extracts features for each connection. After extracting the features, it calculates the deviation of this connection from normal profile and if this deviation is greater that a threshold it raises an alert. Otherwise, it considers this connection as normal and sends the feature vector to the control agent.

The control agent receives alerts from the anomaly detector as well as from the misuse detector. If both these detectors raise an alert for the same connection, the control agent gives priority to the misuse detector and passes its alert to server, ignoring the alert from the anomaly detector. If the misuse detector alone raises an alert for some connection and the anomaly detector declares that connection as normal, the control agent again gives priority to the misuse detector and considers this connection as anomalous; hence it does not send the feature vector for such a connection to the server. In either of these cases, the misuse detector has more priority as its alerts are considered to be more reliable than those of the anomaly detector. Finally, the control agent sends these alerts and feature vectors to the server.

At the server, the alerts and feature vectors received from agents are saved in a database. The alerts are shown to the user through the Sachet console; the feature vectors are used by the learning agent to construct the profile. The server periodically requests the learning agent to construct the profile. When the learning agent receives such a request, it connects to the database and fetches the most recent alerts as well as the previously stored synopsis. It first updates the synopsis with the new data using the stream handling technique implemented at the learning agent and then applies the learning algorithm on this synopsis. The result of learning (the normal profile) is then passed on to the server. The server distributes this profile to the agents and the agents use the new profile for detection from that point.

Since the profile is generated from feature vectors, the agents do not have a profile initially and hence they cannot do anomaly detection until feature vectors are gathered for some sufficient amount of time, and the profile is constructed and given to them. During this period they declare all connections as normal. Another possible option is to initialize the agents with a profile constructed from artificial

normal data. But doing this may result in a lot of false positives and false negatives initially. In any case, we believe that the system stabilizes after some time.

In this design, the learning agent is an optional component and the user may decide not to install a learning agent. In that case, the profile cannot be constructed and hence there is no use of extracting features at the agents. Therefore, if the learning agent is not installed, the server will instruct the agents to stop anomaly detection. Even if the learning agent is installed, the user may wish to turn off the entire anomaly detection in Sachet. This option is provided through the console; the console passes such a request to the server and the server instructs all the agents to stop anomaly detection. The option to start the anomaly detection in Sachet is implemented similarly.

5.2 Implementation

In this section, we describe some important implementation issues in feature extraction followed by the implementation of the proposed anomaly detection scheme at each of the Sachet components. We also describe the changes made to the Sachet protocol for implementing this scheme in Sachet.

5.2.1 Feature Extraction

All the features except 'hot_login', described in Section 4.2, are extracted in Sachet. These features can be divided into general, time-based, host-based and content-based features. They are extracted for each connection observed by the anomaly detector at the agents. In the case of connection-less protocols like UDP and ICMP each packet is considered as a separate connection. In the case of TCP, the byte streams in both directions of a connection must be reassembled for extracting content-based features. For this purpose, an open source software called *tcptrace* [5] is used at each agent. Tcptrace has a plug-in architecture in which each plug-in provides a standard set of functions that are called for each new packet, for the first packet of each connection and when a connection is closed. Feature extraction is implemented as a plug-in of *tcptrace* in which information about connections is maintained in a

time sorted linked list.

Features are extracted for each connection when the connection is closed. A potential problem with this approach is that connections may last for several hours or even days. To overcome this problem, features are extracted either when the connection is closed or when 15 minutes have elapsed since the connection is initiated and is still not closed. In the following subsections, we describe the extraction of the four kinds of features.

General features

General features include some common information about a connection like destination port number, protocol, number of bytes transferred in both directions, duration of connection etc. These features can be directly extracted from packet headers. The values of these features are obtained from the information maintained by *tcptrace* for each connection.

Time-based features

Time-based features are extracted by taking a two second time window into consideration. The values of these features are derived by inspecting all connections in the past two seconds. Hence, the information about a connection whose features have been extracted cannot be thrown away as it may be required to extract the time-based features of a future connection. Most of these features are defined as percentage of connections in the past two seconds that have a common property (such as same destination port or IP address), and have a value between 0 and 1.

Host-based features

Host-based features are derived from the past 100 connections to the same host as the current destination host. But if 100 connections are not available at that point, as many as available are taken into consideration. The 100 most recent connections are found by traveling the doubly linked list of connection information. Host-based features involve percentages and have a value between 0 and 1.

- *Content-based features*

Content-based features are extracted from the payload of the packets by analyzing application layer protocols. In Sachet, these features are extracted for four protocols: telnet, FTP, HTTP and SMTP. The entire data transferred in a connection in both directions is required in sequence, to extract these features. Since TCP packets can appear out of order, the packets must be reassembled to obtain the two streams in a connection. This reassembling is done by *tcptrace* for the four protocols mentioned above. For all other protocols, *tcptrace* is instructed to ignore the payload of the corresponding packets. These features are extracted separately for each of the four protocols. Depending on the destination port number, the protocol is identified and the appropriate function is called. The extraction of features for the four protocols is described below. In all these protocols except HTTP, a state based analysis of the data is done by examining the client-to-server and the server-to-client streams simultaneously. But if an inconsistency in state is found at any point, we assume that some packets are lost and process the two streams separately.

Telnet

Data from telnet connections contains telnet control codes and console control codes. These codes should be removed to get the data generated by the user. The telnet control information includes option negotiations and sub negotiations identified by special codes defined by the telnet protocol, and it always starts with the IAC code. Console codes are used to format and present data at the remote terminal. They are either special ASCII characters which do not appear in the data, or sequence of bytes starting with an escape sequence. After removing these codes, we have the commands typed by the user in one stream, and the responses from remote terminal in the other stream.

Most of the features are extracted by looking at the commands typed by the user and the response of the remote system. Since these depend on the operating system, the first task in the extraction process is to determine the server side operating system. All Windows telnet servers have the word “microsoft" in their welcome message which is printed as the first line. The server side operating system is assumed to be Windows if this word is found; otherwise it is assumed to be a

flavour of UNIX.

The login related features are extracted as follows: if there is a “Login incorrect" message or a “Logon failure" message immediately following the login and password prompts in a telnet connection, the login attempt is considered as a failure. If no such message is found after the login prompt, the login attempt is considered as a success. The login attempt is also considered as successful if the login prompt is not found. The ‘num_failed_logins’ feature is set to the number of failed login attempts found in this connection. The ‘logged_in’ feature is set to 1 if a successful login attempt is found in the connection. The ‘is guest’ feature is set to 1 if the login name is “guest".

The root-attempt related features are extracted only for UNIX based systems. If at least one *su* command is found in the client to server data, the ‘root attempted’ feature is set to 1. To find whether an *su* attempt is a success or failure, the response of the server immediately after the *su* command is observed. If this response starts with “su:" at the beginning, the corresponding attempt is considered as a failure. Otherwise it is considered as a success. The ‘num root accesses’ feature is calculated as the number of commands typed as root.

The ‘hot’ feature gives the number of accesses to system directories and creation and execution of programs. It is incremented whenever compiler and linker commands like ‘gcc’ and interpreter commands like perl, java and awk are present in the client to server data stream of the connection. The ‘compromised’ feature gives the number of ‘file not found’ errors. The ‘num file creations’ feature is incremented whenever commands like cp, mv are found or the command has the redirection symbol ‘>’. The ‘num access files’ command gives the number of accesses to system files. It is incremented if any of the commands has a system folder path as its argument.

FTP

FTP uses the telnet protocol in the control connection. The telnet control codes are first removed from the ftp data before doing further processing. The ftp session starts after the server sends the 220 code to the client, which says that the server is ready for accepting requests. Every command from client gets at least one 3 digit response code from server. The first digit in the response code indicates the result

of the command. The first digit can be a number from 1 to 5. 4 and 5 indicate error conditions, 1 indicates that another response is being sent, 2 indicates success, and 3 indicates that the server is expecting another command from client. If reply code 220 is found in the data from server to client, we can associate commands in the client to server direction with responses in the server to client direction; otherwise we cannot associate commands with responses. At first we assume that we have the complete data transferred in the connection and we process the two streams simultaneously by matching the requests with responses and maintaining states. But if there is an inconsistency in state while processing the requests and responses in the protocol, we assume that some packets are missing. In this case we process the two streams separately.

The 'USER' command in the client to server data indicates a login attempt. A reply code starting with 4 or 5 indicates that this attempt is a failure. If the reply code starts with 3, the server is prompting for a password and if it is 2, the login attempt is a success. The commands 'RETR', 'STOR' and 'STOU' create files either on the local machine or on the remote machine and hence the 'num file creations' feature is incremented when these commands get a reply code starting with 2. Access to system folders is incremented if any of these commands specify a path containing a system directory as its argument. The 'num outbound commands' feature is incremented for each occurrence of the 'SITE' command. The 'logged in' feature is set to 1 if at least one reply code starting with 2 is found. The 'num_compromised' feature is incremented by whenever a reply code starting with 4 or 5 is found.

SMTP

The reply codes of SMTP follow the same format as that of FTP. The client starts issuing commands only after receiving the 220 reply code from server. If this code is not found then the server to client data is ignored. Every time a reply code starting with 4 or 5 is found, the number of compromised conditions is incremented by 1. If at least one reply code from server starts with a 2 then the 'logged in' feature is set to 1. The other features are not relevant for SMTP.

HTTP

In the case of HTTP the client to server data is processed independently of the

server to client data. The 'num compromised' feature is set to the number of 'file not found' replies plus the number of replies indicating failure, found in the server to client data of the connection. The 'logged in' feature is set to 1 if there is at least one reply code indicating success. The 'num access files' and 'hot' features are extracted by examining the arguments to the GET, HEAD and POST requests in the client to server data. If the path in the argument to these methods contains a system folder then the values for these features are incremented.

5.2.2 Changes to Sachet protocol

Many new messages have been added to the protocol to implement the anomaly detection scheme in Sachet. These messages are required to transfer feature vectors from agent to server and to transfer the profile from learning agent to server and from server to agent. In the case of the profile, a single message cannot hold the entire profile as there is an upper limit on the size of a UDP datagram. To overcome this drawback, the profile is sent in multiple messages with a 'more' flag in the data part of the message along with a piece of the profile. A value of 1 for this flag indicates that at least one more message containing the remaining part of the profile can be expected and a value of 0 indicates that the profile is completely transferred to the destination after this message and no more profile messages will follow.

5.2.3 Agent

The anomaly detector is implemented as a plug-in of *tcptrace* which runs as a separate process. The control agent interacts with the anomaly detector and the misuse detector and receives data from both of them. As stated already in Section 5.1, the misuse detector has priority over the anomaly detector. The control agent should correlate the alerts and feature vectors from anomaly detector with the alerts from misuse detector to implement this priority. For this purpose, the 4-tuple and timestamp of the alerts from misuse detector are saved in a list. When the control agent receives an alert or a feature vector from the anomaly detector, it compares the 4-tuple and timestamp of this data with the ones saved in the list. If a match is

found, this alert or feature vector is ignored; otherwise it is sent to the server. Note that if the data from the anomaly detector, for a particular connection, reaches the control agent before the data from the misuse detector, the above implementation will fail. But this will not generally happen because the misuse detector works on a per packet basis and hence it generates the alerts much faster than the anomaly detector.

The agent receives the profile from server in multiple messages. On receiving the first profile message, the agent opens a temporary file and saves the contents of this message in the file. For subsequent profile messages, it appends the contents of the message to the temporary file. If the 'more' flag is set to 0 which indicates that this message is the last profile message, the agent deletes the old profile and saves the contents of the temporary file as the new profile. It then deletes the temporary file and restarts the anomaly detector. The anomaly detector component at the agent is not started until the server sends a 'start' message. If the learning agent is not installed or if anomaly detection is turned off, the server will not send this message and anomaly detection will not be turned on at any agent.

5.2.4 Server

The server receives the profile from the learning agent in a piecemeal fashion and saves it in a file. It cannot interpret the profile and hence the learning algorithm can be changed at the learning agent, resulting in a new profile format, without making any changes to the server or to the protocol. The server sends the profile in a piecemeal fashion to all the agents that are alive at the time it received the profile from the learning agent. It also sends the profile to agents immediately after the authentication, irrespective of whether the agent has this profile or not. In this way the latest profile is maintained at all the agents.

The server identifies the learning agent among the agents using the 'type' field present in the information saved for each agent in the database. At the startup time, if the server cannot find a learning agent, it will not instruct the agents to start the anomaly detector. Anomaly detection can be turned off from the console even if the learning agent is present in Sachet. The server maintains a variable in the

configuration file which indicates whether anomaly detection is currently turned on or not. Again, if the server finds out at the startup time that anomaly detection is turned off, it will not instruct the agents to start the anomaly detector. In this way the entire anomaly detection scheme can be enabled or disabled from the server.

5.2.5 Learning agent

The learning agent starts the learning algorithm in a separate thread upon request from the server. Before executing the learning algorithm, this thread connects to the database and fetches feature vectors and synopsis using the ODBC interface. The data source name, username and password required for this purpose are provided to the learning agent through a configuration file. After fetching the required data from the database, the learning algorithm is applied on this data. When the learning is completed, the learning agent sends the result to the server. If an error occurs at any point during this process, *e.g,* connection to the database may fail, a message indicating the type of failure is sent to the server.

5.2.6 Console

The console has been enhanced with a separate screen for the learning agent in which it shows some basic information about the learning agent including its current status. The same screen also has several buttons through which the user can start or stop the learning of profile, and disable or enable the entire anomaly detection process in Sachet. When the start button for learning is pressed, the console prompts the user for parameters to the learning algorithm and sends them to the server. The user has to provide all the parameters in the form of a string in which consecutive parameters are separated by spaces. This string is passed on from console to server and finally to the learning agent where the parameters are extracted from this string.

Chapter 6

Results

In this chapter we present the results of evaluating the anomaly detection in Sachet on the 1999 DARPA data, a benchmark dataset generated for the purpose of evaluating intrusion detection systems. We first give a brief description of the 1999 DARPA data and the evaluation criteria used. We then describe the experimental methodology, and finally present the results.

The 1999 DARPA data [1] was generated for the 1999 DARPA intrusion detection evaluation [19] conducted by MIT Lincoln Laboratory, using a test bed that simulated an existing military network. This data contains more than 200 instances of 58 attacks types launched against victims using both UNIX and Windows NT. The data also contains a wide variety of background traffic to test the false alarm rate of the system being evaluated. It was collected over a period of five weeks with five days per week, in which the first and third weeks of data is free of attacks and the second, fourth and fifth weeks of data contains attacks along with normal activities. The data collected includes sniffed network traffic, Solaris Basic Security Module (BSM) audit data and Windows NT audit event logs along with nightly listing of all files in the system and dumps of security-related files.

Since Sachet is a network-based IDS, we used only the network data for the evaluation purpose. This network data was obtained by sniffing at two points in the simulated test bed, the inside router and the outside router. Accordingly, there are two tcpdump files called the inside data and the outside data for each of the 25 days

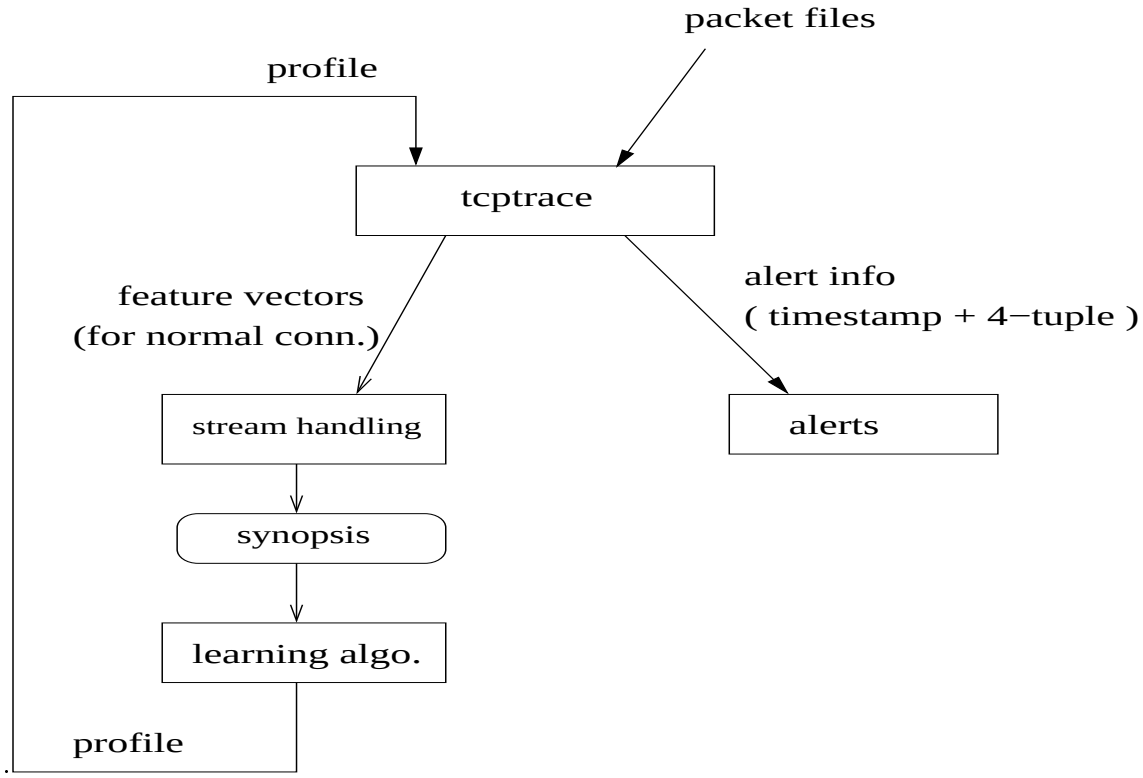


Figure 6.1: Experimental setup

during which the data was collected, except for one day on which the inside data is not available.

The criteria used for evaluation are false alarm rate and detection rate. The false alarm rate is the percentage of false alerts with respect to the total number of connections. The detection rate is measured in terms of number of attacks and not number of connections. The detection rate is defined as the percentage of the total number of attacks, detected by the system.

The experiments were conducted in a simulated test-bed where packet files from the 25 days of data were processed by the anomaly detector (*tcptrace* with our plug-in). Figure 6.1 depicts the experimental setup. Note that the misuse detector was not used in these experiments. This is because all the attacks in the test data are quite well-known now and Snort already has signatures for most of these

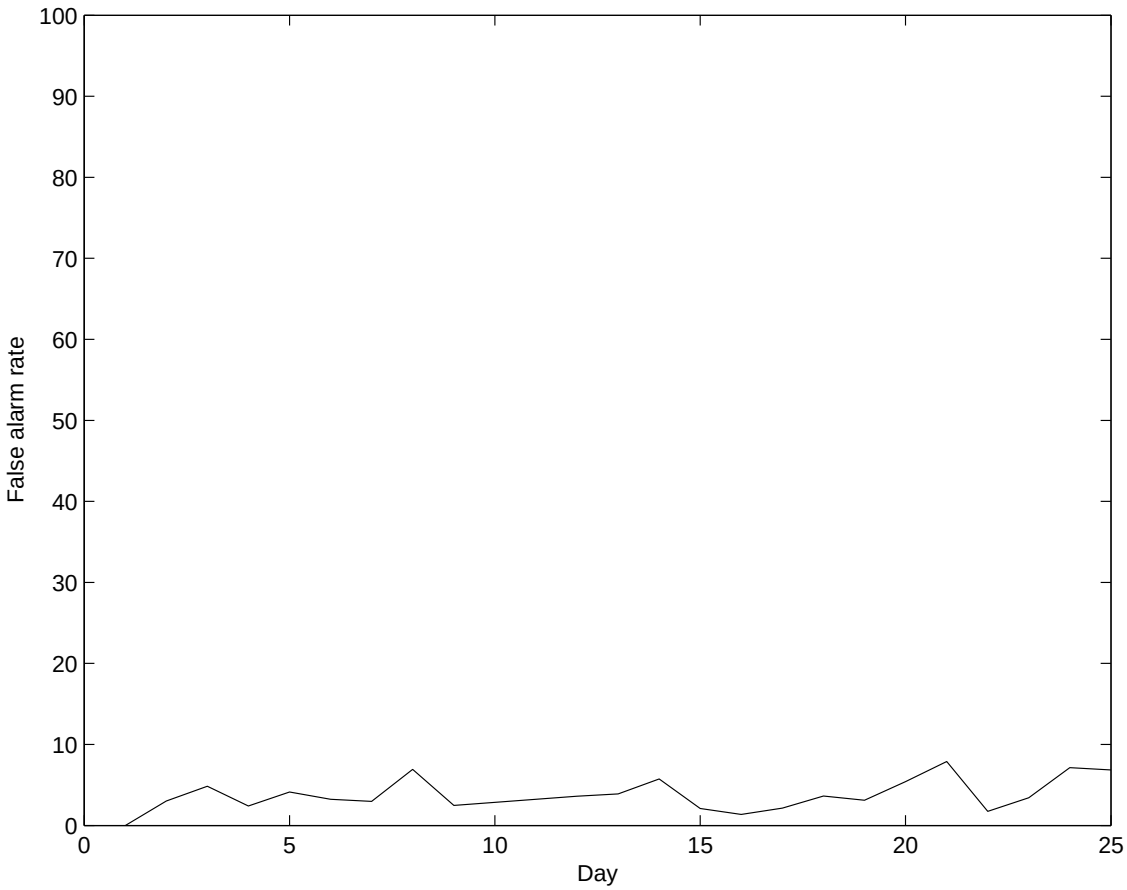


Figure 6.2: False alarm rates

attacks. The *tcptrace* utility was modified by us to take the profile and a packet file in tcpdump format as input and produces alerts for connections flagged as attacks, and feature vectors for normal connections. The alerts were later analyzed manually to calculate the detection rates and false alarm rates. After processing each day's data, the normal feature vectors were processed by the stream handling algorithm and the synopsis was updated as in the actual Sachet system. The learning algorithm was then applied on this synopsis and the profile was generated, which was used by *tcptrace* for processing the next day's traffic. Initially the profile was empty and hence for the first day all connections were assumed to be normal and no alerts were generated.

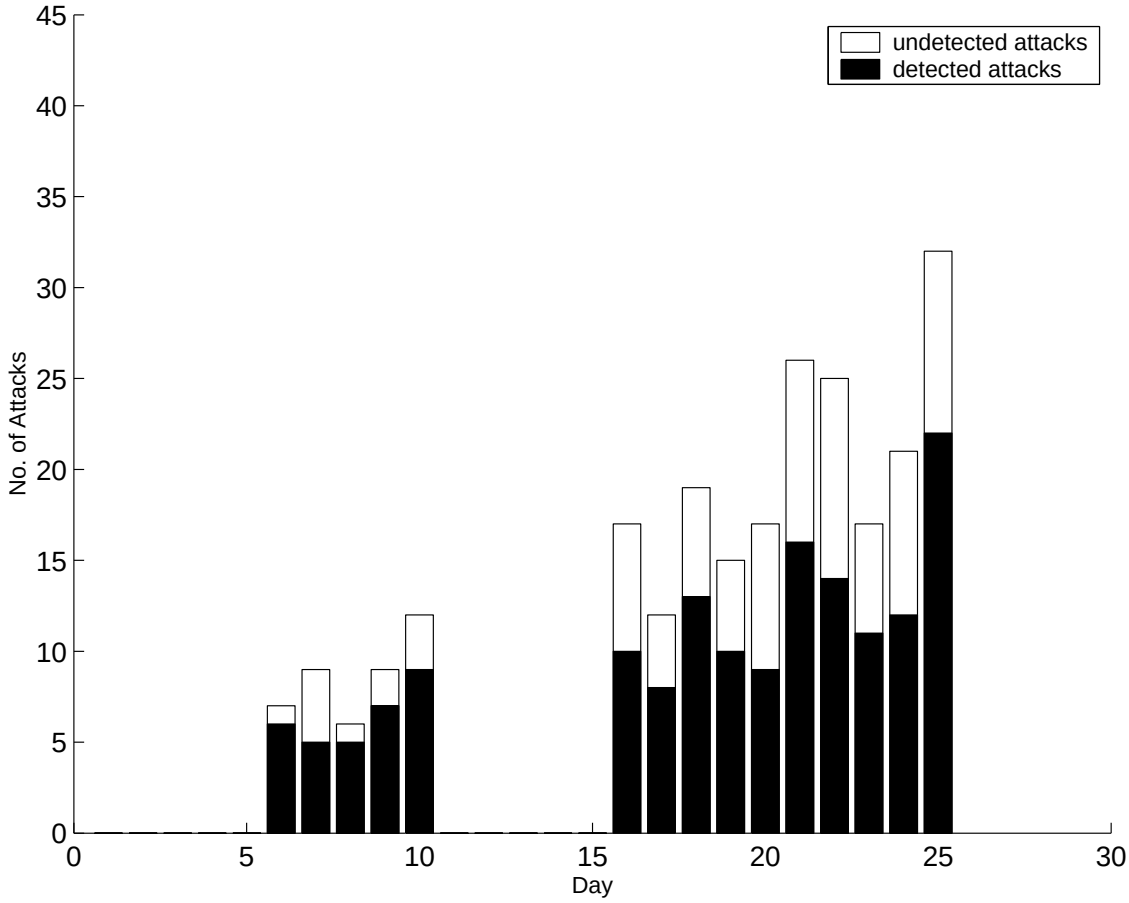


Figure 6.3: Overall attack detection

The false alarm rates for the 25 days are plotted in Figure 6.2. The average false alarm rate is 3.92%.

Figure 6.3 shows the number of attacks present and the number of attacks detected for the data of each day. The average detection rate is 66.46%.

The detection rates for specific classes of attacks are shown in the Figures 6.4 to 6.7. The average detection rate is 65.5% for DoS attacks, 70.47% for Probe attacks, 66.17% for R2L attacks and 75.48 for U2R attacks. Among the four classes, the detection rates for Probes and U2R attacks are quite high as expected. This is due to the fact that the host-based and content-based features capture the characteristics

of these attacks to a very large extent. Even stealthy probes that have a large time delay between successive attempts are successfully detected due to the host-based features. The time-based features are mostly suitable for DoS attacks and Probes. The detection rate for DoS attacks is low because most content-based DoS attacks like *back* [2] and *crashiis* [2] have not been detected.

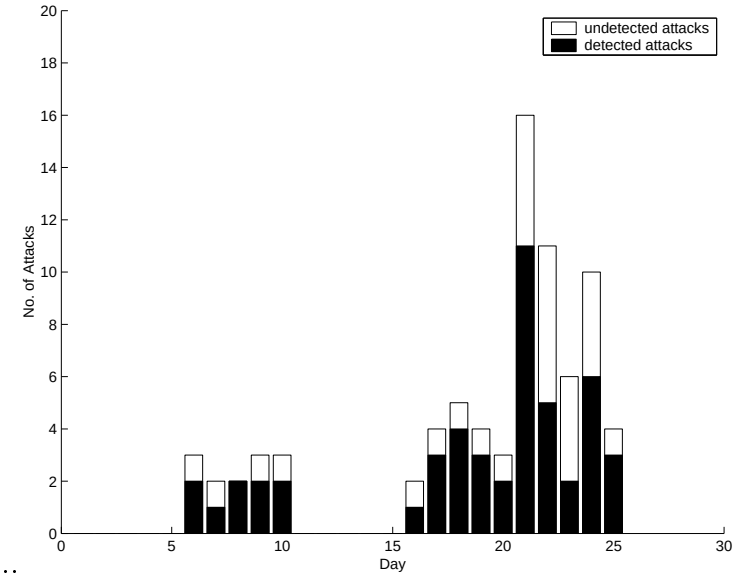


Figure 6.4: DoS attack detection

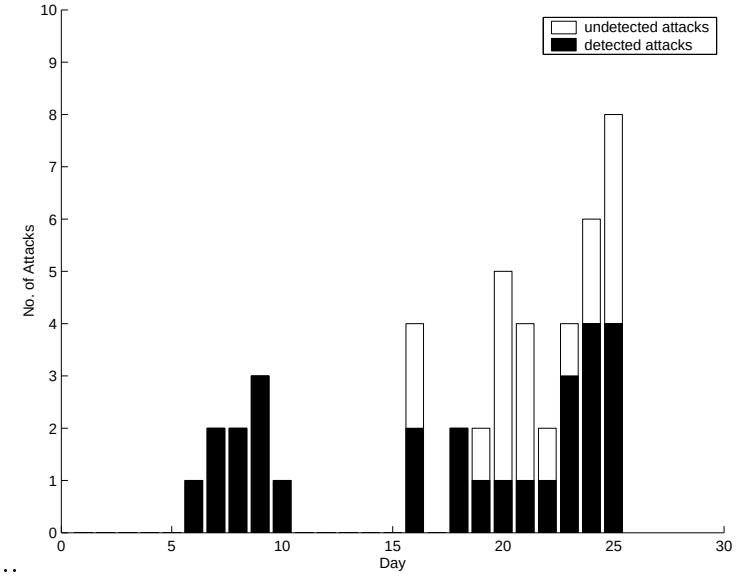


Figure 6.5: Probe attack detection

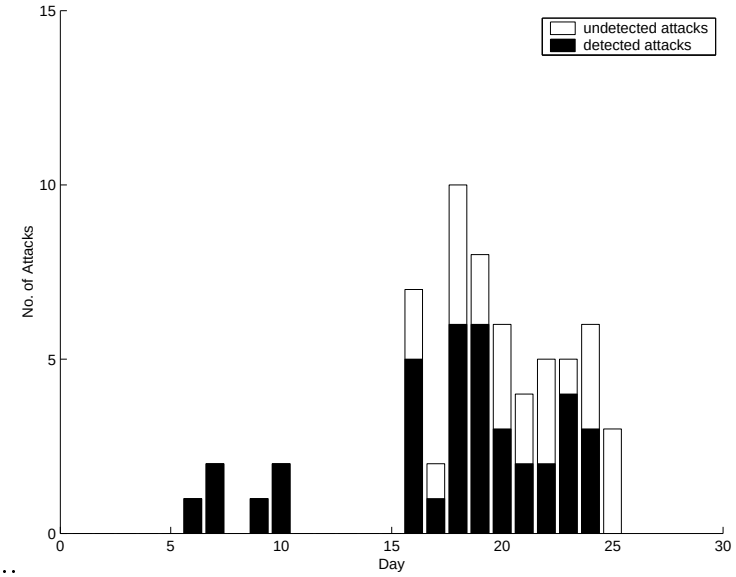


Figure 6.6: R2L attack detection

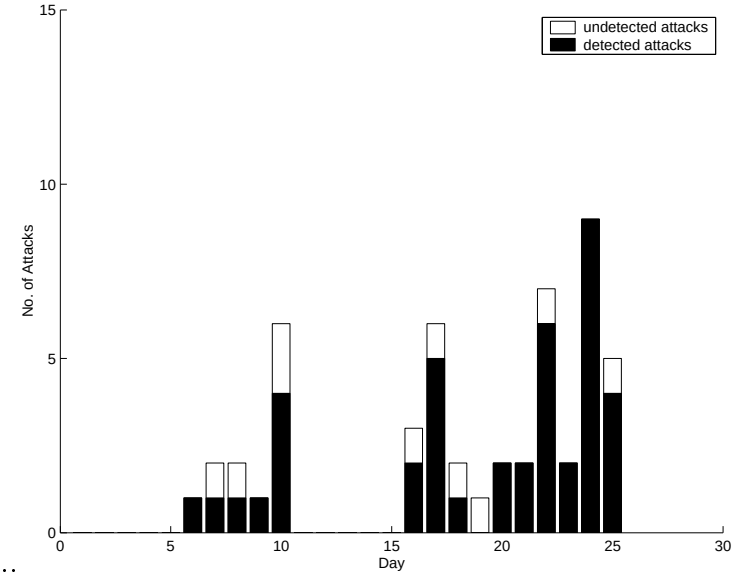


Figure 6.7: U2R attack detection

Chapter 7

Conclusions and Future Work

We have developed a real-time network-based anomaly detection scheme for the Sachet Intrusion Detection System using unsupervised learning and stream handling techniques. We considered various learning and stream handling algorithms and evaluated them on a benchmark dataset. We implemented the best performing techniques in Sachet by modifying the components of Sachet. Finally, we tested this scheme on a benchmark dataset of size 20GB containing 200 instances of 58 attack types and presented the results. The average false alarm rate is 3.92% and the average detection rate is 66.46%.

This scheme is very practical and scalable, involves very little human intervention and shows good detection rates and fairly low false alarm rates. But the false alarm rates have to be brought down even further to make the system more reliable. Also, there are several parameters involved in this system like the gamma value for the learning algorithm, reservoir size, threshold for deviation etc. that need to be tuned properly for maximum accuracy of the system.

In future, our anomaly detection scheme can be extended to include alert correlation and signature generation. The anomaly detector works on individual connections and generates one alert for each anomalous connection. Many attacks comprise of more than one connection and attacks like Denial of Service sometimes comprise of even thousands of connections. Hence the anomaly detector in Sachet may generate multiple alerts for a single attack. The present work can be extended to analyze

these alerts and identify groups of alerts that are generated from the same attack and present the entire group as a single alert to the administrator.

Anomaly detection is generally used to detect unknown attacks, but once an attack is identified it is always desirable to detect it in future using the signature based method due to its reliability. This can be done if a signature is generated for this new attack. This generation can be done in two ways: the system itself can generate a signature and add this signature to the existing database or the system can provide useful information about the attack to the user and the user can manually generate the signature.

Another possible improvement in this anomaly detection scheme is in the information provided for each alert. Right now, only the UDP or TCP 4-tuple and the timestamp are available; there is no indication about the type of attack. But we believe that by looking at the feature vector, one can provide more information about the alert than just the timestamp and the 4-tuple.

Appendix A

New Messages Included in the Sachet Protocol

The new messages added to the Sachet protocol for implementing anomaly detection in Sachet IDS are described in this appendix along with their format. The packet format for these messages is shown in Figure 3.2. Here, we present only the format of the data part of these messages. The numbers in the brackets beside the fields in the format indicate their size in bytes. Strings have variable size and are terminated by a NULL character while including in any message.

CONN_FEATURES: This message is used by the agent to send feature vectors of normal connections to the server. Multiple feature vectors can be sent in a single message. Each feature vector has a four byte timestamp and a string that is obtained by concatenating the values of features with spaces in between and ending in a NULL character. The format is shown below:

CONN_FEATURES | count (2) | timestamp-1 (4) | feature_vector-1 (string) | timestamp-2 (4) | feature_vector-2 (string) | ...

where 'count' is the number of (timestamp, feature_vector) tuples present in this message. The reply from server to this message contains the reply code CONN_FEATURES_REPLY with an empty data field.

LEARNING_RESULT: This message is used to transfer the profile from the learning agent to the server and then from the server to the agent. Generally, the

profile cannot be sent in a single message and requires multiple such messages. The profile is handled by the server and the agents as a text file and while transfering it, they include consecutive lines from this file in each message till the entire file has been transfered. The file is reconstructed at the receving end from these messages. The format is as follows:

LEARNING_RESULT | more (2) | count (2) | line-x (string) | ... | line-y (string)

where 'more' field indicates whether more messages will follow for this transfer and 'count' contains the number of lines of profile present in this message.

The reply to this message contains the message code LEARNING_RESULT_REPLY with an empty data field.

START_AD: The server instructs the agent to start the anomaly detector using this message. The data field of this message is empty.

The reply to this message contains the message code START_AD_REPLY and the data field contains a 2-byte reply code indicating whether the operation is a success or failure.

STOP_AD: The server instructs the agent to stop the anomaly detector using this message. The data field of this message is empty.

The reply to this message contains the message code STOP_AD_REPLY and the data field contains a 2-byte reply code indicating whether the operation is a success or failure.

START_LEARNING: Using this message the server instructs the learning agent to learn the profile. The data part of this message contains a NULL terminated string. The parameters to the learning algorithm are provided by the user in the form of this string through the console. This string is passed on to the learning algorithm where individual parameters are extracted.

START_LEARNING | parameters (string)

The reply to this message contains the message code START_LEARNING_REPLY with a reply code in the data part indicating whether the learning has started or not.

STOP_LEARNING: Using this message the server instructs the learning agent to stop the learning algorithm. The data part of this message is empty.

The server sends this message only when the user presses the STOP button on the console but never on its own.

The reply to this message contains the message code STOP_LEARNING_REPLY with an empty data field.

LEARNING_FAILED: The learning agent sends this message to the server when it encounters any problem during the learning phase. For example, database connectivity may fail in the middle while fetching the feature vectors, memory allocation may fail if the input data is too large etc. The datapart of this message contains a 2-byte code indicating the cause for the failure if possible. Otherwise it contains a zero.

LEARNING_FAILED | code indicating reason (2)

The reply to this message from the server contains the message code LEARNING_FAILED_REPLY with an empty data part.

Bibliography

[1] 1999 darpa intrusion detection evaluation data. World Wide Web `http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html`.

[2] Intrusion detection attacks database. World Wide Web, `http://www.ll.mit.edu/IST/ideval/docs/1999/attackDB.html`.

[3] Kdd cup 1999 data. World Wide Web, `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[4] Snort, open source network intrusion detection system. World Wide Web, `http://www.snort.org`.

[5] Tcptrace, a tool for analyzing tcp dump files. World Wide Web, `http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html`.

[6] Barbara, D., Couto, J., Jajodia, S., and Wu, N. ADAM: a testbed for exploring the use of data mining in intrusion detection. *ACM SIGMOD Record 30*, 4 (December 2001), 15–24.

[7] Ben-Hur, A., Horn, D., Siegelmann, H. T., and Vapnik, V. Support vector clustering. *Journal of Machine Learning Research 2* (2001), 125–137.

[8] Chih-Chung Chang, and Chih-Jen Lin. LIBSVM: a library for support vector machines. World Wide Web, `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[9] Cohen, W. W. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference* (1995), Morgan Kauffman.

[10] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines (and other kernel based learning methods).* Cambridge University Press, 2000. ISBN: 0 521 78019 5.

[11] Crosbie, M., and Spafford, G. Active defense of a computer system using autonomous agents. Tech. Rep. 95-008, Department of Computer Science, Purdue University, 1995.

[12] Denning, D. E. An intrusion detection model. *IEEE Transactions on Software Engineering* (February 1987).

[13] Gibbons, P. B., and Matias, Y. New sampling-based summary statistics for improving approximate query answers. *ACM SIGMOD* (1998).

[14] Guan, Y., Ghorbani, A. A., and Belacel, N. Y-means: A clustering method for intrusion detection. In *Canadian Conference on Electrical and Computer Engineering, IEEE CCECE* (May 2003), vol. 2, pp. 1083–1086.

[15] Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. Clustering data streams. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)* (2000), pp. 359–366.

[16] Hamamoto, Y., Uchimura, S., and Tomita, S. A bootstrap technique for nearest neighbour classifier design. *IEEE Transactions on Patrern Analysis and machine Intelligence 19*, 1 (January 1997).

[17] Javitz, H. S., and Valdes, A. The SRI ides statistical anomaly detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (1991).

[18] Krishna, K. *Hybrid Evolutionary algorithms for Supervised and Unsupervised learning.* PhD thesis, Department of Electrical Engineering, Indian Institute of Science, Bangalore, India, August 1998.

[19] Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., and Das, K. Analysis and results of teh 1999 darpa off-line intrusion detection evaluation, 2000.

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

[21] Mukkamala, S., Janoski, G., and Sung, A. Intrusion detection using neural networks and support vector machines. In *Proceedings of the 2002 International Joint Conference on Neural Networks* (May 2002), vol. 2, pp. 1702–1707.

[22] Ryan, J., Lin, M.-J., and Miikkulainen, R. Intrusion detection with neural networks. *Advances in Nueral Information processing Systems*, 10 (1998). Cambridge, MA:MIT Press.

[23] Seo, S., Bode, M., and Obermayer, K. Soft nearest prototype classification. *IEEE Transactions on Neural Networks 14*, 2 (March 2003).

[24] Srikant, R., and Agrawal, R. Mining generalized association rules. In *Proceedings of the 21st VLDB Conference* (Zurich, Switzerland, 1995).

[25] Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 1 (March 1985), 37–57.

[26] Weber, D. A taxonomy of computer intrusions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

[27] Wenke Lee et al. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security* (2000).