TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Ananda Niketan, Pulchowk, Lalitpur, P.O. Box 1175, Kathmandu, Nepal.
Tel : 5534070, 5521260 extn. 315, Fax : 977-1-5553946, E-mail : doece@ioe.edu.np

Our Ref :

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Institute of Engineering for acceptance, a project report entitled "Host Based Intrusion Detection System" submitted by Ashok Adhikari, Deepak Gautam and Mohan Khanal in partial fulfillment of the requirements for the Bachelor's degree in Computer Engineering.

Supervisor, Dr. Aman Shakya
Lecturer
Department of Electronics and Computer Engineering, Pulchowk Campus

Supervisor, Mr. Manoj Ghimire
Lecturer
Department of Electronics and Computer Engineering, Pulchowk Campus

Internal Examiner
Prof. Timila Yami Thapa
Assistant Dean
Institute of Engineering

External Examiner,
DIGP Mahesh Singh Kathayat
ME, MBA(Executive)
Head of Nepal Police Computer Directorate
Nepal Police, HQs, Naxal, Kathmandu
Nepal

Project Coordinator
Surendra Shrestha, PhD
Deputy Head
Department of Electronics and Computer Engineering, Pulchowk Campus

Date of Approval: November, 2011

COPYRIGHT

ACKNOWLEDGEMENT

We would like to express our gratitude to the Department of Electronics and Computer Engineering, Pulchowk Campus for providing us with the opportunity to work on the Major Project as a part of our syllabus. We are deeply indebted to our supervisors Dr. Aman Shakya and Mr. Manoj Ghimire whose help, stimulating suggestions and encouragement helped us in each and every step of our project.

We would like to give special thanks to all the teachers and friends whose suggestions and ideas proved to be vital in our project completion.

We would also like to acknowledge MIT Lincon Library Department for providing us with the datasets needed for validating various models used in our intrusion detection system.

Ashok Adhikari

Deepak Gautam

Mohan Khanal

November 2011

ABSTRACT

Because of rapid increase in computer attacks, intrusion detection has become the cornerstone in the field of computer security. Anomaly based variant is one of the most effective and reliable form of intrusion detection systems.

In our project, we developed an Anomaly based Intrusion Detection System. We modelled our system to capture the frequency and transitional behaviour of the data. In order to capture the frequency property, we used SOM and in order to capture the transitional behaviour, we used Markov Model. The system developed was both trained and tested using the datasets provided by DARPA. DARPA has collected and distributed the first standard corpora for evaluation of computer intrusion detection system.

We believe the research carried out during the project will be of great value for the people interested in developing anomaly based intrusion detection systems.

Keywords

HIDS, Anomaly based IDS, SOM, Markov Model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AFRL/SNHS	Air Force Research Laboratory sponsorship
ANN	Artificial Neural Network
BSM	Basic Security Module
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
FFB	Fast Frame Buffer
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
HIDS	Host Based Intrusion Detection System
HMM	Hidden Markov's Model
IDS	Intrusion Detection System
IP	Internet Protocol
KNN	k Nearest Neighbour
MIT	Massachusetts Institute of Technology
NIDES	Next-generation real-time Intrusion Detection Expert System
NIDS	Network based Intrusion Detection System
OSSEC	Open Source Security
ROC	Receiver Operating Characteristics
SOM	Self Organizing Map

SVM	Support Vector Machine
TCP	Transmission Control Protocol
TP	True Positive
TPR	True Positive Rate

1 INTRODUCTION

1.1 Motivation

Information has become an organization’s most precious asset. Organizations have become increasingly dependent on the information since more information is being stored and processed on network-based systems. The wide spread use of e-commerce, has increased the necessity of protecting the system to a very high extend. The nature of threat has changed from physical infiltration and password breaking to computer viruses, self propagating and self replicating worms, backdoor software, trojan horses, script kiddies, computer criminals and terrorists.

Despite the increase in defence provided to the security system, however, the threat has neither being eliminated nor mitigated. In fact, computer crimes have become more organised and sophisticated as the intents and motives of these crimes are changing from mere fun and bragging to high financial gains, information gathering for information warfare and terrorism.

Because the last few years have seen a dramatic increase in the number of attacks, intrusion detection has become the mainstream of information assurance. While firewalls do provide some protection, they do not provide full protection and still need to be complimented by an intrusion detection system. The purpose of intrusion detection is to help computer systems prepare for and deal with attacks. Intrusion detection systems collect information from a variety of sources within computer systems and networks. For most systems, this information is then compared to predefined patterns of misuse to recognize attacks and vulnerabilities. However, there are new techniques of intrusion detection including the use of support vector machines and neural networks. These techniques, along with behavioural data forensics, create a database of normal user behaviour and will alert the security officer if a deviation from that normal behaviour occurs.

The number of security tools and their complexity is rapidly growing. They produce a growing number of logs that are collected and stored. It is impossible to analyze all the data manually. Therefore, automatic methods are needed to scan the data sets and detect the most interesting or suspicious parts of the data. These potentially interesting data are then to be examined by human expert. Detection of anomalies or outliers is important in log data analysis. Locating rare or suspicious parts of the data can reveal new valuable information from the system.

Thus reacting to a changing threat environment, is a key requirement for modern intrusion detection systems. It is obvious from the exponential growth of the enabling technologies and the increasing demand in the autonomy and mobility of these systems that traditional techniques of dealing with the protection of these systems will not suffice and we need alternative approaches that can adapt to these drifting concepts over time.

1.2 Problem Description

Systems found in Nature, often referred to as complex adaptive systems, are highly robust and resilient that can adapt to environmental changes and constantly evolve their states for their betterment. It generally refers to the ability of an organism to survive in a new environment by accommodating changes in the environment. Hence, according to the behaviors of the intrusion attacked upon a system can provide a good way for system to change its states towards betterment.

There are various approaches in the real world for detecting intrusions; these are signature based and anomaly based. We implement anomaly based approach because signature based approach rely only on the rules defined. The signatures are mainly created manually by domain experts and are usually updated only after new intrusions had compromised the security of protected systems. The problem with a signature based detection approach is that it is reactive by nature, so once a new form of intrusion is developed, its signature must be updated to the system, so signature based system can’t capture new form of intrusion. Anomaly based intrusion detection system closely monitors behaviors of the normal activities and intruder’s activities. When a program is misused, its behavior will differ from its normal usage. Therefore, if the normal range of program behavior can be adequately and compactly represented, then behavioral features captured by audit mechanisms can be used for intrusion detection.

To detect novel attacks against systems, we develop anomaly-based systems that report any unusual use of system programs as potential intrusions. The main advantage of anomaly based approach is that it can recognize both known attacks and novel attacks. We develop an anomaly based detection system that uses learning of the normal behavior of the programs. The trained model then used to detect possibly intrusive behavior by identifying significant behaviors of the intrusive programs. The goal of these approaches is to be able to detect not only known attacks and but also detect future novel attacks using off-the-shelf auditing mechanisms provide by the operating system vendor.

1.3 Objectives

This project aims to design and implement a system that can learn normalities of the presented dataset and dynamically identify abnormal entries. In broad objective of our project can be defined as follows:

- **To study the ways intrusions can be detected:** Intrusive behaviors can be of various forms and exhibit various phenomena. So one of our major objectives is to study various grounds on which we can develop our intrusion detection system.
- **To develop an Anomaly Based Intrusion Detection System:** We finally aim to develop a IDS which can detect intrusions by comparing their behavior against the already modeled normal behavior.
- **To study the nature of intrusion contained on the datasets we use to generate the detection model:** We aim on studying and understanding the intrusions present in the dataset that we use in developing our detections system and then verify why each intrusion is called an intrusion by our system, if it does, according to the behavior it exhibits.
- **To validate our system using labeled datasets:** We also aim to validate our system in terms of its efficiency and detection rates using already labeled datasets in terms of *intrusions* or *non-intrusions.*

1.4 Scope

Our system detects intrusion on the data by viewing transition and frequency property of the input data. Thus the anomaly on the frequency and the transitional property of any kind of data provided can be detected by our system.

MIT Lincoln Library provided training and testing data sets for the testing of intrusion detection systems. We take these data sets input to our system and tested the data sets. If our system works in this dataset with quite good results than we can be certain that our system can model any other data set that exhibits frequency and transitional property. Thus our system is a generalized system for intrusion detection using frequency and transitional property of data.

1.5 Overview

The following section contains a brief overview of the report. A brief summary of what has been presented till now and what will be discussed later is mentioned here.

Chapter 2 contains the background theory of the project. It contains the description of the basic concepts that are involved in understanding what intrusion detection systems are. It also contains the techniques and approaches to anomaly detection.

Chapter 3 contains the literature reviews that we did and also contains a brief review of the existing systems.

Chapter 4 contains the methodology that we followed in our project. It contains the description of system flow as well as the examples of datarows we used in training and testing of our system.

Chapter 5 contains the details of the implementation tasks. It contains the techniques we used in our project i.e. SOM and Markov Model. It also contains the algorithms and details of the techniques.

Chapter 6 contains the results and analysis that we obtained. This section contains the results that we obtained after exposing our system to the datasets that we used in verifying the system i.e.1998 DARPA Datasets. Various ROC curves have been presented and the analysis also has been presented.

Finally Chapter 7 contains the conclusions that we drew form the project.

2 BACKGROUND THEORY

2.1 Intrusion

Intrusion is the act of wrongfully entering upon, seizing or taking possession of the property of another [1]. In computer systems, intrusions may be of various forms just like spam, viruses, unauthorised access, and so on.

2.2 Intrusion detection System (IDS)

Intrusion detection is defined as the process of intelligently monitoring the events occurring in a computer system or network, analyzing them for signs of violations of the security policy [2]. The primary aim of IDS is to protect the availability, confidentiality and integrity of critical networked information systems.

IDS are defined by both the method used to detect attacks and the placement of the IDS on the network. IDS may perform either misuse detection or anomaly detection and may be deployed as either a network-based system or a host-based system. Misuse detection relies on matching known patterns of hostile activity against databases of past attacks. They are highly effective at identifying known attack and vulnerabilities, but rather poor in identifying new security threats. Anomaly detection will search for something rare or unusual by applying statistical measures or artificial intelligence to compare current activity against historical knowledge.

2.3 Host Based Intrusion Detection System (HIDS)

A host-based IDS monitors the characteristics of a single host and the events occurring within that host for suspicious activity [3]. Examples of the types of characteristics host-based IDS might monitor are network traffic (only for that host), system logs, running processes, file access and modification, system calls, registry access, etc.

2.4 Anomaly Detection

Anomaly-based detection is the process of comparing definitions of what activity is considered normal against observed events to identify significant deviations. An IDS using anomaly-based detection has profiles that represent the normal behaviour of the system. The profiles are developed by monitoring the characteristics of typical activity over a period of time. The IDS then uses statistical methods to compare the characteristics of current activity to thresholds related to the profile for determining anomalous activity. The anomalous activity if detected is alerted to the administrator.

An initial profile is generated over a period of time called a training period. The profile is established by supervised or unsupervised learning using various data mining approaches. The profile of the system is compared with the current activity to determine the anomalous activity on the system. The profile of the system is regenerated periodically because systems change over time.

There exist a variety of methods of anomaly detection that have been shown to perform well on different data sets. Below is a brief description of those methods:

2.4.1 Probabilistic approaches

This category of approaches is based on statistical modelling of data and then estimating whether the test data come from the same distribution that generates the training data. First estimate the density function of the training data. By assuming the training data is normal, the probability that the test data belong to that class can be computed. A threshold can then be set to signal the anomaly if the probability calculated is lower than that threshold. Examples: Hidden Markov's Model (HMM).

2.4.2 Non-parametric approaches

For non-parametric methods, the overall form of the density function is estimated from the data as well as parameters of the model. The k-nearest neighbour algorithm is non-parametric approach for anomaly detection. For anomaly detection the distribution of normal vectors is described by a small number of spherical clusters placed by the k-nearest neighbour technique. Anomaly is assessed by measuring the normalised distance of a test sample from the cluster centres.

2.4.3 Neural network based approaches

Quite a number of different architectures of neural networks are applied to anomaly detection. A neural network can detect anomaly by setting a threshold on the output values of the network. Or it can calculate the Euclidean distance between output patterns and target patterns and throw those with highest distance out as the anomaly. Examples: Self Organizing Map (SOM), Artificial Neural Network (ANN).

2.5 Types of Anomaly

2.5.1 Point Anomalies

If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed a point anomaly [4].

2.5.2 Contextual Anomalies

If a data instance is anomalous in a specific context, but not otherwise, then it is termed a contextual anomaly (also referred to as conditional anomaly) [4].

2.5.3 Collective Anomalies

If a collection of related data instances is anomalous with respect to the entire data set, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous [4].

2.6 Anomaly Detection Techniques

2.6.1 SOM

Self Organising Maps was proposed by Kohonen as a projection method and a clustering algorithm. It is basically a dimensionality reduction method where high dimensional data are mapped onto lower dimension data. So using this technique we can visualize or analyse higher dimensional data by mapping them to lower dimensional data space.

SOM is a category of neural network which preserves the topology of the original data. The learning phase of SOM is trained to model the input space and has the following two characteristics:

- Competition: the learning is enforced by competition among the neurons: when an input x arrives, the neuron that is best able to represent it wins the competition and is allowed to learn it better.
- Cooperation: not only the winning neuron but also its neighbours on the lattice are allowed to learn. Neighbouring neurons will gradually specialize to represent similar inputs, and the representations will become ordered on the map lattice. The cooperative learning is based on neighbourhood functions.

In intrusion detection SOM can be used to capture the frequency property of the data. Firstly the network is trained with normal data set. The map then clusters the data presented according to their similarity. This means we now have a map with normal clustered data along with their similarity preserved. During the testing phase we can simply feed the test data to the network and see if the fed input is close enough to any node in the map. If close enough than some predefined specified value, the data presented is not an anomaly as it fits in the map, else it is an intrusion.

2.6.2 Hidden Markov Model

Hidden Markov Model is a probabilistic approach for anomaly detection. It captures the transitional property the data possesses. It basically detects collective anomaly present in the dataset.

An HMM describes a doubly stochastic process. Each HMM contains a finite number of unobservable (or hidden) states. Transitions among the states are governed by a set of probabilities called transition probabilities. An HMM defines two concurrent stochastic processes: the sequence of the HMM states and a set of state output processes. There are three central issues in HMMs including the evaluation problem, the decoding problem, and the learning problem. Given an input sequence of system calls, an HMM can model this sequence by three parameters:

- state transition probability distribution A,
- observation symbol probability distribution B and,
- initial state distribution PI.

Training of data is done using Bourn-Welch algorithm. Training is basically setting the values of matrix A, B and PI which corresponds to the normal behaviour of the data set.

For any given sequence of data presented to the model, we can obtain the probability of the model generating the sequence. By comparing this probability value to the predefined standard value of probability we can find whether that sequence is normal or abnormal.

2.6.3 SVM

SVM is an useful supervised learning technique for classification. Given a training set, each signal belongs to one of two categories, SVMs uses the training set to build a model, that can help to predict the class of a new testing signal. SVMs build hyperplanes to separate between points belong to different categories. The hyperplanes can be in the original space of input signals or in a higher dimensional feature space after applying a mapping function to the input signals. A good hyperplane is selected based on the gap or the distance between the hyperplane and the nearest training datapoints of any class. The larger the distance, the better the hyperplane [5].

2.6.4 ANN

ANN is a biologically inspired form of distributed computation. It is composed of simple processing units, or nodes, and connections between them. The connection between any two units has some weight, which is used to determine how much one unit will affect the other. A subset of the units acts as input nodes and another subset acts as output nodes, which perform summation and thresholding. An early stopping strategy is usually used to overcome the over-6tting problem. The early stopping method tracks the network performance using a separate validation set. Typically, the errors in the validation set will decrease as the network fits the data, and then increase as the network fits idiosyncrasies in the training data. The ANN has successfully been applied in different settings, including network reliability, sports winning prediction, medical, marketing, retail, banking and 6nance, insurance, telecommunications, operations management and other industries. In this study, we will employ a classic feed-forward neural network trained with the back-propagation algorithm to predict intrusions [5].

2.6.5 KNN

k-Nearest Neighbor is one of the machine learning algorithms. It is a method for classifying a signal based on closest training samples in the feature space. kNN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification1. A signal is classified by a majority vote of its neighbors. A signal is assigned to the class most common amongst its k nearest neighbors [6].

2.7 Data Sets

2.7.1 Introduction

Many parties are working on the development of intrusion detection systems, including universities, commercial software companies, and organizations within the Department of Defense. As these groups explore different methods and develop various new systems for intrusion detection, it is clearly advantageous to have a means of evaluating the success of these systems in detecting attacks. The best environment for testing and evaluation of an intrusion detection system is the actual environment in which it will be used. However, research groups often do not have access to operational networks on which to test their systems, and these systems are tested in a simulated environment. The ability to perform accurate testing and evaluation in a simulated environment requires high-quality data that is similar to the traffic (including attacks) that one finds on operational networks. In general, this data is difficult to acquire because it contains private information and reveals potential vulnerabilities of the networks from which the data is collected. These factors led to DARPA sponsorship of MIT Lincoln Laboratory's 1998 intrusion detection evaluation, which created the first standard corpus for the evaluation of intrusion detection systems.

The Cyber Systems and Technology Group of MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency (DARPA ITO) and Air Force Research Laboratory (AFRL/SNHS) sponsorship, has collected and distributed the first standard corpora for evaluation of computer network intrusion detection systems. The first formal, repeatable, and statistically significant evaluations of intrusion detection systems have been carried out in 1998 and 1999 [7].

These evaluations measured probability of detection and probability of false alarm for each system under test. These evaluations contributed significantly to the intrusion detection research field by providing direction for research efforts and an objective calibration of the technical state of the art. The evaluation was designed to be simple, to focus on core technology issues, and to encourage the widest possible participation by eliminating security and privacy concerns, and by providing data types that were used commonly by the majority of intrusion detection systems [7].

The DARPA 1998 Intrusion Detection Evaluation data set consists of about 5 million connections of labeled training data and 2 million connections of test data. The goal of the simulation network (or simnet) was to accurately simulate the network traffic of a Local Area Network that one might find at a United States Air Force facility. Automatically generated traffic used more than 20 network services, including dns, finger, ftp, http, ident,

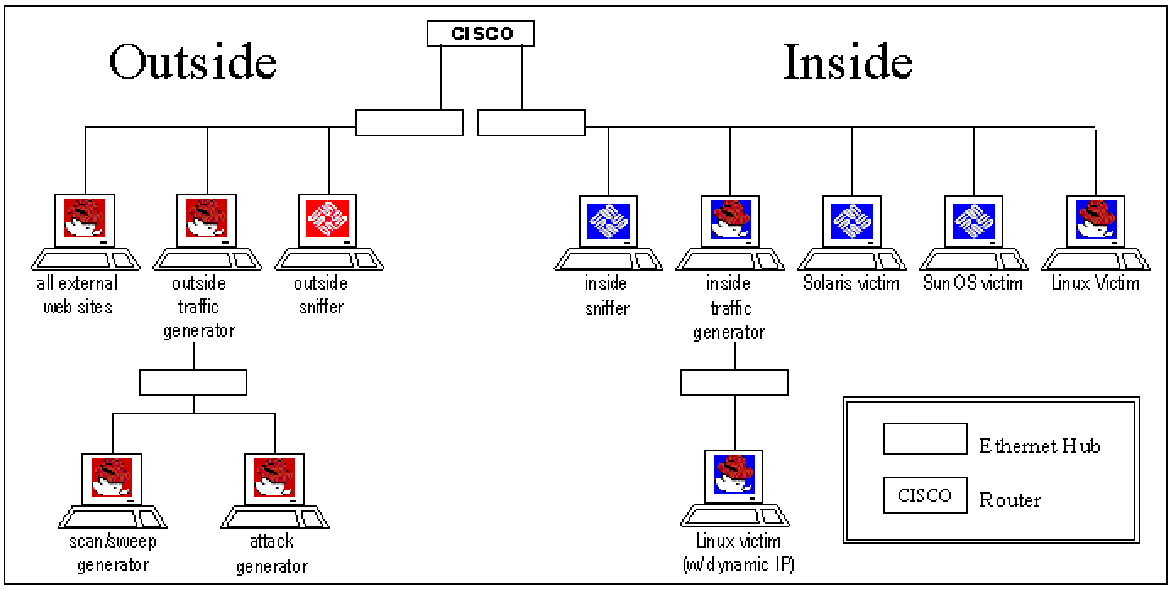


Figure 2.1 Simulation Network Topology

ping, pop, smtp, snmp, telnet, time, and X. In order to accurately model the features of an Air Force network, statistics were measured from months of actual network traffic that was collected from more than fifty Air Force computer networks. Traffic statistics for automatically generated traffic matched average Air Force statistics. The content of email, ftp, web sites, and files was similar to actual documents. Content was generated using open-source documentation or statistical reconstruction from a large set of unclassified documents that preserved both unigram word frequencies and also the frequency of two-word sequences. Generating the data from public sources allowed the data to be distributed without security or privacy concerns.

The 1998 DARPA Intrusion Detection System Evaluation program provides a large sample of computer attacks embedded in normal background traffic. The *TCPDUMP* and *BSM* audit data were collected on a network that simulated the network traffic of an Air Force Local Area Network. The audit logs contain seven weeks of training data and two weeks of testing data. There were 38 types of network based attacks and several realistic intrusion scenarios conducted in the midst of normal background data.

2.7.2 Types of Attacks

2.7.2.1 DoS attack

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of DoS attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon [8].

One of the Dos attack, *Process Table,* which we te4sted in our system is described below.

2.7.2.1.1 Process Table

The Process Table attack is a denial-of-service attack. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the super-user can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine’s process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine. An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program’s source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed: 1. Open a connection to the target's finger port. 2. Wait for 4 seconds. 3. Repeat steps 1-2. This attack has been attempted against a variety of network services on a variety of operating systems [8].

2.7.2.2 User to Root attack

User to Root exploits are class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions that was to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Despite the fact that programmers can eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks . Another class of User to Root attack exploits programs that make assumptions about the environment in which they are running. Other User to Root attacks take advantage of programs that are not careful about the way they manage temporary files [8].

Finally, some User to Root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously . Although careful programming could eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today. Note from this table that all of the User to Root attacks can be run from any interactive user session (such as by sitting at the console, or interacting through telnet or rlogin), and that all of the attacks spawn a new shell with root privileges. The following sections describe each of the User to Root attacks that was used in the 1998 DARPA intrusion detection evaluation in greater detail.

Some of user-to-root attacks which we tested in our system are described below.

2.7.2.2.1 Eject

The Eject attack exploits a buffer overflow is 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 [8].

2.7.2.2.2 Format

The format attack exploits a buffer overflow is the "format" program distributed with Solaris 2.5. The format 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 [8].

2.7.2.2.3 FfbConfig

The Ffbconfig attack exploits a buffer overflow is the ffbconfig program distributed with Solaris 2.5. The ffbconfig program configures the Creator FFB Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWffbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the ffbconfig program [8].

2.7.2.2.4 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 [8].

2.7.2.3 Remote to User attacks

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network—but who does not have an account on that machine—exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks exploit buffer overflows in network server software (imap, named, sendmail).

The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering—in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse.

The following sections provide details of each of these attacks.

2.7.2.3.1 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 [8].

2.7.2.3.2 Xlock

In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their 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 display open, 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 [8].

3 LITERATURE REVIEW

Various works have been done and are being done in the field of Intrusion Detection Systems. Systems developed use anomaly based detection schema or rule based detection schema as mentioned earlier. Systems like OSSEC, TRIP WARE use misuse detection (i.e. rule based approach) for detecting intrusion. In anomaly detection schema, profiling the normal behaviour of the system is the core part. Once normal behaviour is modelled, the system can then detect anomalies using its normal behaviour's profile. Systems like NIDES/STAT use anomaly based detection schema where statistical approaches are used for profiling the normal behaviour of the system.

3.1 Existing System

Following is the list of currently existing popular intrusion detection systems:

3.1.1 Rule Based HIDS

3.1.1.1 OSSEC

It is a *rule based intrusion* detection system. It basically performs log file analysis, integrity checking, windows registry monitoring, root kit detection, time-based alerting and active response.

3.1.1.2 Intruder Alert

It is also *a rule based intrusion detections system*. Rules and clauses are used to form security policies. These policies are the core to detection misuse or intrusion entering the system. Administrators have the privilege of defining the policies. Further, it maintains a *centralized server* and all the hosts report back the activity occurring in their systems to the central server or the administrator where the analysis of the data is done for intrusion detection.

3.1.1.3 Symantec Intruder Alert

It is also another *rule based IDS*. It monitors systems and networks in real time to detect and prevent unauthorized activities. It also enables the creation of powerful, customizable intrusion detection polices and responses. Like Intruder Alert, it maintains a *central management console* who takes the whole responsibility of analyzing the data for intrusion.

3.1.1.4 TripWire

It is a variant of HIDS which focuses on *file integrity checking*. Normal state is represented by creating a *database of critical system files* in a known secure state. Properties such as last write time, read time, modify time are captured and stored in the database. Finally intrusion is detected by comparing the state of the system at any time with that stored in the database.

3.1.1.5 CyberSafe Centrex

It is also basically a *centralized* and a *policy driven* intrusion detection system. Administrators can define security policies, control target agents as well as perform high level vulnerability assessment.

3.1.2 Anomaly Based HIDS

3.1.2.1 NIDES/STAT

NIDES/STAT means Next-generation real-time Intrusion Detection Expert System Statistical component. It uses statistical models for describing the normal behaviour of the system. The observed behaviour of a subject is flagged as a potential intrusion if it deviates significantly from the subject’s expected behaviour. Profile for each subject is created which reflects the subject’s behaviour. The main problem with this system (and this kind of systems) is that it introduces the possibility of an attacker to gradually train the profile to consider his/her intrusive activities as normal behaviour.

3.1.2.2 Haystack

Like NIDES/STAT it also uses statistical model for profiling the behaviour of the system. Unlike NIDES/STAT, the Haystack algorithm has a step that determines resemblance to known attacks. The advantages are that more knowledge about the possible attacks can be derived from this step and better responses can follow the alarms. However, extra knowledge about possible intrusion types is required. The main problem is that the process of generating the weighted intrusion vectors is time consuming and error prone.

3.2 Research done

Various works and researches have been done on this field. Apart from the existing system described earlier various researchers have done research on the very field. List of some research work that we have come across during our research are listed below.

3.2.1 Intrusion Detection System for Classifying Process Behavior

This is a master thesis done by **Trung [6]**. In his paper he has proposed a host based intrusion detection system. In his paper he has proposed detecting anomaly on process behaviour. He has proposed using SVM, HMM and KNN for anomaly detection and DARPA data sets for training and testing.

3.2.2 Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data

This is a research work done by **Wanga**, **Guana**, **Zhangc** and **Yanga [9]**. The research also proposes a HIDS. The system models the normal behaviour of the host by using the transition and frequency property of the audit data. In their research they have proposed using SOM and HMM to frequency and transitional property respectively. They also have used DARPA data set for validating their system.

3.2.3 Application of SVM and ANN for intrusion detection

This is another research work done by **Chen**, **Hsu** and **Shen [5]**. They proposed a system for detecting anomaly on sequence of system calls. They have used ANN and SVM to build their system. They have used DARPA 1998 BSM audit data to validate their system as well.

3.3 Comparison

We have previously defined the techniques of anomaly detection on background. These are KNN, SVM, SOM, ANN and Markov Model. We have used SOM to capture frequency property and Markov Model for capturing transitional property.

KNN, SVM and ANN can also be used to capture frequency property of data but ANN and SVM use supervised learning for training the model and KNN although uses unsupervised learning is computationally intensive, especially when the size of the training set grows. SOM uses unsupervised learning and has much lesser computational complexity. So SOM is a best option for capturing the frequency property of data. Markov Model are the only model that can capture the transitional property of data.

4 METHODOLOGY

4.1 Block Diagram

The overall system is presented in the block diagram given below.

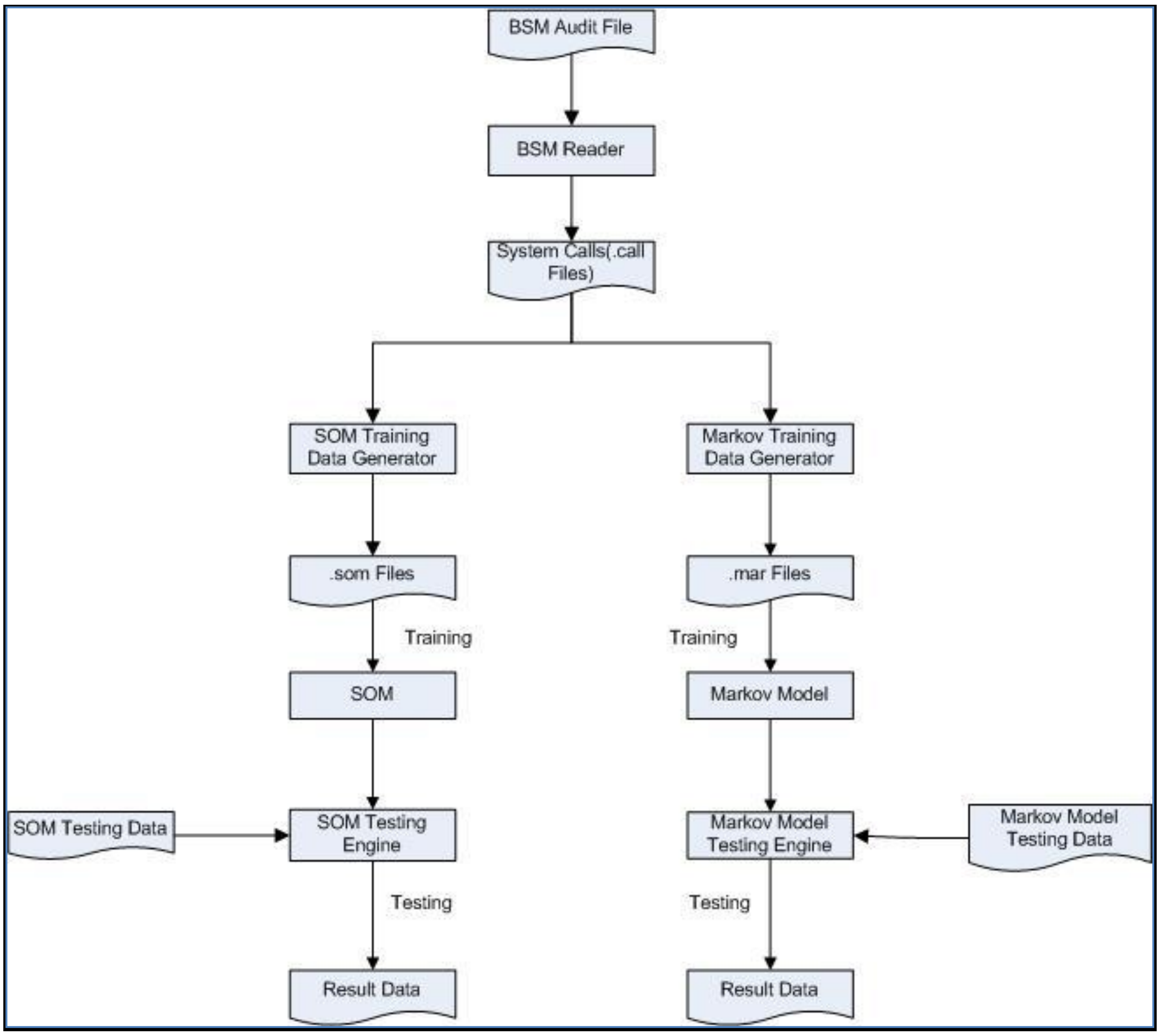


Figure 4.1: System Block figure

4.1.1 BSM Audit Reader

We developed our own BSM reader. The raw BSM audit file i.e. .bms files provided by the solaries machine contains log of system calls. Each entry of the .bsm file is a audit record. The structure of the audit record is shown in Appendix B. Each audit record consist of various tokens. Each tokens consist of specific data of that token. The various token structure used is shown in Appendix B.

BSM Audit data is fed as an input to this block. It then converts each entry to a list of system calls made by it. Now what we get is a file containing list of system calls made by the processes contained in the original BSM file. These files are *.call* files.

We also generate a list of unique processes from the list of processes.

4.1.2 SOM Training Data Generator Engine

The *.call* files are fed to the SOM Training Data Generator Engine. Here, the frequency of system calls of each process contained in the *.call* file is calculated and a *.som* file is generated. This is the file that is used in training of the Self Organizing Map.

4.1.3 Markov Model Training Data Generator Engine

The *.call* files are fed into this engine as well in order to generate training data for the Markov Model. These files are the *.mar* files.

4.1.4 SOM

This is where the Self Organizing Map gets ready. It uses the *.som* as training data and produces a trained model which is stored as a *.strained* file. It is later used in testing.

4.1.5 Markov Model

Here training of the Markov Model takes place. It takes its input training file as *.mar* file and produces a trained network as *.mtrained* file.

4.1.6 Testing

4.1.6.1 SOM Testing

This block takes the input as the *.strained* file and the test dataset. It returns the number of True Positives, False Positives, True Negatives and False Negatives at the given value of threshold.

4.1.6.2 Markov Model Testing

Similarly as the SOM testing, this block takes the input, the *.mtrained* file and the test dataset. It also returns the number of True Positives, False Positives, True Negatives and False Negatives at the given value of threshold.

4.2 Example

The life cycle of a raw BSM audit data as it passes through each block described above as an example.

4.2.1 BSM audit data

header,126,2,open(2) - read,,Tue Jul 28 07:43:17 1998, + 761508026 msec

path,/etc/security/audit_control

attribute,100664,root,other,8388608,62781,0

subject,2110,root,other,root,other,257,257,0 0 172.16.112.50

return,success,4

trailer,126

The above data is a sample taken from the BSM audit data. It contains the detail about the system call open(2)-read called by the process with ID 257 and session ID 257 which can be viewed from subject token.

4.2.2 System Call File

Close ioctl fcntl close ioctl ioctl fork close close read close write fcntl close fcntl close close setuid close close close close close close close exit

112 158 30 112 158 158 2 112 112 72 112 76 30 112 30 112 112 200 112 112 112 112 112 112 112 1

The above numeric sequence is the numeric translation of the above system call sequence. Each system call has unique ID as given in Appendix B. The sequence above is the sequence of system calls generated by the process (Process ID = 257) during its execution.

4.2.3 SOM Training Data

The frequency of the system calls made by the above process is the data presented below. The frequencies of system calls given by above process are given below. The data is generated by feeding the system call file to the data generator.

Close: 15 ioctl: 3 fcntl:3 fork:1 read:1 write:1 setuid:1 exit:1

The data given below is the actual training sample. It contains total of 63 fields because the unique number of system calls present overall is 63. So, each field represents the count of each unique system call.

0 0 3 1 15 0 0 1 0 0 0 0 1 0 0 0 0 0 3 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Each numeric field in the above sample denotes the frequency of individual system calls made by the above mentioned process that is having process ID 257. The more samples of training data for SOM are shown in Appendix C.

4.2.4 Markov Model Training Data

The data given below represent the training data for markov model. It is the sequence of the system call given by the process.

112 158 30 112 158 158 2 112 112 72 112 76 30 112 30 112 112 200 112 112 112 112 112 112 112 1

The more samples for training Markov Model is shown in Appendix C.

4.2.5 Test Data

The test data are similar to the training data for both SOM and the Markov Model.

5 IMPLEMENTATION TASKS

5.1 Data generation

Our system is an intrusion detection system which detects intrusion on the host by process mining. Since our system is an anomaly detection system, we first define the normal behaviour of the system and detect the deviation for the normal behaviour. We use semi-supervised learning for training our system that is we only feed intrusion free data to our system during the training period.

We have used the DARPA 1998 data sets for training and testing of our system. Most operating systems offer some level of auditing of operating system events. The amount of data that is collected could be as limited as logging failed attempts to log in, or as verbose as logging every system call. Basic Security Module (BSM) data from a Solaris victim machine was collected and distributed as part of the DARPA evaluation data.

The BSM file consists of system call log generated by the victim Solaris machine. It consists of audit record. The audit record structure is shown in Appendix A. An audit record is a sequence of audit tokens. Each token contains event information such as user ID, time, and date. Logically, each token has a token type identifier followed by data specific to the token. Each token type has its own format and structure. The structures of different tokens are shown in Appendix A. There are different tokens in the record but the header token and subject token are the important token for our project. We have extracted the system call name, and timestamp of the call from header token and process ID and session ID is extracted form the subject token. A record starts with the header token. In header token, position 4 is the system call and position 6 is the timestamp of this system call. There are several other tokens, but those tokens are not important for this project. The other important token is subject token. Subject token includes audit ID, effective user ID, effective group ID, real user ID, real group ID, process ID, session ID, terminal ID (including port ID and machine address). The important fields in subject token are process ID and session ID.

In each BSM file for a day there are different sessions. Each session consists different processes and each processes consists of different system calls. At the first step we extract different processes with list of system calls generated by each processes. Individual sessions can be programmatically extracted from the BSM audit data. Each session consists of one or more processes. A complete ordered list of system calls is generated for every process. A sample system call list is shown below. The first system call issued by Process 994 was close, execve was the next, then open, mmap, open and soon. The process ended with the system call exit.

Process ID: 994

close execve open mmap open mmap mmap munmap mmap mmap close open mmap close open mmap mmap munmap mmap close close munmap open ioctl access chown ioctl access chmod close close close close close exit

In DARPA data, there are list files. These files specify sessions which must be scored by intrusion detection systems. These list file consists of various sessions which are labelled as intrusion and non-intrusion. The start time and the duration of the session can be extracted from the list file. By using this we can label the processes as the normal or abnormal process.

DARPA provides seven week of training data and two weeks of test data. In these seven weeks of training data, there are fifteen days free of attack. In these fifteen days, Tuesday of third week, Wednesday of fourth week, Thursday of fifth week and five continuous days in seventh week are free of attack. These eight days are used to build a data for normal process data. In these eight days seven days are used for training data and one day that is Friday of seventh week is used as test data.

The converting process also generates a file that contains the list of unique system calls in those seven days. The list of the unique system call is stored in ***unique.sys*** file.

After doing all the extractions from BSM files, there are **1908** normal processes used as training data, **530** normal processes used as test data, and **9** abnormal processes used as test data. **63** unique system call was extracted from the seven week training data. This can be tabulated as:

Table 5.1: Data statistics used for training and testing

Normal Processes used as training data	1908
Normal Processes used as test data	530
Abnormal Processes used as test data	9
Unique System Calls Number	63

5.1.1 List of Attacks

Table 5.2: List of Attacks

DoS attack	**Process table**
	Mailbomb
Remote to user	**Ftpwrite**
	Xlock
	Xsnoop
User to superuser	**Eject**
	Ffbconfig
	Fdformat
	Ps
Surveillance / probing	Mscan
	Saint
Others	**Warez**
	Rootkit
	Httptunnel

The above table lists the various attacks provided the DARPA data sets. We have only extracted the attacks whose names are written in bold on above table.

Data for SOM

Frequency-based methods model the frequency distributions of various events. For the system-call application, the events are occurrences of each pattern of system calls in a sequence.

The next step is converting each process into a vector by using frequency weighting technique. For this we read the list of unique system calls generated previously. For each distinct system call in a process, the number of occurrences of that call is counted. The number of elements in a vector is the number of distinct calls in the whole normal process data set. Each element in a vector is the number of occurrences of the system call which is corresponding to the position of the element in the vector.

Data for Markovian Model

Working on Markovian Model, the ordered sequence of system calls is the main and important information of a process. Each process will be converted in to a vector. Its length is the number of system calls in the process. Each entry in the vector is a number corresponding to the system call at this position. Each system call has its own number which is its order in the unique system calls file.

5.2 Training

We used DARPA 1998 dataset for the purpose of training our system. The data is first made ready by the ***data generation engine*** discussed in the earlier section. The following section contains the details of the implementation involved in training of our system which includes training both the SOM network and the HMM network.

5.2.1 Self Organizing Maps

SOM is a topology preserving neural network and a method of dimensionality reduction. It captures the frequency property of the data. The training data generated from the ***data generator engine*** contains a total of *64* dimensions, each dimension characterized by the system call each process calls. Using SOM we map those dimension to a 2 x 2 matrix topology. Upon training, similar processes get clustered in the topology and then we can use distance metrics to test new process data.

The basic idea of the SOM is to make available a certain amount of classificatory resources which are organized based on the patterns available for classification i.e. input patterns. A very relevant characteristic of SOM is the development of an ordered network in which nearby neurons share similarities, this way patterns which are similar will activate similar areas in the SOM. Hence different parts of SOM will be activated by specific types of input patterns. This leads to a global organization and local specialization. Generally, a SOM is trained iteratively through a large number of epochs. An epoch consists of processing all the input patterns once.

Each node of the SOM is a neuron which has a capability of representing processes in our context. When we train the SOM with the given input pattern, we calculate the distance between that specific pattern and every neuron in the network. We then select the neuron that is closest as the ***winning neuron*** and say that the pattern is mapped onto that neuron or that the neuron has won the representation of that input pattern. This results in the map neuron moving towards the input pattern to improve its representation. The extent of this movement is controlled by a parameter referred to as ***learning rate.***

Not only the winning neuron but also the neighborhood of the winning neurons is also corrected. This way the network is progressively organized with certain parts of input space being represented by certain subsets of neighboring neurons. The factor that takes into account the area in the map to be taken as neighborhood is defined by ***neighborhood radius***. If the SOM has been trained successfully, then the patterns that are close in the input space will be mapped to the neurons that are close in the output space, and vice-versa.

5.2.1.1 Training Algorithm

Let x_k the n-dimensional training patterns. Here k starts from 1 to N, the total number of training patterns.

Let w_{ij} be the neuron in position(i, j).

Let the learning rate be in the range$0 \leq \propto \leq 1$.

Let N_c be the neighborhood function that is high for neurons that are close in output space, and small (or 0) for neurons far away in the map.

Let w_{winner} be the winning neuron for a given input pattern.

Then, for each input pattern:

1) Calculate the distance between the pattern and all the neurons in the map ($dij = |xk - wij|$)
2) Select the nearest neuron as winner w_{winner} ($wij : dij = \min(d)$)
3) Update each neuron according to the rule ($wij = wij + \propto Nc\ |xk - wij|$)
4) Repeat the process for a fixed number of iterations.

5.2.1.2 Variations Possible

The parameters that can be altered and the possibilities with those are:

5.2.1.2.1 Distance Metric

- Euclidean Distance
- Minkowski Distance

5.2.1.2.2 Learning Rate

- Constant Learning Rate
- Linear Learning Rate
- Exponential Learning Rate

5.2.1.2.3 Topology

- Hexagonal Topology
- Matrix Topology

5.2.1.2.4 Neighbourhood Radius

- Gaussian Neighbourhood Function

5.2.1.3 Our Implementation

In the model that we have developed, we have used the following to train the SOM:

5.2.1.3.1 Distance Metric

We have used Euclidean distance metric to compute the distance between the input pattern and the nodes of the SOM.

5.2.1.3.2 Learning Rate

We have used Constant Learning Rate to train our network. Further we have applied variations in the learning rate to optimize our trained model.

5.2.1.3.3 Topology

We have used Matrix Topology with variations in the topology size i.e. the dimension of the map.

5.2.1.3.4 Neighborhood Function

We have used Gaussian Neighborhood Function as the neighborhood function. The Gaussian function that we have used is:

$$e^{\frac{-d^2}{2r^2}}$$

Where d is the distance of the neighboring neuron from the best matching neuron and

r is the initial radius of the neighborhood.

This means that the neurons that are nearer have a greater degree of learning than the farther neurons.

Furthermore, the radius that defines the neighborhood boundary also decreases with iteration exponentially as:

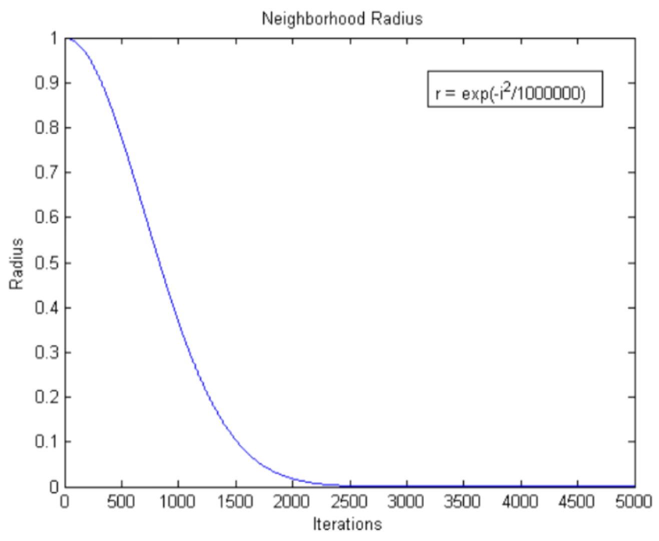


Figure 5.1: Graph of neighborhood function

$$r_{new} = r_{old} e^{-\frac{iteration^2}{1000000.0}}$$

Hence if the current iteration is *i* and the distance between the $winner$ neuron, the input vector v is d and the current neighboring radius is *r*, then the weight of the input vector after adjustment is:

$$w_{new} = w_{old} + \left(0.8 * e^{\frac{-d^2}{2r^2}}\right) * (v - w_{old})$$

5.2.2 Markovian Model

This model which is a probabilistic approach captures the transitional property of the data. We can assume states which represent the features of the data. In our case, each state represents the system calls made by the processes. Each state can be thought of as a generator of that particular system call.

Training sequences are generated using the ***data generator engine*** discussed earlier in the data generator section. The training sequences are fed into the probabilistic model and the transitional probabilities associated each state with itself and others are estimated.

5.2.2.1 Algorithm

Let $x_{1:k}$ be the training sequence where k is the length of the training sequence.

Let a_{ij} be the transitional probability associated with transition from state *I* to state *j*.

Now given $x_{1:k}$ the transitional probability a_{ij} can be computed as:

$$P(m,n) = a_{mn} = \frac{count(m,n)}{\sum_{var=1}^{k} count(m,\ var)}$$

Where,

Count (m,n) is the count of the occurrences of the system calls (m, n) in the order.

Hence by counting the occurrence of transition, say from m to n, and then dividing the count by the overall count of the transition from the state m to any other state, we can find the transitional probability from state m to n.

Once we have the transitional probabilities, we can calculate the probability of the model generating any sequence of interest i.e.

$$P(\lambda/x_{1:k})$$

Can be calculated and by comparing the probability with a threshold, we can find out the validity of the test sequence.

5.3 Testing

In 1998, DARPA funded an intrusion detection evaluation program at Lincoln Laboratories. The training datasets consisted of seven weeks while testing datasets consisted of only two weeks. BSM data in Solaris Victims are used for testing our system. There are two types of test data – One for testing SOM module and other for HMM module.

5.3.1 Generation of Test Data

MIT Lincoln Library provided two weeks of test data for testing intrusion. It consists of a BSM and a list for each day. The list files specify the sessions. Each row in a list file describes a single session. There are eleven columns in each row, and each column defines different attributes of the session. Following are some rows of a List file.

13308 07/24/1998 10:04:47 00:00:01 smtp 16451 25 135.008.060.182 172.016.112.050 0 -

13583 07/24/1998 10:06:35 00:01:00 telnet 16496 23 194.007.248.153 172.016.112.050 1 format,35,,y,u2r,stealthy,old,script*

13783 07/24/1998 10:08:42 00:00:01 smtp 9066 25 172.016.114.148 172.016.112.050 0 –

Listing: List file sample for BSM.

Column 1: Unique Session ID.

Column 2: Starting date of the session.

Column 3: Starting time of the session.

Column 4: Duration of the session.

Column 5: Service name used by this session.

Column 6/7: Source/Destination port.

Column 8/9: Source/Destination IP.

Column 10: Score whether the session has attack or not.

Column 11: Attack details if there is any attack in the session.

The labeling of the test data is the major part in testing of SOM. First of all, the sessions with attacks are identified for each day. Then, within the session duration the processes are scanned in respective BSM audit file to see if there exists process making attack manually. Then the infected process is labeled whether it is an attack or not. This is confirmed by exploring the nature, type and behavior of attack described in infected session in list file. The details about different types of attacks are explained in literature review.

Like generation of training data, test data for SOM are generated by counting various types of unique system calls for each unique process. SOM takes a vector consisting count of unique system calls for a process along with label attached at the end of the vector. SOM declares the process whether it is infected or not based on the training data fed to the network. Here is the input vector format for testing SOM.

[Count 1, Count2,…, Label] – Input format for SOM

Here, Count1 represents the total number of count of one type of system call; similarly Count2 is the number of count of another unique system call and so on.

For testing intrusion using HMM, we make a vector of sequences of system calls made by each unique processes. Each system calls are denoted by their unique IDs. In the end, label of the process is also attached to notify whether the process is attack free or not. Label is 1 for the attack process and 0 for normal process.

[SysCall1, SysCall2,..., Label] – Input format for HMM.

5.3.2 Detection of Intrusion

- Using SOM: The vector generated above is fed into the SOM model. The training of the model generates a table of best matching units. If input vector doesn't lie in best matching unit, then it's declared as intrusion. If input vector matches best matching units, then distance of the vector from the unit is calculated. If the distance thus calculated is less than the threshold, then input vector is intrusion free, otherwise it is intrusion vector and the respective process is infected and labeled as an intrusion.
- Using HMM: The vector generated above is fed into the HMM. The training of the model defines the probabilities of transition between the states. The transition between the states is simulated by sequence of system calls. So, the sequence of system calls of a process to be tested when fed into the HMM, according to the probability defined by Model via training, the probability of input vector is calculated. Input vector should be of fixed length, otherwise the vector should be divided into windows of fixed length and probability for each window is calculated. If the probability is below threshold, then the input vector is anomalous otherwise the vector is intrusion free.

Let, $x_{1:k}$ is a test sequence, ϴ be the threshold and λ denotes the model.

The probability of the sequence generated by the model λ is given as,

$$\begin{aligned} P(x_{1:k}/\lambda) &= P(x_2/x_1) * P(x_3/x_2) * P(x_4/x_3) * \ldots * P(x_{k+1}/x_k) \\ &= a_{12} * a_{23} * a_{34} * \ldots * a_{k+1,k} S \end{aligned}$$

Where a_{ij} = Transition from state i to state j

If P ($x_{1:k}/\lambda$) $\leq \Theta$ then the sequence can be generated by λ,

Else the λ doesn't generate the sequence $x_{1:k}$.

5.4 Tools Used

- Java as programming language
- Netbeans as IDE
- 1998 DARPA Datasets for training and testing
- Training Data
    - Monday of week 3 training data
    - Tuesday of week 4 training data
    - Wednesday of week 5 training data
    - Monday, Tuesday, Wednesday and Thursday of week 7 training data
- Testing Data
    - Friday of week 7 training data for normal or intrusion free processes
    - Week 1 and week 2 of test data for intrusion processes

6 RESULT AND ANALYSIS

6.1 ROC curve

ROC (receiver operating characteristic)

An ROC curve is a two-dimensional depiction of the accuracy of a signal detector, as it arises on a given set of testing data. Two dimensions are required to show the whole story of how the true-positive rate of detection decreases as the false-positive rate of error increases.

An ROC curve plots the true-positive rate of detection, or *TP rate*, against the corresponding false-positive rate of error, or *FP rate*. These two numbers change in relation to each other (determined by theory or experiment) as the detection threshold, or decision cut-off, varies. Whenever the rate of true positives is the highest, the rate of false positives is the lowest, and the reverse is also true.

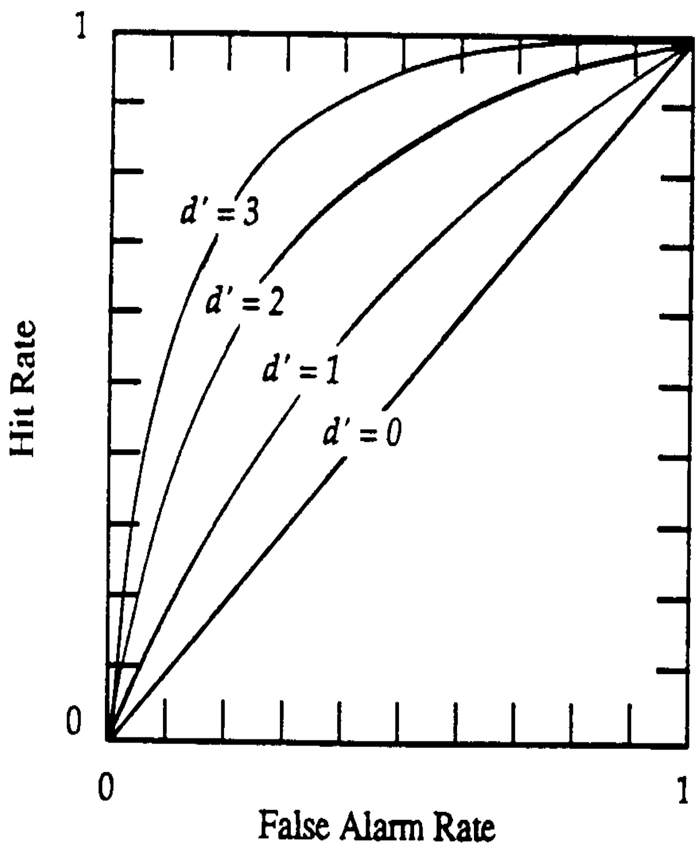
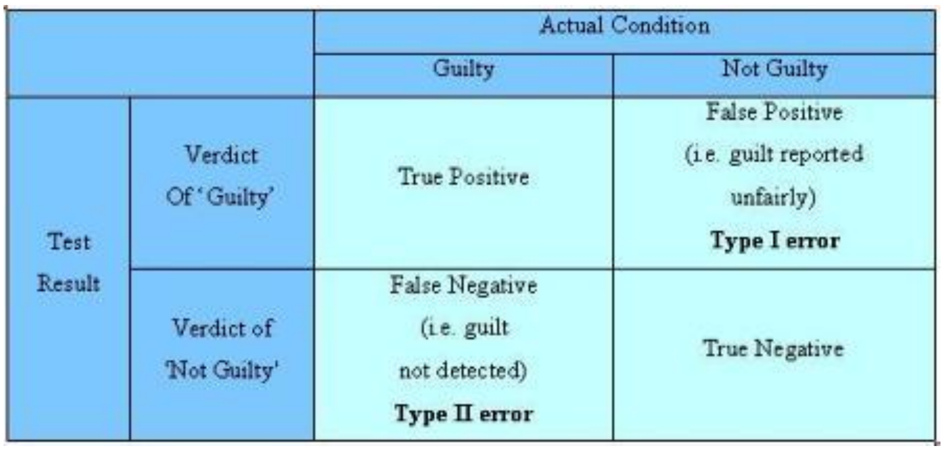


Figure 6.1: ROC curve

In figure, different shapes of ROC curves indicate different levels of detector accuracy. The better (more accurate) curves will bulge further outward to the upper-left, nearing the point of perfection at (0,1). A perfect detector will have a success rate of 1.0 for signals while having an error rate of 0.0 for noises. Unfortunately, this result is difficult to achieve, so most ROC curves will tend to look less angular and more gently bowed, like the middle two curves shown (d' = 3 and d’ = 2).

When the *TP rate* is much higher than the *FP rate*, over more values of *TP rate*, then the curve will have a more extremely bowed shape and represent the performance of a superior detector.

Table 6.1: Types of Results



		Actual Condition	
		Guilty	Not Guilty
Test Result	Verdict Of ‘Guilty’	True Positive	False Positive (i.e. guilt reported unfairly) **Type I error**
	Verdict of ‘Not Guilty’	False Negative (i.e. guilt not detected) **Type II error**	True Negative

Basic needs (assumptions) of ROC curves:

• Source data must be classifiable into two categories, signals and noises1

• The “ground truth” label for each data event is available, i.e., knowledge about what is really a signal, and what is really a noise

TP rate = # of times the detector labelled a signal event as a signal / # of signal events in evaluation data

FP rate = # of times the detector labelled a noise event as a signal / # of noise events in evaluation data

6.2 Results

We can use anomaly detection technique for binary classification that is normal group and abnormal group. The data set contains mostly the normal data, and rarely the abnormal data. The abnormal data are qualitatively different from the normal data. Since the abnormal data are both different from normal and are rare, they will appear as outliers in the data which can be detected.

The benefit of anomaly detection technique is it can work without the need for massive sets of pre-labelled training data and has the added versatility of being free of labelling different kinds for abnormal data. We only consider normal and abnormal data, so if there is a new kind of abnormal data, it may be detected not as a specific kind of abnormal but just as an abnormal data.

6.2.1 Self Organizing Maps

We trained the SOM network with DARPA 1998 dataset. We varied the training environment so as to find out the optimal network parameters that can best fit the dataset being trained. The training consisted of a total of ***530 intrusion free data*** and a total of ***9 intrusion data.*** We varied the threshold distance to obtain the ROC curve.

The experiments we conducted are presented below:

6.2.1.1 Variation 1

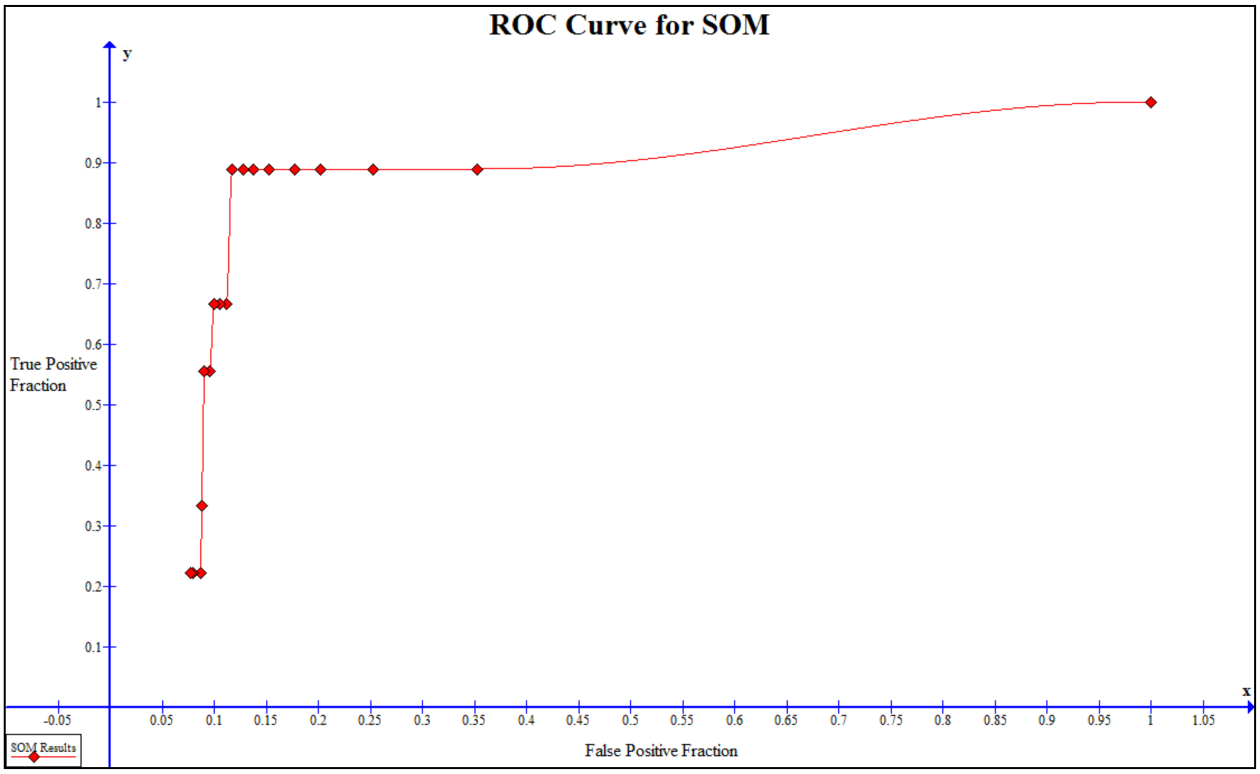


Figure 6.2: ROC curve for SOM (dimension: 25*25)

Description:

During training the parameters used are as follows:

Map dimension = 25 x 25

Neighborhood Radius (Initial) = 10

Number of Iterations = 5

Learning Rate = 0.95

We obtained the optimum detection rate at a threshold of 400 Euclidean distance.

Intrusion free count	Intrusion count	True Positive	False Positive	False Negative	True Negative
530	9	8	62	1	468

6.2.1.2 Variation 2

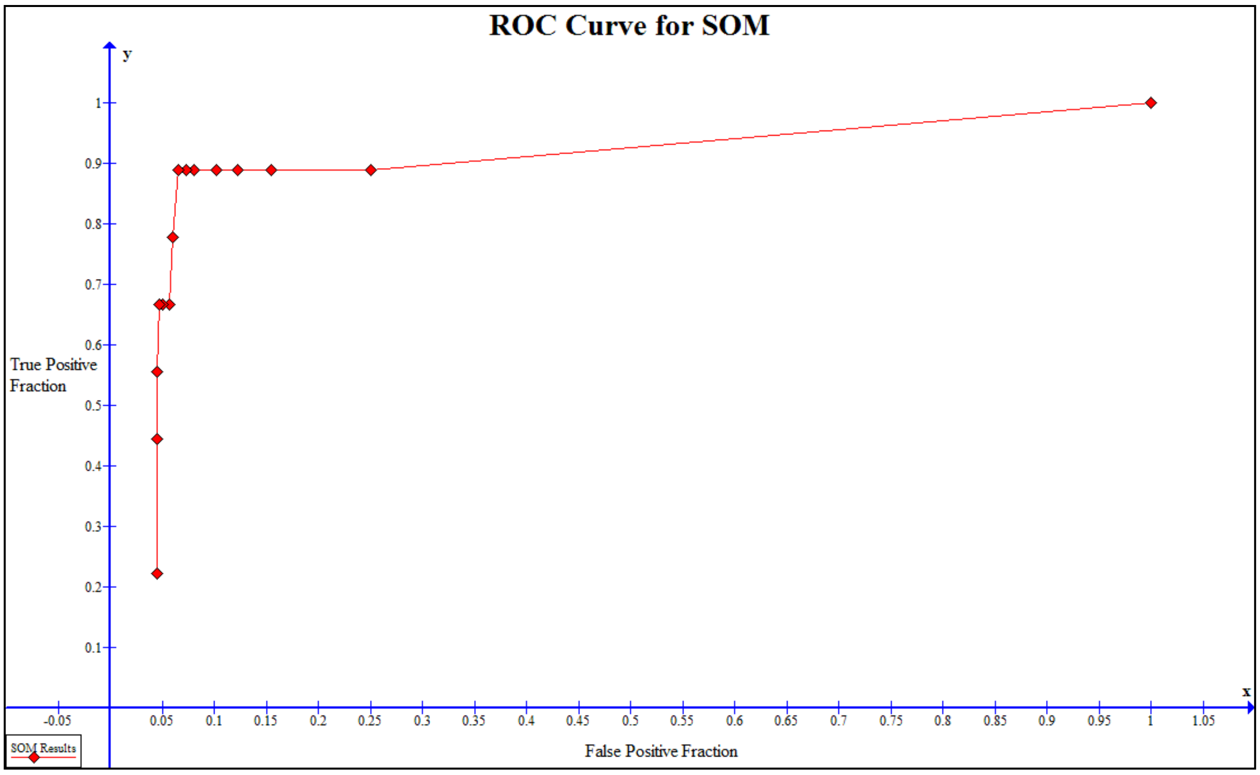


Figure 6.3: ROC curve for SOM (dimension: 25*25)

Description

Training parameters were as follows:

Map Dimension = 50 x 50

Neighborhood Radius (Initial) = 10

Number of Iterations = 5

Learning Rate = 0.8

In this model we obtained the optimal detection rate at threshold of 350 Euclidean distance and the statistics at the distance threshold is presented below:

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	8	35	1	495

6.2.1.3 Variation 3

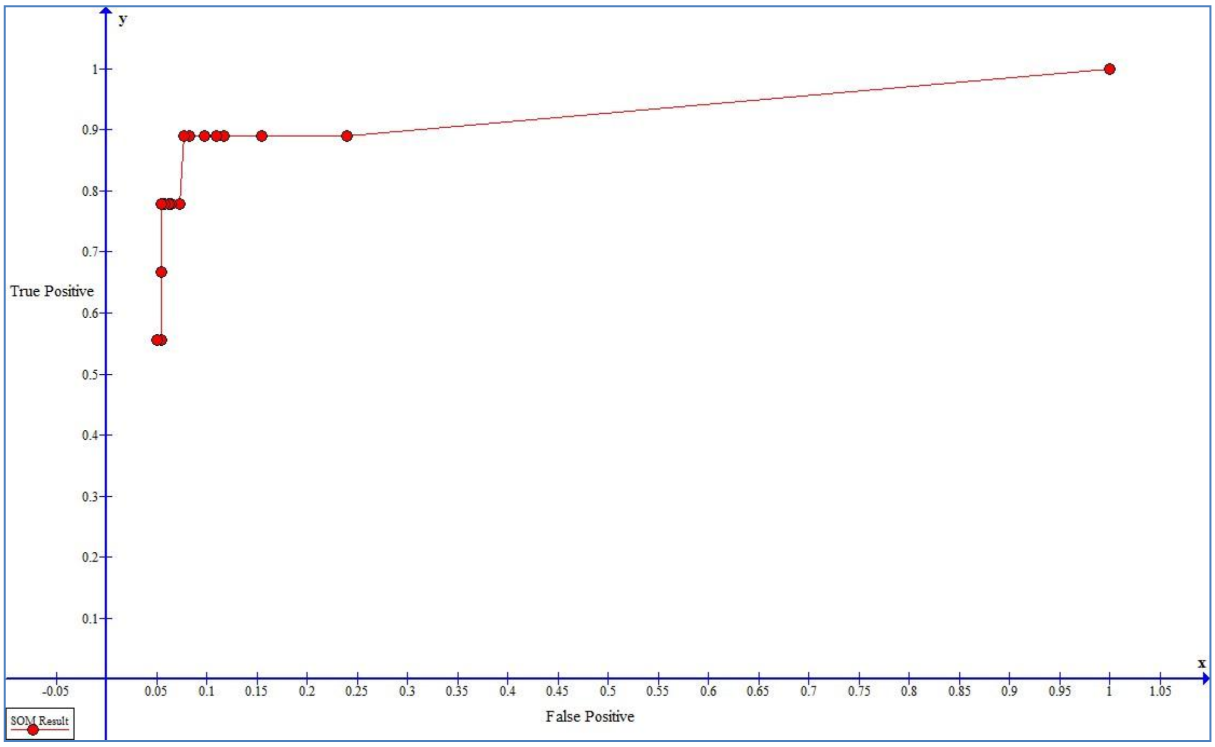


Figure 6.4: ROC curve for SOM (dimension 50*50)

Description

The parameters used during this mode of training were:

Map Dimension = 50 x 50

Neighborhood Radius (Initial) = 12

Number of Iterations = 5

Learning Rate = 0.8

The optimal value was obtained at a threshold distance of 350.

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	8	41	1	489

6.2.1.4 Variation 4

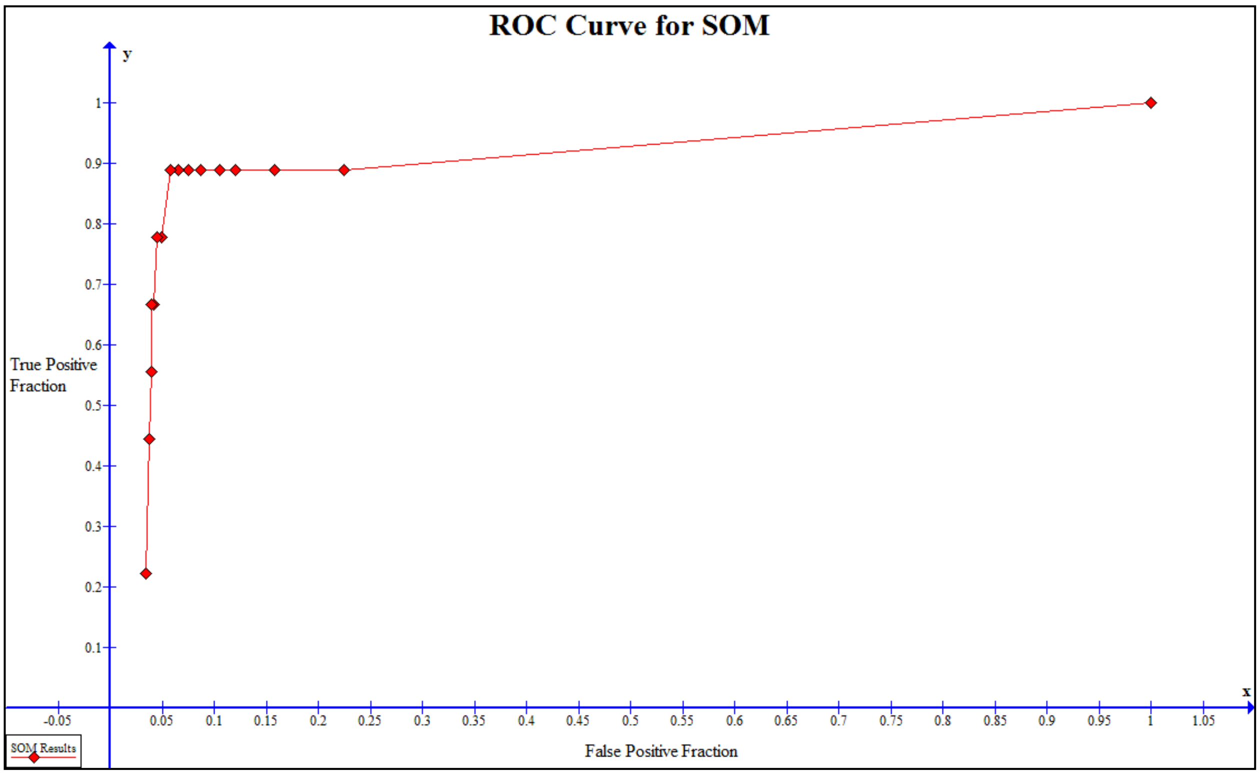


Figure 6.5: ROC curve for SOM (dimension: 50*50)

Description

The parameters used during this mode of training were:

Map Dimension = 50 x 50

Neighborhood Radius (Initial) = 10

Number of Iterations = 5

Learning Rate = 0.95

The optimal value was obtained at a threshold distance of 400.

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	8	31	1	499

6.2.1.5 Variation 5

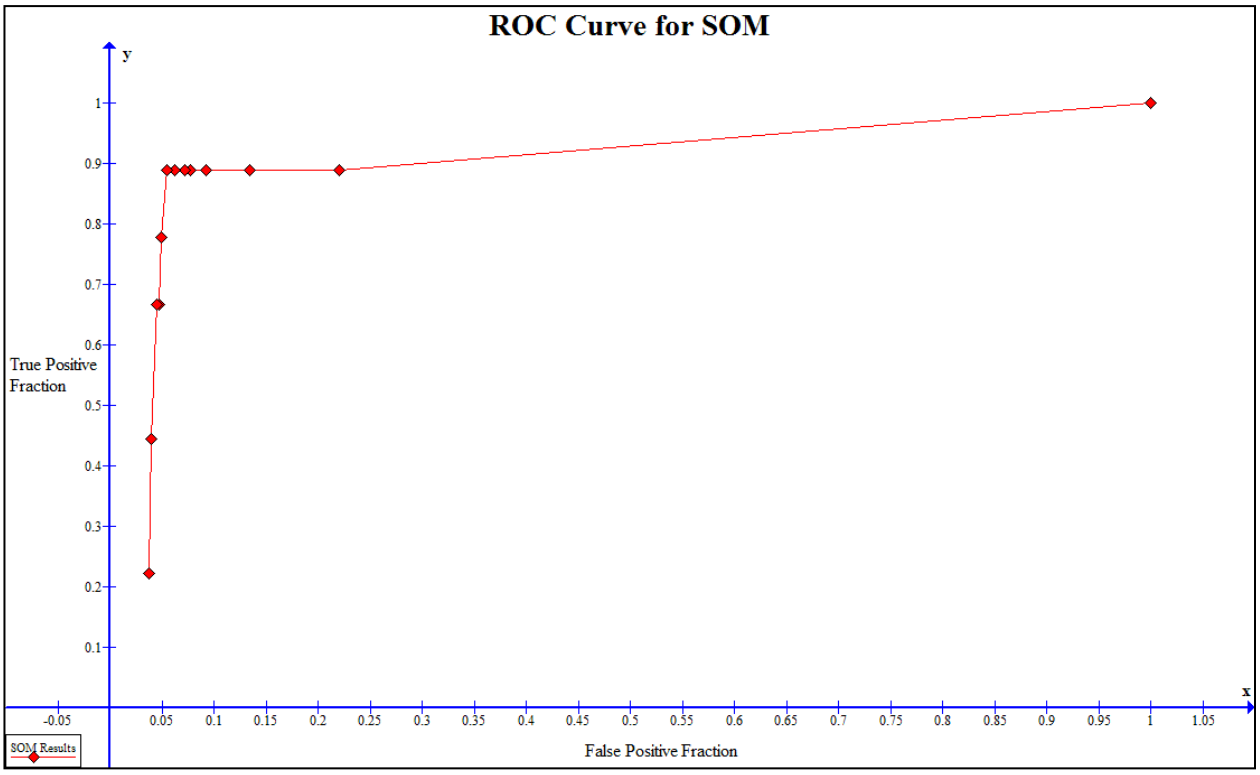


Figure 6.6: ROC curve for SOM (dimension: 100*100)

Description

Map Dimension = 100 x 100

Neighborhood Radius (Initial) = 10

Number of Iterations = 5

Learning Rate = 0.95

Optimal Threshold Distance = 350 and at this threshold the detection statistics are:

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	8	29	1	501

6.2.1.6 *Combined Result*

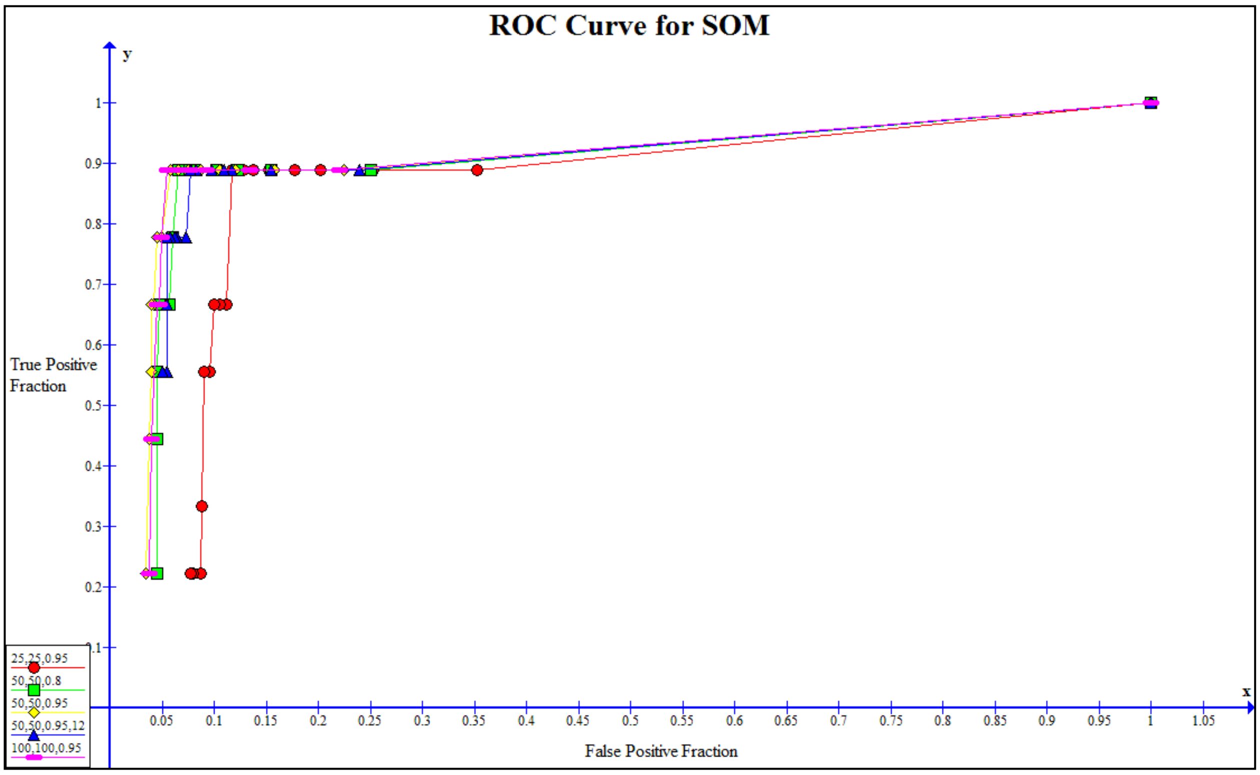


Figure 6.7: ROC curve for SOM

6.2.1.7 *Findings*

By varying map dimension, neighbourhood function, learning rate and the threshold distance, we obtained results as summarized as below:

- For 25 x 25 topology, Detection Rate = 88.89% and False Positive Rate = 11.59%
- For 50 x 50 topology, Learning Rate 0.95, Detection Rate = 88.89% and False Positive Rate = 6.60%
- For 100 x 100 topology, Learning Rate 0.95, Detection Rate = 88.89% and False Positive Rate = 5.57%

6.2.1.8 *Analysis*

From the experiments performed, we found out that as the topology is increased, the False Positive Rate decreased, meaning that the map more accurately fits the training data. The FPR for 100 x 100 topology and 50 x 50 topology is similar. This can be reasoned by the fact that the number of distinct processes that we have used in building up the SOM is about 2000 and a 25 x 25 topology has only a total of 625 nodes. Whereas a 50 x 50 as well as a 100 x 100 topology both have nodes 2500 and 10000 respectively which are well above the distinct processes we are trying to model i.e. about 2000. So they have similar and more accurate FPR.

We also found out that that by altering the value of learning rate we found some increase in FPR. In the 50 x 50 topology, at 0.95 learning rate, the FPR was 5.8% whereas at 0.8 learning rate, the FRP was 7.07%.

Moreover the larger the size of the map, the higher time the model takes to train and to test. So there is an inherent trade-off between detection accuracy and the computational performance. One increases at the cost of the other.

One intrusion among nine is always missed by the model. This is due to the fact that its frequency property exhibits features similar to other normal processes. But this intrusion gets detected by the other transitional model.

6.2.2 Markovian Model

With transitional model, a process is treated as a sequence of system calls. Sequence of each process does not need to have the same length or number of system calls. This method is based on the assumption that sequence of system calls of a process exposes the behaviour of that process. We assume that normal processes have different behaviours from abnormal processes expressed in their sequence of system calls and their transition. 1908 normal processes are used to train a transitional model. This model models the normal process behaviour or transitional property of sequence of system call of normal processes. Sequences of system calls of abnormal processes are considered as out-liers which are different from the rest of normal processes.

Since during the probability calculation if we calculate the probability of the sequence of system calls as a whole then the process with longer length of calls will have lower probability than that of the process having fewer calls thus we have used the ***fixed length window method*** to calculate the probability of the sequence. We calculate the probability of the calls within the window if the probability is lesser than the threshold the process is termed as abnormal else the window is *slided* and again the probability is calculated.

The trained model is used to test with 530 normal testing processes and 9 abnormal testing processes. This test is calculating the likelihood of a sequence, given the trained Model. The model consists of 63 states which is the unique number of system calls seen during training period.

By varying the threshold, we can have different detection rates and false positives. Thresholds are chosen in [−100-0]. A ROC curve can be drawn from list of different detection rates and false positives. By varying the length of window for calculating the probability we can generate different ROC curves. The different results after adjusting the length is shown below.

6.2.2.1 Variation 1

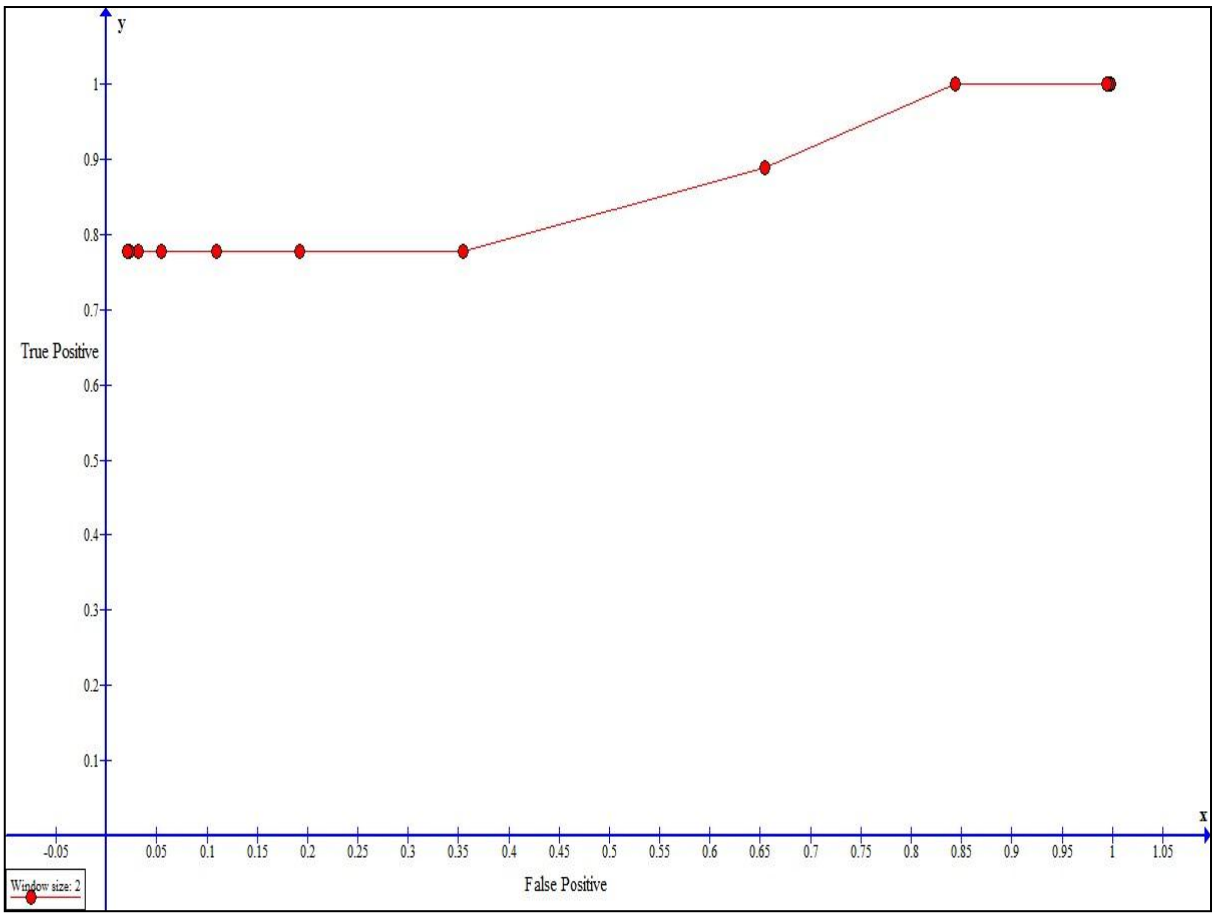


Figure 6.8: ROC curve for Markov Model (window size 2)

Length of window for probability calculation = 2

Optimal threshold probability = -14.0

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	7	11	2	519

Here in this we used the window size of two i.e only two consecutive system calls are taken in the sequence. The probability of the sequence is calculated and the graph of true positive and false positive is drawn according to the system output.

6.2.2.2 Variation 2

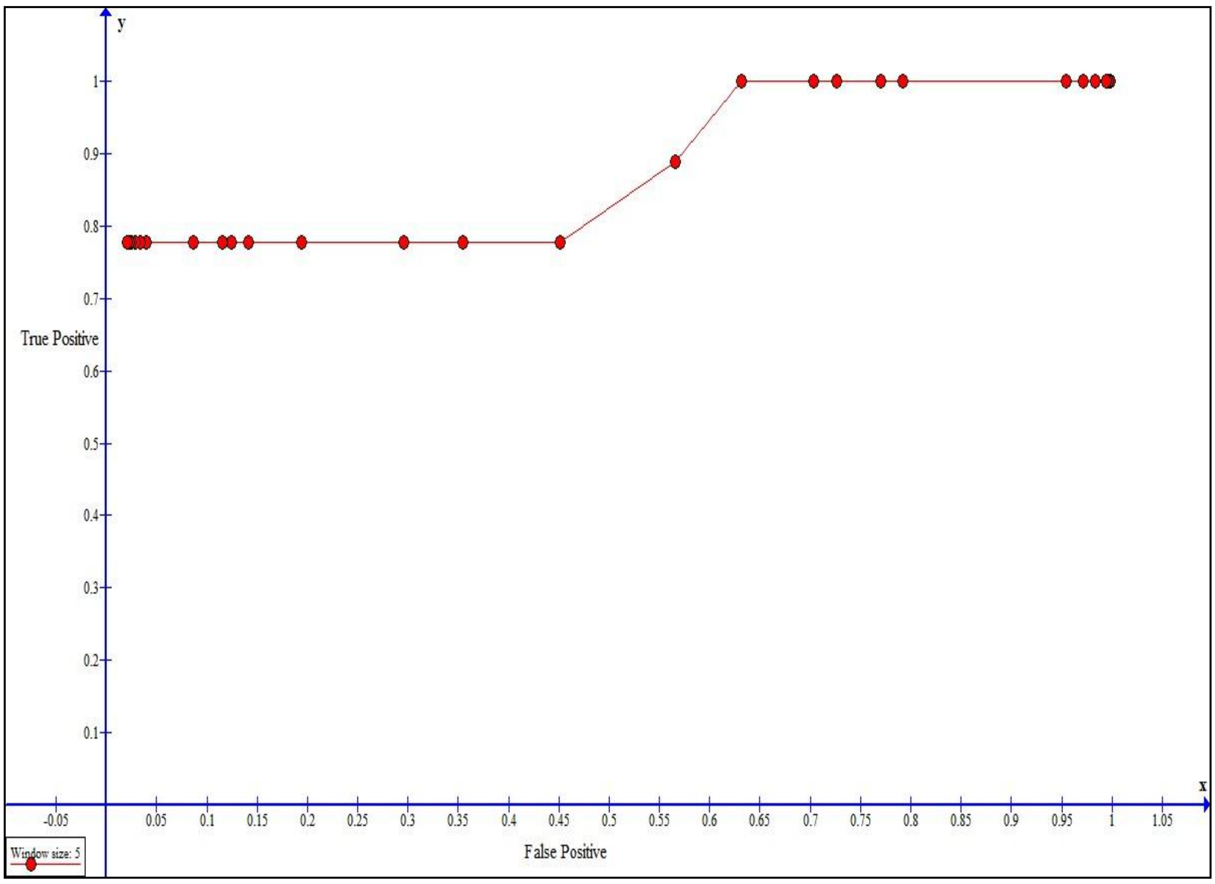


Figure 6.9: ROC curve for Markov Model (window size 5)

Length of window for probability calculation = 5

Optimal threshold probability = -30.0

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	7	11	2	519

6.2.2.3 Variation 3

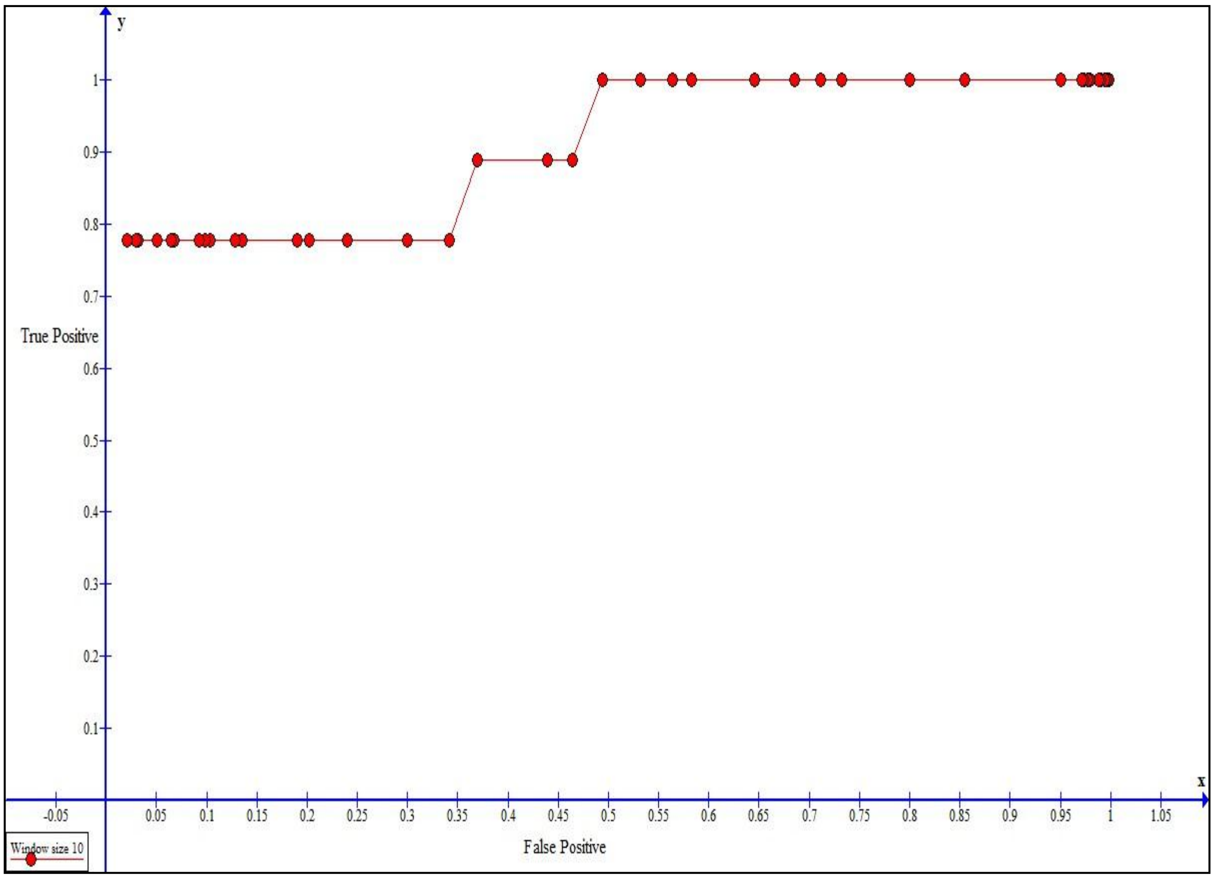


Figure 6.10: ROC curve for Markov Model (window size 10)

Length of window for probability calculation = 10

Optimal threshold probability = -46.0

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	7	11	2	519

6.2.2.4 *Variation 4*

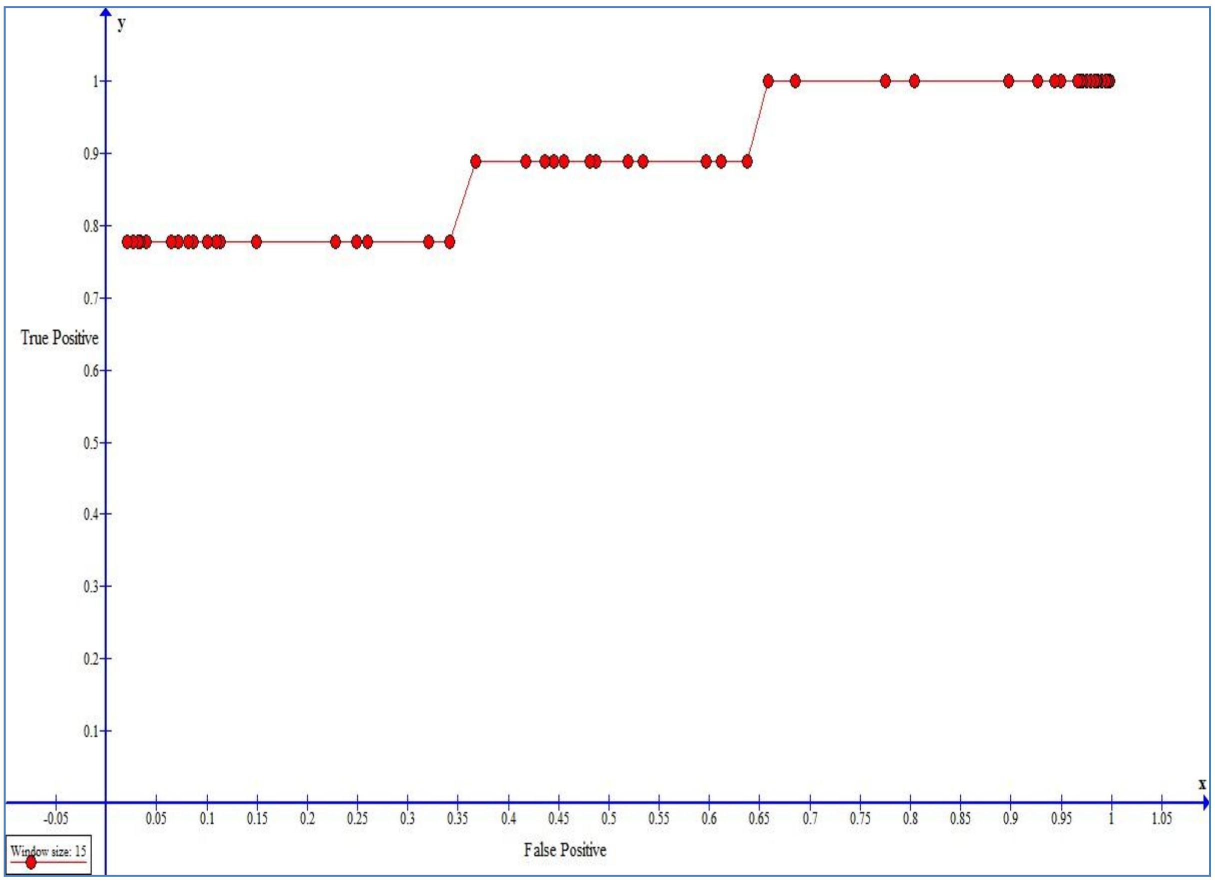


Figure 6.11: ROC curve for Markov Model (window size 15)

Length of window for probability calculation = 15

Optimal threshold probability = -64.0

Intrusion Free Count	Intrusion Count	True Positive	False Positive	False Negative	True Negative
530	9	7	11	2	519

6.2.2.5 Overall Analysis

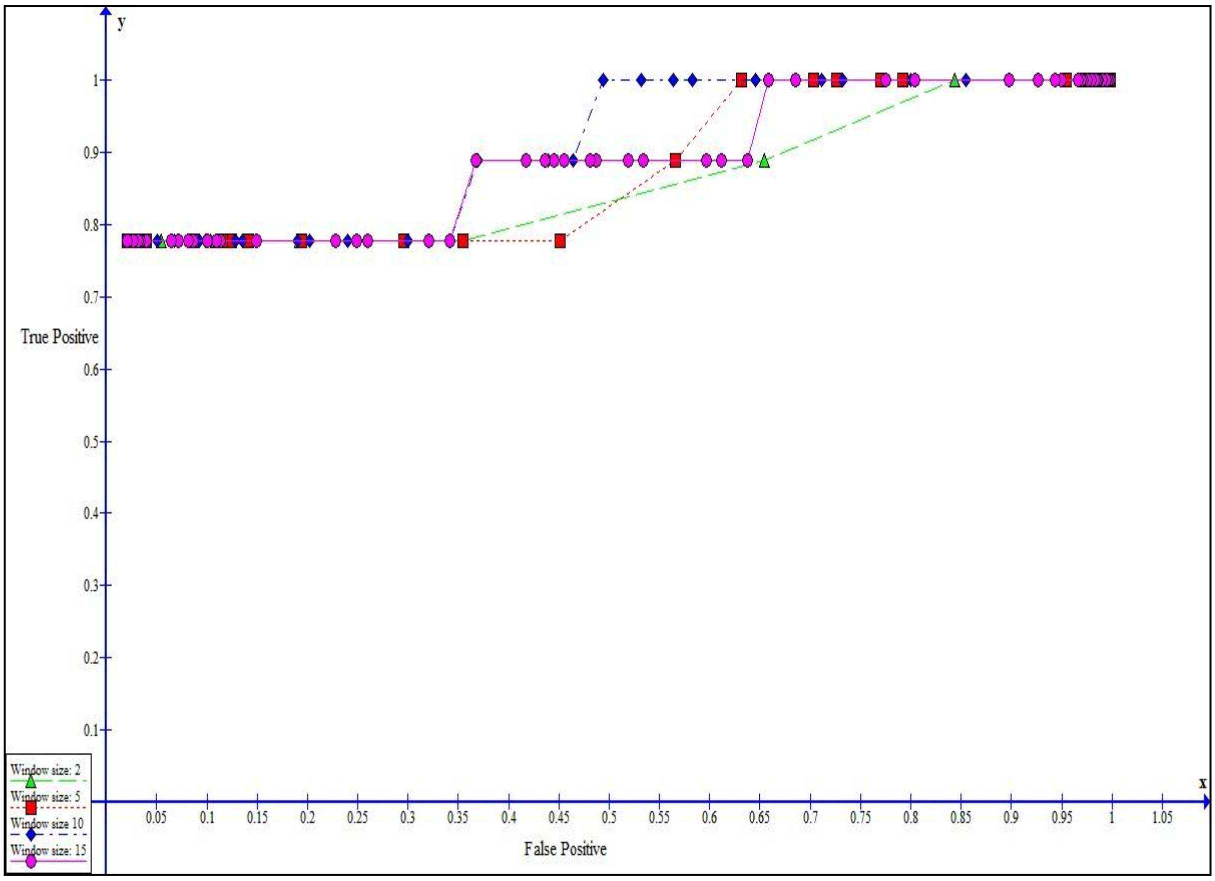


Figure 6.12: ROC curve for Markov Model

6.2.2.6 Findings

The graphs above are the result of varying the window size during testing. For plotting the graph the threshold which is the **log probability** was varied from 0.0 to -100.0. The results that we got are summarized below:

- For window size: 2, Detection rate = 77.78% and False Positive rate = 2.07%
- For window size=5, Detection rate = 77.78% and False Positive rate = 2.07%
- For window size = 10, Detection rate = 77.78% and False Positive rate = 2.07%
- For window size = 15, Detection rate = 77.78% and False Positive rate = 2.07%

6.2.2.7 Analysis

Here we see that at threshold -100.0 we get a true positive rate of about 77% and false positive rate of about 3% and as the threshold is increased the false positive rate increases rapidly. As we seen from the graph the complete detection that is 100% detection of intrusion can be active by window size 10 with minimum false positive rate of 50%. The false positive rates for 100% detection at various window sizes are listed below:

- For window size 2: False positive rate = 85%
- For window size 5: False positive rate = 63%
- For window size 10: False positive rate = 50%
- For window size 15: False Positive rate = 65%

But for others for 100% detection the false positive rate is high. The minimum false positive rates for 100% detection are about 63%, 65% and 85% for window size 5, 15 and 2 respectively. We also see that the change in the window size doesn't change the optimum detection rate and false positive rate.

6.2.3 Overall Findings

Table below shows results from applying SOM and Markov Model to intrusion detection systems with DARPA set 1998. For this the data set includes 1908 normal processes for training, 530 normal processes and 9 abnormal processes for testing. SOM could reach the maximum detection rate of 88.89% and minimum false positive rate of 5.57%. Whereas the Markov model can have maximum detection rate of 77.78% and false positive rate of 2.07%.

Table 6.2: Overall Results

Variations	Detection Rate	False Positive
SOM (25*25)	88.89%	11.59 %
SOM (50*50)	88.89%	6.60%
SOM (100*100)	88.89%	5.57%
Markov Model	77.78%	2.07%

The overall detection rate for SOM and Markov model combined was found to be 100%. The intrusion missed by SOM was detected by Markov model and that missed by Markov model was detected by SOM.

7 CONCLUSION

We developed an anomaly based intrusion detection system. We found out that by analyzing the frequency and transitional property of data we can effectively detect intrusions. SOM can be used to capture frequency property of data and Markov's Model can be used to capture the transitional property of data. And by combining the two modelling techniques we can fully model a data. The conclusion follows straightforward from the fact that the intrusions missed by the SOM are detected by the Markovian Model and the intrusions missed by the Markovian model are captured by the SOM, hence giving a 100% detection rate combined.

8 FUTURE ENHANCEMENT

- Our system is an offline intrusion detection system. So, our system can be extended to detect intrusion on the online data.
- We have detected the intrusion in the sequence of system call only, the other parameters of the system call like system call argument, return value; etc can also be used for anomaly based intrusion detection.
- Our model only captures frequency and transitional property of data and detects anomaly on that so the model can be extended to capture other properties of data.
- With anomaly base detection the detection rate is high with some amount of false positive where as on signature based detection the detection rate is low without the false alarm. So, the Hybrid system can be developed combining this two approaches to develop a model with high detection rate and almost zero false positive rate.
- Our system is an intrusion detection system not a prevention system. So, the system can be extended to intrusion detection and prevention system.

REFERENCES

[1] http://dictionary.reference.com/browse/intrusion

[2] Hybrid Intelligent Intrusion Detection System Norbik Bashah, Idris Bharanidharan Shanmugam, and Abdul Manan Ahmed

[3] Guide to Intrusion Detection and Prevention Systems (IDPS) – Karen Scarfone and Peter Mell

[4] Anomaly Detection : A Survey VARUN CHANDOLA, ARINDAM BANERJEE, VIPIN KUMAR

[5] Application of SVM and ANN for intrusion detection Wun-Hwa Chen, Sheng-Hsun Hsu, Hwang-Pin Shen

[6] Intrusion Detection System for Classifying Process Behavior -Nguyen Quang

[7] www.ll.mit.edu/mission/communications/ist/corpora/.../index.html

[8] A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems by Kristopher Kendall

[9] Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data5 Wei Wanga, Xiaohong Guana Xiangliang Zhangc, Liwei Yanga

[10] Host-based Intrusion Detection Systems Pieter de Boer & Martin Pels

[11] INTRUSION DETECTION SYSTEM USING SELF ORGANIZING MAP Liberios VOKOROKOS, Anton BALÁŽ, Martin CHOVANEC

[12] Dynamic Intrusion Detection Using Self-Organizing Maps Peter Lichodzijewski, A.Nur Zincir-Heywood, Malcolm I. Heywood

[13] Host-Based Intrusion Detection Using Self-Organizing Maps Peter Lichodzijewski, A. Nur Zincir-Heywood, *Member, IEEE,* Malcolm I. Heywood, *Member, IEEE* Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

[14] A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition LAWRENCE R. RABINER

[15] Efficient High-Order Hidden Markov Modelling by Johan A. du Preez

[16] Intrusion Detection: Host-Based and Network-Based Intrusion Detection Systems Harley Kozushko

[17] Log Analysis-Based Intrusion Detection via Unsupervised Learning Pingchuan Ma

[18] Data Mining for Network Intrusion Detection: How to Get Started Eric Bloedorn, Alan D. Christiansen, William Hill, Clement Skorupka, Lisa M. Talbot, Jonathan Tivel

[19] Intrusion Detection Using Datamining Techniques Anshu Veda, Prajakta Kalekar. Anirudha Bodhankar

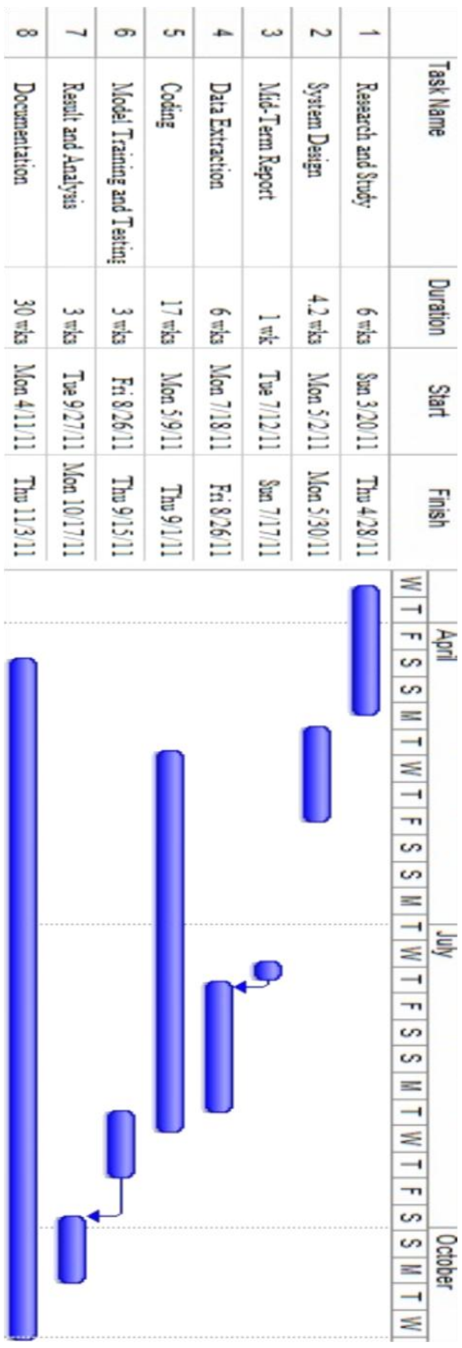
[20] Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance W.M.P. van der Aalst and A.K.A. de Medeiros

[21] ON THE LEARNING OF SYSTEM CALL ATTRIBUTES FOR HOST-BASED ANOMALY DETECTION GAURAV TANDON, PHILIP K. CHAN

[22] Detecting Intrusions Using System Calls: Alternative Data Models ChristinaWarrender Stephanie Forrest Barak Pearlmutter

[23] Introduction to Kohonen’s Self-Organizing Maps Fernando Bacao and Victor Lobo

APPENDIX A: GANTT CHART



	Task Name	Duration	Start	Finish
1	Research and Study	6 wks	Sun 3/20/11	Thu 4/28/11
2	System Design	4.2 wks	Mon 5/2/11	Mon 5/30/11
3	Mid-Term Report	1 wk	Tue 7/12/11	Sun 7/17/11
4	Data Extraction	6 wks	Mon 7/18/11	Fri 8/26/11
5	Coding	17 wks	Mon 5/9/11	Thu 9/1/11
6	Model Training and Testing	3 wks	Fri 8/26/11	Thu 9/15/11
7	Result and Analysis	3 wks	Tue 9/27/11	Mon 10/17/11
8	Documentation	30 wks	Mon 4/11/11	Thu 11/3/11

Figure A.1 Schedule

APPENDIX B: BSM FILE TOKENS

Table B.1 Audit Record Structure

Header token
Arg token
Data token
Subject token
Return token

Table B.2 File Token

Token ID	Date and Time	Name length	Previous/next file name
1 byte	8 byte	2 byte	N bytes

Table B.3: Header Token

Token ID	Byte count	Version#	Event ID	ID modifier	Data and time
1 byte	4 byte	1 byte	2 bytes	2 bytes	8 bytes

Table B.4: Text Token

Token ID	Text length	Text string
1 bytes	2 bytes	N bytes

Table B.5: Path Token

Token ID	Path length	Path
1 byte	2 bytes	N bytes

Table B.6: Attribute Token

Token ID	File mode	Owner UID	Owner GID	File system ID	File inode ID	Device ID
1 byte	4 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

Table B.7: Return Token

Token ID	Process error	Process value
1 byte	1 byte	4 bytes

Table B.8: Trailer Token

Token ID	Pad number	Byte count
1 byte	2 bytes	4 bytes

Table B.9: argument Token

Token ID	Argument#	Argument value	Text length	Text
1 byte	1 byte	4 bytes	2 bytes	N bytes

Table B.10: exec_args Token

Token ID	Count	Env_args
1 byte	4 bytes	Count null-terminated strings

Table B.11: ip Token

Token ID	IP header
1 byte	20 bytes

Table B.12: socket Token

Token ID	Socket type	Local port	Local Internet address	Remote port	Remote internet address
1 byte	2 bytes	2 bytes	4 bytes	2 bytes	4 bytes

Table B.13: process Token

Token ID	Audit ID	User ID	Group ID	Real user ID	Real group ID	Process ID	Session ID	Device ID	Machine ID
1 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes

APPENDIX C: TRAINING DATA

Training Data for SOM

Sample 1: 0 1 35 17 80 0 1 1 0 35 12 0 0 0 0 0 2 1 4 2 0 1 1 0 9 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 2: 0 0 2 5 9 0 0 0 0 9 3 0 0 0 0 0 2 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 3: 0 0 21 45 64 0 0 0 0 62 42 0 0 0 0 1 1 1 7 10 0 1 0 0 10 4 1 1 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 4: 0 0 3 1 15 0 0 1 0 0 0 0 1 0 0 0 0 0 3 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 5: 0 3 36 6 24 0 0 1 1 0 0 1 1 3 1 0 1 0 8 6 0 0 1 0 5 0 0 0 0 0 0 4 0 0 0 2 0 0 0 0 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 6: 0 3 27 4 27 0 0 0 1 0 0 1 1 3 0 0 0 0 8 3 0 0 1 0 5 1 0 0 0 0 0 4 1 0 1 2 0 1 0 0 3 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 7: 0 9 6 13 62 0 1 0 0 25 8 0 0 0 0 1 0 1 4 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sample 8: 0 0 14 16 38 0 0 0 1 28 10 0 0 2 0 0 1 1 64 0 4 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0
0 4 0 0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Training Data for Markov Model

Sample 1: 23 72 210 72 210 210 213 210 210 112 72 210 112 72 210 210 213 210 112 112 213 72 158 14 11 158 14 10 112 112 112 112 112 1

Sample 2: 23 72 210 72 210 210 213 210 112 72 210 210 213 210 112 72 210 210 213 210 112 72 210 210 213 210 210 112 72 210 210 213 210 112 72 210 210 213 210 210 112 72 210 112 72 210 210 213 210 112 72 210 210 213 210 112 72 210 210 213 210 112 72 210 210 213 210 112 112 213 80 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 72 72 158 112 80 158 158 158 158 158 158 239 72 112 158 72 158 112 72 239 72 72 158 72 22 16 72 158 112 112 8 14 16 72 72 30 30 16 16 72 158 112 77 30 77 158 158 72 16 112 16 16 80 80 14 158 112 112 72 112 80 158 158 158 158 158 158 158 158 158 112 239 72 158 112 80 158 158 158 158 158 158 216 217 112 80 158 158 158 158 158 158 216 217 112 80 158 158 158 158 158 158 216 217 112 79 30 158 158 158 112 76 216 72 30 72 72 158 158 158 158 112 158 80 30 112 112 112 6 112 2 200 112 112 112 112 112 112 112 112 112 1

Sample 3: 27 158 27 158 158 27 158 112 112 112 158 158 23 23 23 112 112 112 112 112 112 112 112 112 1

Sample 4: 133 72 158 112 26 26 205 72 158 112 26 200 72 30 112 112 4 30 112 30 112 112 213 27 27 158 2 158 27 158 112 112 112 1