CHAPTER 4

PARALLEL CLASSIFIER USING DECISION TREE

4.1 INTRODUCTION

In view of increasing complexity in the attacks aimed at communication infrastructure especially that supports real-time application, there is a need for computer aided model building to automate various steps in the intrusion detection process. Model generation is one of the most difficult problems in a communication and computer protection system.

Intrusion Detection System is a key element in providing communication security to real-time applications. The design and implementation of IDS to support the communication with stringent constraints on QOS parameter requirements of real-time applications is challenging task. Even innocent looking and very commonly occurring few packet drops due to load on the shared networks because of unwanted traffic like DOS attack, will lead to dangerous consequences in real-time applications. This chapter presents a parallel classifier model for intrusion detection which will guarantee essential and stringent QOS requirements of communication infrastructure to support real-time application.

4.1.1 Real-Time Traffic Requirements

The requirements for inelastic real-time traffic may include the following (William 2002):
• **Throughput:** A minimum throughput value may be required. Unlike most elastic traffic, which can continue to deliver data with perhaps a degraded service, many inelastic applications require a firm minimum throughput.

• **Delay:** Delay must be very minimal. Depending on the type of real-time application, a maximum bound on delay is arrived over which application can not survive.

• **Delay variation:** The larger the allowable delay, the longer the real delay in delivering the data and the greater the size of the delay buffer required at receivers. Real-time interactive applications, such as teleconferencing, may require a reasonable upper bound on delay variation.

• **Packet loss:** Real-time applications vary in the amount of packet loss, if any, which they can sustain.

### 4.2 ISSUES AND CHALLENGES

Communication system to support real-time system must provide transparent protection, better adaptation to changing environments, assured availability of critical services, timing and performance guarantees and the protection against disruption through naturally occurring events or malicious attack.

Most of the thrust in today’s IDS research is focused on accurate detection of attacks. Time critical real-time application demands the detection of attacks at a minimal, uniform and predictable time.
4.3 **IDS USING DATA MINING PROCESS**

The data mining process of building intrusion detection models is depicted in Figure 4.1 (Lee et al 1998). Raw data is first converted into ASCII network packet information, which in turn is converted into connection level information. These connection level records contain within connection features like service, duration etc. Data mining algorithms are applied to this data to create models to detect intrusions (Grossman et al 1998). These algorithms are applied to audit data to compute models that accurately capture the actual behavior of intrusions as well as normal activities. The main advantage of this system is automation of data analysis through data mining, which enables it to learn rules inductively replacing manual encoding of intrusion.

![Figure 4.1 Data mining process of building Intrusion detection models](image-url)
4.4 FRAME WORK FOR CONSTRUCTION OF PARALLEL IDS

The constituent components of the proposed parallel integrated IDS framework are shown in Figure 4.2. In this figure, labeled attributes are the intrusion detection example data tagged with labels indicating whether it is an attack or normal traffic. Decision tree induction algorithm is a model construction algorithm which constructs classification model from pre-labeled data set. Sequential algorithm and Parallel algorithm are developed and implemented for the classification model, learned from the Decision Tree induction algorithm as shown in the Figure 4.2 as Sequential IDS and Parallel Integrated IDS respectively. This module takes unlabelled attributes and classifies into various classes of traffic type.

![Figure 4.2 Framework for construction of IDS using decision tree](image)

A classification model is represented as a decision tree. A decision tree consists of nodes, where attributes are tested. The outgoing
branches of a node correspond to all the possible outcome of the test at the node. The samples, at a nonleaf node in the tree structure, are partitioned along the branches and each child node gets its corresponding subset of samples. Every path to the leaf in the decision tree represents a classification rule. During classification, the left hand side of the rules are applied sequentially until one of them evaluates to true, and then the implied class label from the right hand side of the rule is offered as the class prediction.

4.5 DECISION TREES

Decision trees and decision rules are data-mining methodologies applied in many real-world applications as a powerful solution to classification problems (Mehmed Kantardzic 2003). Therefore, these initial few paragraphs briefly summarize the basic principles of classification. In general, classification is a process of learning a function that maps a data item into one of several predefined classes. Every classification based on inductive-learning algorithms is given as input a set of samples that consist of vectors of attribute values (also called feature vectors) and a corresponding class. The goal of learning is to create a classification model, known as a classifier, which will predict, with the values of its available input attributes, the class for some entity (a given sample). In other words, classification is the process of assigning a discrete label value (class) to an unlabeled record, and a classifier is a model (a result of classification) that predicts one attribute-class of a sample-when the other attributes are given. In doing so, samples are divided into predefined groups. Different classification methodologies are applied today in almost every discipline where the task of classification, because of the large amount of data, requires automation of the process.

In a classification model, the connection between classes and other properties of the samples can be defined by something as simple as a
flowchart or as complex and unstructured as a procedure manual. Data-
mining methodologies restrict discussion to formalized, "executable" models
of classification, and there are two very different ways in which they can be
constructed. On one hand, the model might be obtained by interviewing the
relevant expert or experts, and most knowledge-based systems have been built
this way despite well-known difficulties faced while taking this approach.
Alternatively, numerous recorded classifications might be examined and a
model constructed inductively by generalizing specific examples that are of
primary interest for data-mining applications.

The statistical approach to classification gives one type of model
for classification problems: summarizing the statistical characteristics of the
set of samples. The other approach is based on logic. Instead of using math
operations like addition and multiplication, the logical model is based on
expressions that are evaluated as true or false by applying Boolean and
comparative operators to the feature values (Rich and Knigh 1999). These
methods of modeling give accurate classification results compared to other
nonlogical methods, and they have superior explanatory characteristics.
Decision trees and decision rules are typical data-mining techniques that
belong to a class of methodologies that give the output in the form of logical
models.

A particularly efficient method for producing classifiers from data
is to generate a decision tree. The decision-tree representation is the most
widely used logic method. Decision Tree induction algorithms are supervised
learning methods that construct decision trees from a set of input-output
samples (Quinlan 1987). A typical decision-tree learning system adopts a
top-down strategy that searches for a solution in a part of the search space. It
guarantees that a simple tree, but not necessarily the simplest, will be found.
A decision tree consists of nodes where attributes are tested. The outgoing
branches of a node correspond to all the possible outcomes of the test at the node. The samples, at a nonleaf node in the tree structure, are partitioned along the branches and each child node gets its corresponding subset of samples. Decision trees that use univariate splits have a simple representational form, making it relatively easy for the user to understand the inferred model, at the same time, they represent a restriction on the expressiveness of the model. In general, any restriction on a particular tree representation can significantly restrict the functional form and thus the approximation power of the model. A well-known tree-growing algorithm for generating decision trees based on univariate splits is Quinlan's IDS with an extended version called C4.5 (Quinlan 1992). Greedy search methods, which involve growing and pruning decision-tree structures, are typically employed in these algorithms to explore the exponential space of possible models.

The ID3 algorithm starts with all the training samples at the root node of the tree. An attribute is selected to partition these samples. For each value of the attribute a branch is created, and the corresponding subset of samples that have the attribute value specified by the branch is moved to the newly created child node. The algorithm is applied recursively to each child node until all samples at a node are of one class. Every path to the leaf in the decision tree represents a classification rule. Note that the critical decision in such a top-down decision tree-generation algorithm is the choice of attribute at a node. Attribute selection in ID3 and C4.5 algorithms are based on minimizing an information entropy measure applied to the examples at a node.

The approach based on information theory insists on minimizing the number of tests that will allow a sample to classify in a database. The attribute-selection part of ID3 is based on the assumption that the complexity of the decision tree is strongly related to the amount of information conveyed.
by the value of the given attribute. An information-based heuristic selects the attribute providing the highest information gain, i.e., the attribute that minimizes the information needed in the resulting subtree to classify the sample. An extension of ID3 is the C4.5 algorithm, which extends the domain of classification from categorical attributes to numeric ones. The measure favors attributes that result in partitioning the data into subsets that have low class entropy, i.e., when the majority of examples in it belong to a single class. The algorithm basically chooses the attribute that provides the maximum degree of discrimination between classes locally. More details about basic principles and implementation of these algorithms will be given in the following sections.

To apply some of the methods, which are based on the inductive-learning approach, several key requirements have to be satisfied:

**Attribute-value description:** The data to be analyzed must be in a flat-file form—all information about one object or example must be expressible in terms of a fixed collection of properties or attributes. Each attribute may have either discrete or numeric values, but the attributes used to describe samples must not vary from one case to another. This restriction rules out domains in which samples have an inherently variable structure.

**Predefined classes:** The categories to which samples are to be assigned must have been established beforehand. In the terminology of machine learning this is called supervised learning.

**Discrete classes:** The classes must be sharply delineated - a case either does or does not belong to a particular class. It is expected that there will be far more samples than classes.
**Sufficient data:** Inductive generalization given in the form of decision tree proceeds by identifying patterns in data. The approach is valid if enough number of robust patterns can be distinguished from chance coincidences. As this differentiation usually depends on statistical tests, there must be sufficient number of samples to allow these tests to be effective. The amount of data required is affected by factors such as the number of properties and classes and the complexity of the classification model. As these factors increase, more data will be needed to construct a reliable model.

**Logical classification models:** These methods construct only such classifiers that can be expressed as decision trees or decision rules. These forms essentially restrict the description of a class to a logical expression whose primitives are statements about the values of particular attributes. Some applications require weighted attributes or their arithmetic combinations for a reliable description of classes. In these situations logical models become very complex and, in general, they are not effective.

4.6 **C4.5 ALGORITHM: GENERATING A DECISION TREE**

The most important part of the C4.5 algorithm is the process of generating an initial decision tree from the set of training samples. As a result, the algorithm generates a classifier in the form of a decision tree. It is a structure with two types of nodes i.e. a leaf indicating a class, or a decision node that specifies some test to be carried out on a single-attribute value, with one branch and subtree for each possible outcome of the test.

A decision tree can be used to classify a new sample by starting at the root of the tree and moving through it until a leaf is encountered. At each nonleaf decision node, the features’ outcome for the test at the node is determined and attention shifts to the root of the selected subtree.
The skeleton of the C4.5 algorithm is based on Hunt's Concept Learning System (CLS) method for constructing a decision tree from a set T of training samples. Let the classes be denoted as \{C_1, C_2, \ldots, C_k\}. There are three possibilities for the content of the set T.

Possibility (1): T contains one or more samples, all belonging to a single class C_j. The decision tree for T is a leaf identifying class C_j.

Possibility (2): T contains no samples. The decision tree is again a leaf but the class to be associated with the leaf must be determined from information other than T, such as the overall majority class in T. The C4.5 algorithm uses a criterion the most frequent class at the parent of the given node.

Possibility (3): T contains samples that belong to a mixture of classes. In this situation, the idea is to refine T into subsets of samples that are heading towards a single-class collection of samples. Based on single attribute, an appropriate test that has one or more mutually exclusive outcomes \{O_1, O_2, \ldots, O_n\} is chosen. T is partitioned into subsets T_1, T_2, \ldots, T_n where Ti contains all the samples in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test and one branch for each possible outcome.

The same tree-building procedure is applied recursively to each subset of training samples, so that the i-th branch leads to the decision tree constructed from the subset T_i of training samples. The successive division of the set of training samples proceeds until all the subsets consist of samples belonging to a single class.
The tree-building process is not uniquely defined. For different tests, even for a different order of their application, different trees will be generated. Ideally, it is desirable to choose a test at each stage of sample-set splitting so that the final tree is small. Since it is a compact decision tree that is consistent with the training set, it is possible to explore all possible trees and select the simplest.

The problem of finding the smallest decision tree consistent with a training data set is NP-complete. Enumeration and analysis of all possible trees will cause a combinatorial explosion for any real-world problem. For example, for a small database with five attributes and only twenty training examples, the possible number of decision trees is greater than $10^6$, depending on the number of different values for every attribute. Therefore, most decision tree construction methods are non-backtracking, greedy algorithms. Once a test has been selected using some heuristics to maximize the measure of progress and the current set of training cases has been partitioned, the consequences of alternative choices are not explored. The measure of progress is a local measure, and the gain criterion for a test selection is based on the information available for a given step of data splitting.

Suppose the task is to select a possible test with $n$ outcomes ($n$ values for a given feature) that partitions the set $T$ of training samples into subsets $T_1$, $T_2$, ..., $T_n$. The only information available for guidance is the distribution of classes in $T$ and its subsets $T_i$. If $S$ is any set of samples, let $freq(C_i, S)$ stand for the number of samples in $S$ that belong to class $C_i$ (out of $k$ possible classes), and let $|S|$ denote the number of samples in the set $S$.

The original ID3 algorithm used a criterion called gain to select the attribute to be tested which is based on the information theory concept: entropy.
The relation (4.1) gives the computation of the entropy of the set S (bits are units):

\[
\text{Info}(S) = -\sum_{i=1}^{k} \left( \left( \text{freq}(C_i, S) \right) \log_2 \left( \frac{\text{freq}(C_i, S)}{|S|} \right) \right)
\]

(4.1)

Now consider a similar measurement after T has been partitioned in accordance with n outcomes of one attribute test X. The expected information requirement can be found as the weighted sum of entropies over the subsets as given in equation (4.2):

\[
\text{Info}_x(T) = -\sum_{i=1}^{n} \left( \left| T_i \right| \text{Info}(T_i) \right)
\]

(4.2)

The quantity given in equation (4.3)

\[
\text{Gain}(X) = \text{Info}(T) - \text{Info}_x(T)
\]

(4.3)

measures the information that is gained by partitioning T in accordance with the test X. The gain criterion selects a test X to maximize Gain(X), i.e., this criterion will select an attribute with the highest info-gain.

4.7 EXPERIMENT SETUP AND PERFORMANCE EVALUATION

The KDD Cup 1999 Intrusion detection contest data (KDD cup 99 Intrusion detection data set) is used in these experiments. These data were prepared by the 1998 DARPA Intrusion Detection Evaluation program by MIT Lincoln Labs (MIT Lincoln Laboratory). Lincoln labs acquired nine weeks of raw TCP dump data. The raw data was processed into connection records, which consist of about 5 million connection records.

A connection is a sequence of TCP packets starting and ending at some well defined times between which data flows to and from a source IP
address to a target IP address under some well defined protocol. Each connection is labeled as either normal or an attack with exactly one specific attack type. Each connection record consists of about 100 bytes.

4.8 KDD CUP 1999 DATA SET

In its broadest definition, a computer attack is any malicious activity directed to a computer system or the services it provides. Examples of computer attacks are viruses, use of a system by an unauthorized individual, denial-of-service by exploitation of a bug or abuse of a feature, probing of a system to gather information, or a physical attack against computer hardware. A subset of the possible types of computer attacks were included in the 1998 DARPA intrusion detection system evaluation, including: (1) Attacks that allow an intruder to operate on a system with more privileges than are allowed by the system security policy, (2) Attacks that deny someone else access to some service that a system provides, or (3) Attempts to probe a system to find potential weaknesses.

The following paragraphs provide some examples of many ways that an attacker can either gain access to a system or deny legitimate access by others (Kristopher Kendall 1999).

**Social Engineering:** An attacker can gain access to a system by fooling an authorized user into providing information that can be used to break into a system. For example, an attacker can call an individual on the telephone impersonating a network administrator in an attempt to convince the individual to reveal confidential information (passwords, file names, details about security policies etc.). Or an attacker can deliver a piece of software to the user of a system which is actually a trojan horse containing malicious code that gives the attacker system access.
**Implementation Bug:** Bugs in trusted programs can be exploited by an attacker to gain unauthorized access to a computer system. Specific examples of implementation bugs are buffer overflows, race conditions, and mishandled temporary files.

**Abuse of Feature:** There are legitimate actions that one can perform that when taken to the extreme can lead to system failure. Examples include opening hundreds of telnet connections to a machine to fill its process table, or filling up a mail spool with junk e-mail.

**System Misconfiguration:** An attacker can gain access because of an error in the configuration of a system. For example, the default configuration of some systems includes a “guest” account that is not protected with a password.

**Masquerading:** In some cases it is possible to fool a system into giving access by misrepresenting oneself. An example is sending a TCP packet that has a forged source address that makes the packet appear to come from a trusted host.

The benchmark KDD Cup 1999 Intrusion detection contest data set contains 24 attack types. These attacks fall into four main categories:

- Denial of service (DoS)
- Remote to user (R2L)
- User to root (U2R)
- Probing
4.8.1 Denial of Service

In this type of attack an attacker makes some computing or memory resources too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. Examples are Apache2, Back, Land, Mailbomb, SYN Flood, Ping of death, Process table, Smurf, Teardrop.

**Apache2**: DoS attack performed against an apache web server where an adversary submits an http request with several http headers. In theory if the server receives too many of such requests it will slow down the functionality of the web server and eventually crashes. This attack denies the web service temporarily; the service can be regained with system administrator’s intervention.

**Back**: DoS attack performed against an apache web server where an adversary submits an URL request with several front slashes. While trying to process these requests, the server’s service becomes unavailable for legitimate users. This attack denies the web service temporarily. The service can be regained automatically.

**Land**: DoS attack performed against TCP/IP implementations where an adversary sends a spoofed SYN packet where the source and destination IP address are the same. In theory it’s not possible to have the same destination address as the source address. The adversary targets the badly configured networks and uses the innocent machines as zombies for performing distributed attacks. This attack can be prevented by carefully configuring the network, which prevents requests containing the same source and destination IP addresses.
**Mail bomb:** DoS attack performed against the server where an adversary floods the mail queue, possibly causing failure. The adversary tries to send thousands of mails to a single user. This attack denies the service permanently. The service can be regained by the system administrator intervention, blocking the mails coming from or to the same user within a short period of time can prevent the attack.

**SYN Flood (Neptune):** DoS attack performed against every TCP/IP implementations where an adversary utilizes the half open TCP connections to flood the data structure of half open connections on the innocent server causing to deny access to legitimate requests. This attack in some cases can cause permanent failure. The service can be regained automatically. Looking for a number of simultaneous SYN packets coming form the same host or unreachable host in a given short period of time can prevent this attack.

**Ping of Death (PoD):** DoS attack performed against older versions of operating systems where an adversary tries to send an oversized IP packet, and the system reacts in an unpredictable manner, causing crashing, rebooting and even freezing in some cases. This attack causes temporary failure of services. Looking for Internet Control Message Protocol (ICMP) packets that are longer than 64000 bytes and blocking them is the way to prevent this attack.

**Smurf:** DoS attack performed against all the systems connected to the Internet where an adversary uses the ICMP echo request packets to IP broadcast addresses from remote locations to deny services. This attack causes temporary denial of services and can be automatically recovered. Looking for a large number of echo replies to the innocent machine from
different places without any echo request made by the innocent machine helps in detecting this attack.

**Teardrop:** DoS attack performed against older versions of TCP/IP stack where an adversary exploits the feature of IP fragment reassembly. This attack denies the services temporally.

### 4.8.2 Remote to User

In this type of attack an attacker who does not have an account on a remote machine sends packets to that machine over a network and exploits some vulnerability to gain local access as a user of that machine. Examples are Dictionary, Ftp_write, Guest, Imap, Named, Phf, Sendmail, Xlock.

**Phf:** The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server.

**Ftp_write:** The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system.
**Guest:** The Guest attack is a variant of the Dictionary attack. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit the system.

**Imap:** The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges.

**Xlock:** In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing 'xhost +'), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.

**4.8.3 User to Root**

In this type of attacks an attacker starts out with access to a normal user account on the system and is able to exploit system vulnerabilities to gain
root access to the system. Examples are Eject, Loadmodule, Ps, Xterm,Perl, Fdformat.

Ps: The Ps attack takes advantage of a race condition in the version of 'ps' distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition.

Perl: The Perl attack is a User to Root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access.

Loadmodule: The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine.

Xterm: The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 and allows an attacker to execute
arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system.

**Fdformat:** The Fdformat attack exploits a buffer overflow in the 'fdformat' program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program.

**Eject:** The Eject attack exploits a buffer overflow in the 'eject' binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems.

4.8.4 Probing

In this type of attacks an attacker scans a network of computers to gather information or find known vulnerabilities. An attacker with a map of
machines and services that are available on a network can use this information
to look for exploits. Examples are Ipsweep, Mscan, Saint, Satan, Imap.

**Ipsweep:** Probing attack is performed against all operating systems
using ICMP service where an adversary performs a surveillance sweep to
determine which hosts are responding on a network. Information obtained
from surveillance is useful to an adversary in launching automated attacks or
in making the vulnerable hosts as stepping stones for future distributed
attacks. This attack helps the adversary identify active machines on the
network and might degrade services for legitimate users. Looking for multiple
ping requests, destined for all possible machines on a network, all coming
form the same host can help detect this attack.

**Mscan:** Probing tool used to perform an attack against all operating
systems using multiple services; where an adversary uses both DNS zone
transfers and/or brute force scanning of IP addresses (Uyliss Black 2002) to
locate machines, and look for vulnerabilities to launch future attacks. This
attack helps the adversary identify known vulnerabilities on the network and
the host machine. Looking for connection requests from an outside machine
to vulnerable services (netbios-ns, epmap, ms-sql-m, dameware, microsoft-ds,
realsecure, domain, bind, imap, pop, NFS, cgi-bin, open X servers) within a
specified period of time, can help detect this attack.

**Nmap:** General-purpose probing tool used to perform network
scans against all operating systems using multiple services with user specified
time intervals; an adversary can specify which services to scan for, how much
time to wait between each service, and whether the services should be
scanned sequentially or in a random order. This attack helps the adversary
identify services running, operating system, and known vulnerabilities on the
network and the target machine. Looking for connection requests to multiple
services within a specific time window, can help detect this attack.
**Saint:** Security Administrator’s Integrated Network Tool is used to gather information about remote hosts (all operating systems) using multiple services; an adversary uses a few network services such as finger, ftp, tftp, statd, rpc, NIS, NFS (Comer 1999) and other relevant network services. This attack helps the adversary identify network services running, system flaws, and critical security flaws on the victims’ machine. Looking for connections requests to specific network services from a machine other than an authorized machine within a specific time window, can help detect this attack.

**Satan:** Probing tool used to perform scans against all operating systems using a few network services; where an adversary uses legitimate network services to gather information on particular vulnerabilities on the victims’ machine. Looking for connection requests to specific vulnerable network services from a machine other than an authorized machine within a specific time window can help detect this attack.

### 4.9 Attributes in Data Set

The data set has 41 attributes for each connection record and one class label. R2L and U2R attacks don’t have any sequential patterns like DOS and Probe because the former attacks have the attacks embedded in the data packets whereas the later attacks have many connections in a short amount of time. Therefore, some features that look for suspicious behavior in the data packets like numbers of failed logins are constructed and these are called content features.
Tables 4.1, 4.2 and 4.3 give the details about 41 attributes.

Table 4.1 Basic features of individual TCP/IP connections

<table>
<thead>
<tr>
<th>FEATURE NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>duration</td>
<td>length (number of seconds) of the connection</td>
<td>continuous</td>
</tr>
<tr>
<td>protocol_type</td>
<td>type of the protocol, e.g. tcp, udp, etc.</td>
<td>discrete</td>
</tr>
<tr>
<td>service</td>
<td>network service on the destination, e.g., http, telnet, etc.</td>
<td>discrete</td>
</tr>
<tr>
<td>src_bytes</td>
<td>number of data bytes from source to destination</td>
<td>continuous</td>
</tr>
<tr>
<td>dst_bytes</td>
<td>number of data bytes from destination to source</td>
<td>continuous</td>
</tr>
<tr>
<td>flag</td>
<td>normal or error status of the connection</td>
<td>discrete</td>
</tr>
<tr>
<td>land</td>
<td>1 if connection is from/to the same host/port; 0 otherwise</td>
<td>discrete</td>
</tr>
<tr>
<td>wrong_fragment</td>
<td>number of “wrong” fragments</td>
<td>continuous</td>
</tr>
<tr>
<td>urgent</td>
<td>number of urgent packets</td>
<td>continuous</td>
</tr>
</tbody>
</table>
Table 4.2 Content features within a connection suggested by domain knowledge

<table>
<thead>
<tr>
<th>FEATURE NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>number of connections to the same host as the current connection in the past two seconds</td>
<td>continuous</td>
</tr>
<tr>
<td>serror_rate</td>
<td>% of connections that have “SYN” errors</td>
<td>continuous</td>
</tr>
<tr>
<td>rerror_rate</td>
<td>% of connections that have “REJ” errors</td>
<td>continuous</td>
</tr>
<tr>
<td>same_srv_rate</td>
<td>% of connections to the same service</td>
<td>continuous</td>
</tr>
<tr>
<td>diff_srv_rate</td>
<td>% of connections to different services</td>
<td>continuous</td>
</tr>
<tr>
<td>srv_count</td>
<td>number of connections to the same service as the current connection in the past two seconds</td>
<td>continuous</td>
</tr>
<tr>
<td>srv_serror_rate</td>
<td>% of connections that have “SYN” errors</td>
<td>continuous</td>
</tr>
<tr>
<td>srv_rerror_rate</td>
<td>% of connections that have “REJ” errors</td>
<td>continuous</td>
</tr>
<tr>
<td>srv_diff_host_rate</td>
<td>% of connections to different hosts</td>
<td>continuous</td>
</tr>
</tbody>
</table>

Note: The following features refer to these same-host connections.
Table 4.3 Traffic features computed using two-second time window

<table>
<thead>
<tr>
<th>FEATURE NAME</th>
<th>DESCRIPTION</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>number of “hot” indicators</td>
<td>continuous</td>
</tr>
<tr>
<td>num_failed_logins</td>
<td>number of failed login attempts</td>
<td>continuous</td>
</tr>
<tr>
<td>logged_in</td>
<td>1 if successfully logged in; 0 otherwise</td>
<td>discrete</td>
</tr>
<tr>
<td>num_compromised</td>
<td>number of “compromised” conditions</td>
<td>continuous</td>
</tr>
<tr>
<td>root_shell</td>
<td>1 if root shell is obtained; 0 otherwise</td>
<td>discrete</td>
</tr>
<tr>
<td>su_attempted</td>
<td>1 if “su root” command attempted; 0 otherwise</td>
<td>discrete</td>
</tr>
<tr>
<td>num_root</td>
<td>number of “root” accesses</td>
<td>continuous</td>
</tr>
<tr>
<td>num_file_creations</td>
<td>number of file creation operations</td>
<td>continuous</td>
</tr>
<tr>
<td>num_shells</td>
<td>number of shell prompts</td>
<td>continuous</td>
</tr>
<tr>
<td>num_access_files</td>
<td>number of operations on access control files</td>
<td>continuous</td>
</tr>
<tr>
<td>num_outbound_cmds</td>
<td>number of outbound commands in an ftp session</td>
<td>continuous</td>
</tr>
<tr>
<td>is_hot_login</td>
<td>1 if the login belongs to the “hot” list; 0</td>
<td>discrete</td>
</tr>
<tr>
<td>is_guest_login</td>
<td>1 if the login is a “guest” login; 0 otherwise</td>
<td>discrete</td>
</tr>
</tbody>
</table>

4.10 EXPERIMENT AND RESULT

Available data set is divided into training set and testing set. Test is repeated by selecting different percentage between these two to study the prediction and time performance.

Initially, model learning is achieved by randomly selecting 75 percent of 370514 records from the total available labelled data set of 494021 records. Learning process took 42491 seconds for training. Learned model is implemented as a sequential process and parallel process in symmetric
multiprocessing (SMP) UNIX running on Intel core 2 due at 2 GHz with 2GB main memory and 250 GB hard disc drive. It’s prediction performance and time performance is studied. For testing, test data set is separated into 5 sets each containing one class of records of different percentage to gauge the performance of the classifier for various size of input files. These create one input file for each of DOS, PROBE, U2R, R2L and NORMAL type classes.

Test was conducted by taking 100%, 75%, 50% and 25% of test data and studied the accuracy and time taken for classification for normal method and parallel method.

Initially, learned classifier is tested with 25% of test data, which is 6.25 percent of actual data. Classification accuracy obtained is 100, 99.61, 100, 100 and 99.37 percentages for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 4.4.

Number of records tested in PROBE is 257, U2R is 3, R2L is 70, DOS is 24466 and Normal traffic is 6080. Misclassification rate is 0.126 percentage. Gain in performance through parallel implementation compared with sequential processing is 1.85 percentages.

Table 4.4 Results with 25% of bench mark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>Total test cases</td>
<td>257</td>
<td>3</td>
<td>70</td>
<td>6080</td>
<td>24466</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>99.61</td>
<td>100</td>
<td>100</td>
<td>99.37</td>
<td>100</td>
</tr>
<tr>
<td>Time for Parallel Processing (ms)</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>140</td>
<td>502</td>
</tr>
<tr>
<td>Time for Sequential Processing (ms)</td>
<td>12</td>
<td>4</td>
<td>6</td>
<td>247</td>
<td>954</td>
</tr>
</tbody>
</table>
Classification accuracy for 50% of test data, which is 12.5 percent of actual data, is 100, 99.80, 57.14, 100 and 99.63 percentages for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 4.5. Total number of records in each of DOS, PROBE, U2R, R2L and NORMAL are 48933, 514, 7, 141 and 12160. Number of misclassified records in each of these categories in the same order is 0, 1, 3, 0 and 44. Average performance gain obtained for parallel implementation is 1.86 times compared to sequential single processor implementation.

Table 4.5 Results with 50% of benchmark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>Total test cases</td>
<td>514</td>
<td>7</td>
<td>141</td>
<td>12160</td>
<td>48933</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>99.80</td>
<td>57.14</td>
<td>100</td>
<td>99.63</td>
<td>100</td>
</tr>
<tr>
<td>Time for Parallel Processing (ms)</td>
<td>14</td>
<td>4</td>
<td>6</td>
<td>266</td>
<td>1012</td>
</tr>
<tr>
<td>Time for Sequential Processing (ms)</td>
<td>21</td>
<td>4</td>
<td>8</td>
<td>495</td>
<td>1897</td>
</tr>
</tbody>
</table>

When 75% of remaining data, that is 18.75 percent of actual data were tested, obtained accuracy is 99.98 99.74, 40, 100 and 99.70 percentage for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 4.6. Total number of records in each of DOS, PROBE, U2R, R2L and NORMAL are 73399, 771, 10, 212 and 18240. Number of misclassified records in each of these categories in the same order is 11, 2, 10, 212 and 18240. Average performance gain obtained for parallel implementation is 1.89 times compared to sequential single processor implementation.
Table 4.6 Results with 75% of Bench Mark Test Data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>54</td>
<td>11</td>
</tr>
<tr>
<td>Total test cases</td>
<td>771</td>
<td>10</td>
<td>212</td>
<td>18240</td>
<td>73399</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>99.74</td>
<td>40</td>
<td>100</td>
<td>99.70</td>
<td>99.98</td>
</tr>
<tr>
<td>Time for Parallel Processing (ms)</td>
<td>19</td>
<td>4</td>
<td>7</td>
<td>386</td>
<td>1516</td>
</tr>
<tr>
<td>Time for Sequential Processing (ms)</td>
<td>28</td>
<td>4</td>
<td>10</td>
<td>722</td>
<td>2896</td>
</tr>
</tbody>
</table>

When 100% of remaining data, that is 25 percent of actual data were tested, obtained accuracy is 99.77, 99.80, 28.57, 99.64 and 99.75% for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 4.7. Total number of records in each of DOS, PROBE, U2R, R2L and NORMAL are 97866, 1028, 14, 283 and 24321. Number of misclassified records in each of these categories in the same order is 221, 1028, 14, 283 and 24321. Average performance gain obtained for parallel implementation is 1.9 times compared to sequential single processor implementation. When number of data records is very meager the overhead of initial time required to set up a process makes both the implementation to take the same time.

Table 4.7 Results with 100% of bench mark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>60</td>
<td>221</td>
</tr>
<tr>
<td>Total test cases</td>
<td>1028</td>
<td>14</td>
<td>283</td>
<td>24321</td>
<td>97866</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>99.80</td>
<td>28.57</td>
<td>99.64</td>
<td>99.75</td>
<td>99.77</td>
</tr>
<tr>
<td>Time for Parallel Processing (ms)</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>491</td>
<td>1989</td>
</tr>
<tr>
<td>Time for Sequential Processing (ms)</td>
<td>43</td>
<td>4</td>
<td>12</td>
<td>933</td>
<td>3817</td>
</tr>
</tbody>
</table>
Performance gain seen for detecting PROBE attack is depicted in Figure 4.3. It shows sustained performance gain when data records got increased. The gain in improvement is linear and it can clearly be seen in the following Figure 4.3 for the PROBE attack.

![Figure 4.3 Performance for detection of probe attack](image)

As the total number of records available in this U2R attack is very meager as shown in Figure 4.4, performance measurement obtained is not appreciable as expected. Since, initial over head of setup time for the execution of the program itself contributes major portion of the metered time.
R2L attack has total of 283 records in the test set. These records are randomly divided and time taken for the classification process for sequential and parallel implementation of classifier is depicted in the following Figure 4.5. Since, number of records available is minimum, performance gain is less.
Number of normal traffic connection records available for testing is 24321. As evident from the Figure 4.6, perforomance gain achieved is appreciable. As the number of records increases gain also proporitnately increase. In any real world traffic, ideally it is natural to expect significant portion of traffic to be normal legitimate traffic. So, performance gain achieved for the detection is encouraging. As seen in the Figure, the performance improvement realised for 6080 records for parallel implementation of the classifier is 1.76 times of 140 ms compared to the sequential process execution time of 247 ms. But, when number of records increases to 24321 records realised performance gain shoots up to 1.9 times by completing the classification with in 491 ms compared to the sequential single processor execution time of 933 ms.

![NORMAL TRAFFIC](image)

**Figure 4.6 Performance for detection of Normal traffic**

By the nature of the DOS attack, it is reasonable and normal to expect continuous stream of connection records for this attack so as to overwhelm the system or network connection to become unavailable to the legitimate user of the resource. This one is the easy type of traffic to produce
and does not require much indepth knowledge about the technology or topology of the network or the applications on the victim machine or the operating systems running or the open ports on the machine or versions of the various software running. So, in effect any novice can cause nuisance by launching this easy attack without any initial study or knowledge of the victim machine. It is very important to be able to tackle this type of attack as quickly as possible. For that to happen first step is to detect with minimal time. As clearly seen from the Figure 4.7 the performance gain for the parallel approach results in speed up of approximately 1.9 times compared to the sequential single process execution.

![DOS Attack Graph](image)

**Figure 4.7 Performance for detection of DOS attack**

### 4.11 SUMMARY

In this chapter, the challenges associated with supporting real-time traffic are discussed and real-time traffic requirement are listed. The general intrusion detection model using data mining approach is elaborated. The functional requirements of different modules of intrusion detection models are
briefly explained. Various attributes of the bench mark data set and the class labels used in the data set are discussed. Data mining technique DT is explained as to how it learns the proposed prediction model for intrusion detection automatically. Procedure used for the construction of the model and testing of the model is explained. Performance gain achieved for the sequential execution and parallel implementation are studied. The performance shows that the proposed method gives better performance.