Host Based Anomaly Detection Using Data Mining Techniques

A thesis Submitted for
the award of the degree of

Doctor of Philosophy

in

Computer Science

by

Subrat Kumar Dash

Department of Computer & Information Sciences
School of Mathematics and Computer & Information Sciences
University of Hyderabad
Hyderabad, India
July 2009

Department of Computer and Information Sciences
School of Mathematics and Computer & Information Sciences
University of Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that the thesis work entitled **"Host Based Anomaly Detection Using Data Mining Techniques"** being submitted by **Subrat Kumar Dash** in partial fulfillment of the requirement for the award of degree of Doctor of Philosophy (Computer Science) of the University of Hyderabad, is a record of bona fide work carried out by him under my supervision.

The matter embodied in this thesis has not been submitted for the award of any other research degree.

Prof. Arun K Pujari
Supervisor

Prof. Chakravarthy Bhagvati
Supervisor

Head
Department of Computer and
Information Sciences
University of Hyderabad, Hyderabad

Dean
School of Mathematics and
Computer & Information Sciences
University of Hyderabad, Hyderabad

Dedicated to my beloved parents

SELF-DECLARATION

I, Subrat Kumar Dash, hereby declare that the work presented in this thesis has been carried out by me under the guidance of Prof. Arun K Pujari and Prof. Chakravarthy Bhagvati, Department of Computer and Information Sciences, University of Hyderabad, as per the PhD ordinances of the university. I declare, to the best of my knowledge, that no part of this thesis has been submitted for the award of a research degree of any other university.

(Subrat Kumar Dash)

ACKNOWLEDGEMENTS

At the outset, I heartily thank my supervisors Prof. Arun K Pujari and Prof. Chakravarthy Bhagvati for supervising and patiently examining my PhD work.

My mentor Prof. Pujari has been a great inspiration for me to carry out my research work. I am extremely grateful to him for giving me a platform to see the world of research, for the ideas and suggestions he has offered, for all the long hours spend that often departed from the standard schedule and for providing me very good infrastructure and resources. I thank him for patiently analyzing my experiments and ideas and encouraging me to always go in a different way. I express my sincere thanks to Mrs. Pujari for all the help and support she has given in putting up with my schedules.

I sincerely thank Prof. Bhagvati for his unconditional support throughout my research program. I thank him for all the valuable suggestions and guidance that he has given me. In all, both my supervisors contributed a lot in shaping me as a researcher.

I would also like to acknowledge the other members of my committee, Dr. Atul Negi and Dr. Vineet Nair, who have offered their time and expertise to assist in the evaluation of my research work. I appreciate very much the support and encouragement of the department faculty and staff and making it an enjoyable one to do research in. I would also like to acknowledge the support and facilities provided by LNM Institute of Information Technology during my research tenure in that institute.

I learned a lot from the coauthors of my papers Prof. Vijayakumari, Dr. Sanjay Rawat, Krupa Sagar Reddy, Krishna Sandeep Reddy. I thank them all for their support and help. I acknowledge the help and support of Sagar and Sandeep during experimentation of my algorithms. My special thanks goes to Dr. Sanjay Rawat, with whom I have closely worked on some of my research topics. I can't forget those critical discussions that we had, which eventually led to better understanding of inner security concepts.

I am thankful to my colleagues and friends particularly Ahmed, Anil, Ja, Nagamani

Mam, Ravi and many more, who have been a continuous source of entertainment and have made my days memorable.

I am grateful to Dr. Sambhu Mohapatra for his time to time advice which has been a great source of inspiration for me.

My biggest source of motivation are undoubtedly my parents. They gave me incredible amount of support, courage and affection and are a constant source of inspiration for me. It is their guidance and sacrifice that made this possible. I dedicate this thesis to them. I thank my sister and brother-in-law for their support and care. The innocent voice of Abha, my cute ten months niece, made me forget my tensions during the days of writing of my thesis.

During my PhD studies, I was financially supported partly by Ministry of Information Technology, Govt. of India under the grant 12(22)/04-IRSD and Council of Scientific and Industrial Research (CSIR) and University of Hyderabad under the University with Potential for Excellence scheme. Their support is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Host based anomaly detection involves building profiles from normal behavior of a host entity and monitoring the entity for any deviations from these learned profiles. The entity can be a system, a user or a process. The present thesis investigates some specific problems which fall into host based anomaly detection category.

User profiles are built to monitor user level abnormality. This aspect leads to a specific problem called Masquerade attack, where one user impersonates another user. From the users' command line data, we observe that the usage by a user consists of a set of commands rather than isolated commands. Thus, it is more natural to have a detection mechanism based on meaningful sequence of commands, which we call "episodes". We provide a novel and efficient way of finding episodes from users' command line data and propose a masquerade detection technique based on these episodes.

We observe that a user may do multiple tasks simultaneously, and sometimes sequences generated by a user are nested. Thus, it leads to temporal interleaving of actions or tasks. In order to capture this behavior, we propose a novel use of constraint network. We propose to model a legitimate user as an IA-network which captures the binary relationship between patterns or episodes. An unknown user is modeled as a new set of IA relations which are verified for consistency with respect to the network of trained users.

We also observe that a user may temporarily deviate from his/her normal behavior due to short term requirements. Thus, in such cases classifying a user instantly may lead to an erroneous result. It is more appropriate to doubt a user who is consistently deviating from the normal signature rather than a user who deviates momentarily and returns to the normal pattern. We formalize this particular concept as *Deferred Detection* and propose a deferral model for masquerade detection.

We test our proposed masquerade detection algorithms on standard datasets and show that they perform better than many of the earlier methods.

A program can be profiled to monitor any process level abnormality in a system. Different methods are already proposed in the literature which use "system calls" data to detect program abnormality. We observe that the representation of the "systems calls" data are high dimensional in nature but can have a low dimensional embedding which cannot be captured by any linearity assumption. Thus, we propose the use of Locally Linear Embedding (LLE), a non-linear dimensionality reduction technique, to reduce the size of the input data derived from the sequence of system calls. We perform experiments on DARPA BSM audit data and show that the the performance of the proposed method in terms of accuracy is better than earlier methods.

A host system also suffers from the problem of malicious executables. To protect the host system, a HIDS needs to distinguish between benign and malicious classes of executables. We observe that most of the statistical techniques for detection of malicious executables use fixed-length n-grams as features and miss capturing meaningful n-gram sequences if they have a different length. We propose to use variable length n-grams as features and a new feature selection measure named class-wise episode frequency to efficiently discriminate virus and benign executables. We provide a novel way of extracting variable length n-grams based on the concept of episodes from a class of executables. Our comparative analysis shows that the proposed method based on variable length n-grams performs better than fixed-length approach.

CHAPTER I

AN INTRODUCTION: HOST BASED ANOMALY DETECTION

In this introductory chapter, we discuss Intrusion Detection System (IDS) in Section 1.1 and discuss different categories of IDS with a special emphasis on host based anomaly detection in Section 1.2. In Section 1.3, we describe the problem statement and the thesis contributions. Finally, Section 1.4 describes the organization of the thesis.

1.1 Intrusion Detection Systems

IDS has become an essential component of computer security in recent years. To understand an IDS, we need to define its basic unit - *intrusion*. An *intrusion* is an event or action that causes breach of *confidentiality* and/or *integrity* and/or *availability* of system or resources. Confidentiality is said to be breached, if resources residing in a computer are accessed in an unauthorized manner. Integrity is said to be breached, if the states of resources residing in a computer are changed in an unauthorized manner. And, availability is said to be breached if legitimate users are prohibited to access resources or services residing in a computer. *Intrusion detection* is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions. An *intrusion detection system* (IDS) is a software or hardware that automates the process of monitoring and analysis of such events [6].

An IDS generally consists of two main components, one is *Collection/storage unit* and the other is *Processing unit*. The collection/storage unit collects audit data from different sources based on the type of the IDS and process them to store in proper format. The processing unit is the brain of the IDS, where different detection algorithms are executed on the collected data to check for any intrusive behavior. Upon detecting any intrusive behavior, an alert is set off.

1.2 Classification of Intrusion Detection Systems

IDSs can be categorized into various classes based on their data collection or operational aspects (Network based vs. Host based) and processing or detection principles (Anomaly based vs. Misuse based).

1.2.1 Network based vs. Host based

Distinction between IDSs can be made based on the source of the data being audited, as *network* based and *host* based systems. Network based IDS (NIDS) is connected to a network being monitored and collects data from the network in a promiscuous mode (by acting as a *sniffer*) in the form of packets. NIDS needs to reconstruct the packet flow for every single host taking the implementation variability of the TCP/IP standards among different operating systems. NIDS can only detect attacks that come through the network and becomes ineffective if the network traffic is encrypted as the information such as protocols and contents can not be accessed in that case. But, on the other hand, this type of IDSs are OS-independent and thus, easy to deploy.

Host based IDS (HIDS) collects data from the host being monitored, in the form of OS log files, system calls, user generated commands, CPU utilization, event logs, application level logs etc. As HIDS works at system level, it is platform-dependent. HIDS can detect system level anomalies such as abnormal process, buffer overflow attack, misuse of account, masquerade attack etc.

1.2.2 Anomaly based vs. Misuse based

Data is collected by an IDS for further processing and classification. In the simplest case, it is a binary classification problem: normal (acceptable) or anomalous (possibly intrusive). IDSs can be categorized into two types depending on whether they look for abnormal behavior (anomaly based) or for known intrusion signatures (misuse based).

An anomaly based IDS, also known as behavior-based system, tries to create a model of normal behavior for the monitored entity, and flags as suspicious any deviation from this "normal" behavior without regard to actual intrusion scenarios. The assumption being, any

abnormal behavior or activity differs significantly from the normal behavior. But in reality, any deviation from normal activity may not be an intrusion, therefore, anomaly based IDS suffers from high false positive rate. On the other hand, they are capable of detecting new (zero day) and modified attacks as long as these attacks generate abnormal behaviors.

Misuse based IDS, also known as signature based system, maintains a database of signatures of known attacks, which means that instead of trying to describe the normal behavior of a system it tries to describe the anomalous behaviors. The collected audit data are matched with the signature database and an alarm is triggered if any match is found. Misuse based IDS fails to detect zero-day attacks as the signatures for such attacks are not available in the database. This type of IDS requires extensive study of the attacks in order to build and keep up to date the signature database and hence, the efficiency is directly dependent on the signature database. Misuse based IDSs are precise i.e., when an alarm is triggered, we can know which attack signature is activated and thus can know the specific attack. Most of the commercial IDSs belong to this category.

In other words, anomaly detection is about finding the normal usage patterns from the audit data, whereas misuse detection is about encoding and matching the intrusion patterns using the audit data [66].

The strengths and weaknesses of these two types of IDSs are summarized in Table 1.

Table 1: Comparative representation of strengths and weaknesses of anomaly and misuse based IDSs

Anomaly based	**Misuse based**
Long and complex training	No initial training
Do not require updates	Require continuous updates
Can detect new attacks	Cannot detect new attacks
Vague alerts	Precise alerts
High false positives	Almost no false positives

1.2.3 Host Based Anomaly Detection

Host based anomaly detection methods are part of HIDS which works on anomaly detection principles and broadly consists of two stages. The first stage deals with building profiles of normal behavior of a monitored entity, which can be a system or a process or a user (whose

behavior can be captured from a host system), and hence is analogous to the *training phase* of a learning system. In the second stage, these profiles are used to monitor the behavior of the said entity for any significant deviations from the learned normal profile, which is analogous to the *test phase*.

As host based anomaly detection involves *learning* of behavior and profiling them, different techniques from Machine Learning, Data Mining and Artificial Intelligence gets application here.

1.3 Problem Statement and Contribution

This thesis investigates some specific problems which eventually falls into host based anomaly detection category.

1.3.1 Problem Statement

With respect to host based anomaly detection, there are many features to model a normal behavior. For example, user command data and usage patterns, keystroke characteristics, system call sequences, file activities etc. These features can be broadly classified into two main categories based on the entity to be monitored, such as *user profile* and *program profile*.

User profile can be build to monitor abnormality in user behavior. This aspect leads to a very specific problem in computer security known as *Masquerade Attack* [99], where an illegitimate user, called the masquerader, poses as (and assumes the identity of) a legitimate user. In other words, it is a kind of spoofing attack, but the masquerader appears to be a normal user with valid authority and privileges. The broad range of damage that can be performed via the masquerade attack makes this, one of the most serious threats to computer and network infrastructure. The detection of such attack largely depends on extracting correct features and a proper modeling of user behavior. Therefore, in this work, we investigate on *how to extract correct features and model user behavior leading to an efficient masquerade detection.*

In a similar line, program can be profiled to monitor for any process level abnormality in a system. Different techniques exist in literature for abnormal process detection which uses

system calls as basic features. Most of these techniques represent system calls data in either of the two ways: *term frequency* - which is based on frequency of system calls and *decision table* - which is based on subsequences of system calls. But, both these classes of techniques lead to a high dimensional representation of the data. We, in this work, investigate *how to reduce the dimension of the input data without degrading the accuracy of the detection system.*

Another major problem faced by a host system is from malicious executables such as viruses, worms, trojan horses, back doors and spyware. To protect the host system, an HIDS needs to distinguish between benign class and malicious class of executables, which essentially is a classification problem. Any classification technique is as good as the features extracted and the feature selection method employed. Such extracted and selected features are assumed to carry characteristics of different classes and thus, can increase the accuracy of the classifier. This thesis investigates *how to extract and select proper features for efficient classification of benign and malicious executables.*

In a nutshell, in this thesis, we try to investigate the following problems related to host based anomaly detection.

- How to extract correct features and model user behavior leading to an efficient masquerade detection.
- How to reduce the dimension of the input data without degrading the accuracy of the detection system.
- How to extract and select proper features for efficient classification of benign and malicious executables.

1.3.2 Thesis Contributions

The work presented in this thesis mainly contributes to the area of host based anomaly detection, but can also be used for other applications related to feature extraction, behavior modeling and classification. The main contributions of our thesis are as follows:

1. Episode based Masquerade Detection: It is assumed that individual commands

play a very important role in determining legitimate or questionable behavior of a user. We observe that the usage by a user consists of a set of commands for a particular task, rather than isolated commands. Thus, it is more natural to have a masquerade detection technique based on *episodes*, meaningful subsequences of commands that may define a single task or purpose, rather than with individual commands. We, therefore, provide a novel and efficient way of finding *episodes* from users' command line data. Empirically we show that masquerade detection based on episodes performs better than most of the earlier masquerade detection techniques.

2. **Constraint Programming for User Modeling and Masquerade Detection:** We observe that a user may use multiple tasks simultaneously, and sometimes sequences generated by a user are nested, for example a script calling another script. Thus, it leads to temporal interleaving of actions or tasks. In order to capture this behavior, we propose a novel use of *constraint network*. We propose to model a legitimate user as a IA-network which captures the binary relationship between patterns or episodes. An unknown user is modeled as a new set of IA relations which are verified for consistency with respect to the network of trained users.

3. **Adaptive Naive Bayes Method for Deferred Masquerade Detection:** A user may deviate from his normal behavior for a short duration and thus classifying a user instantly may leads to an increase in the false positive rate resulting in erroneous classification. We use a deferral detection model which categorize a block of commands into either *legitimate*, *doubtful* or *masquerade* category. We show empirical results to validate our observations.

4. **Abnormal Process Detection in Low Dimensional Space:** We observe that not all of the system calls generated by a process are important to study the behavior of the process. Also, the representation of the generated system calls are high-dimensional in nature but, can have a low dimensional embedding which can not be captured by the linearity assumption. Thus, we propose the use of *Locally Linear Embedding (LLE)*, a non-linear dimensionality reduction technique, to reduce the size of the input

data derived from the sequence of system calls. We provide direct empirical proof for our observation. We also provide a way of projecting system calls data, which are *categorical* in nature, to a low dimensional Euclidean space.

5. **New Malicious Code Detection Using Variable Length n-grams:** Most of the statistical techniques for malicious code detection use fixed length n-grams [60] as features, but we observe that they miss to capture meaningful n-gram sequences of different lengths. We propose to use *variable length n-grams* as features and a new feature selection measure named *class-wise episode frequency* to efficiently discriminate virus and benign executables. We provide a novel way of extracting variable length n-grams, based on the concept of *episodes*, from a class of executables. Our study also provides empirical evidence that choice of proper feature extraction and feature selection technique is of paramount importance for classification.

1.4 Outline of the Thesis

The thesis is organized as follows:

Chapter I: The first chapter "*Introduction: Host Based Anomaly Detection*", provides a general description of Intrusion Detection and looks into the domain of Host Based Anomaly Detection. We discuss the problem statement and the main contributions of our work. Finally, we outline the organization of the thesis.

Chapter II: In this chapter entitled *"Masquerade Detection"*, we concentrate on a specific problem of host based anomaly detection called "Masquerade Attack". We define the problem of Masquerade Detection as is understood in information security research and provide an extensive study of the existing techniques and methods available in the literature. We provide a brief introduction on available datasets which are considered to be standard for this problem. We also describe our proposed methods for the detection of masquerade attack. We propose three such methods based on "Episode Mining", "Constraint Programming" and "Adaptive Naive Bayes" techniques.

Our proposed *Episode Mining* method provides a novel and efficient way of finding meaningful and frequent patterns from users' behavioral data, which we term as *episodes*. The method extracts episodes from a user's command line data and classify each episode as normal or abnormal using Naive Bayes classifier. Based on the overall strength of the abnormal episodes in the user's data, it is detected as normal or masquerade.

In the method based on *Constraint Programming*, we propose a novel way of modeling user behavior and the detection of masqueraders. We model a user as a binary constraint network such that each node represents an episode of commands and binary relationship between a pair of episodes is encoded as the disjunction of the Allen's Interval relations [1]. We use *path consistency* check by using the *Qualitative-Path-Consistency algorithm* [31] to decide whether a test block belongs to a legitimate user or is a masquerade block.

The motivation of our *Adaptive Naive Bayes* method is that a user may deviate from his normal behavior for a short duration. Thus, instead of classifying a user instantly, this method uses a deferral model in identifying the masqueraders. We propose to identify a block of commands into either *legitimate*, *doubtful* or *masquerade* category and discuss how to change the label of a block with different adaptive test conditions.

We provide experimental results for each of the proposed methods on standard datasets. The results are compared with relevant methods proposed in the literature.

Chapter III: In this chapter entitled "Application of Locally Linear Embedding to Abnormal Process Detection Using System Calls", we provide the details of our work on the usability of *Locally Linear Embedding (LLE)* [96], a non-linear dimensionality reduction technique, to reduce the dimensionality of the data. We describe two different ways of representing a process, one is *term frequency* representation, which is based on frequency of system calls and the other is *decision table* representation, which is based on subsequences of system calls and provide a brief survey of literatures based on these two types of representations. Processes are represented as vectors in

both these ways and LLE is applied to project these vectors into a lower *numeric* dimension. In the reduced dimension we apply k-NN to further classify the processes. We provide empirical results to justify the use of LLE to reduce the dimension of the data. We show that this reduction does not degrade the accuracy of the classifier, by performing experiments on DARPA BSM dataset [28].

Chapter IV: In this chapter entitled "*New Malicious Code Detection Using Variable Length n-grams*", we concentrate on new malicious code detection especially computer viruses. We propose a statistical method for identifying new malicious code by extracting common features of virus and benign programs as variable length n-grams, based on the concept of *episodes*. We describe different statistical techniques used to detect malicious executables available in the literature followed by our observations and motivation for the current work. We introduce a new feature selection measure, namely *class-wise episode frequency* and provide experimental results on our own collected dataset. We provide comparative analysis of the performance of fixed length n-grams versus variable length n-grams as features for virus detection.

Chapter V: We summarize our work presented in the thesis in this chapter entitled "*Conclusions*". We also discuss some possible future directions to which the work presented in this thesis points to.

CHAPTER II

MASQUERADE DETECTION

2.1 Introduction

Denning [32] divides intrusions into eight basic categories: Eavesdropping and packet sniffing; Snooping and downloading; Tampering or data diddling (unauthorized changes to data or records); Spoofing (impersonating other users); Jamming or flooding (overwhelming a system's resources); Injecting malicious code such as viruses and Trojan horses; Exploiting design or implementation flaws (e.g., buffer overflows); Cracking passwords and keys. This chapter focuses on a common form of spoofing, *masquerading*. In a masquerade attack, an illegitimate entity poses as (and assumes the identity of) a legitimate entity. The illegitimate user, called the masquerader, hides his/her identity by impersonating a legitimate user in a computer system or network. A masquerader can either be an insider or an outsider, who generally tries to gain access to the account of the super-user with malicious intent. Masquerade attack can occur in a variety of ways, such as by obtaining a legitimate user's password, accessing an unattended and unlocked workstation, forging email address in messages and overtaking a computer via a network access. It is difficult to detect this type of security breach at its initiation, because the attacker appears to be a normal user with valid authority and privileges.

Masquerading can be a serious threat to the security of computer systems and computational infrastructures. The broad range of damage that can be performed via the masquerade attack makes it one of the most serious threats to computer and network infrastructure. The 1996 Ernest & Young Information Security Survey [115] reports that, of companies who suffered financial losses from problems with information security and disaster recovery, a third cited malicious acts by company insiders, as the security violations. The annual computer crime and security survey [87] shows that attacks detected stemming

from “insider abuse of net access” and “unauthorized access by insiders” are on the rise. A well-known instance of masquerader activity is the case of Robert Hanssen [72] of the FBI who allegedly used agency computers to ferret out information which was later sold. Hanssen was a regular user, but his behaviour was improper. Thus, besides protecting from external intruders, there is a strong need for monitoring authorized users for anomalous behaviours. The difficulty lies in distinguishing anomalous behaviour from the natural change in user behaviour (also called *concept drift*). While adapting to user behaviour, concept drift is a desirable trait in an intrusion detection system (IDS). But, increased adaptability can help abusive actions by valid users to escape from the IDS. Detecting masquerade attack has long been a great challenge in information security research. Traditional approaches to masquerade detection involve creating a behaviour profile for each user. A masquerader can then be identified by detecting a significant change in the behaviour of the respective user.

Though the term ‘masquerade’ means that any unauthorized user poses as a legitimate user, the problem of masquerade detection in information security research does not take the general meaning of the term ‘masquerade’ but is referred to a very specific intrusion detection problem. In the following sections, we formally describe the masquerade detection problem as it is understood in computer security research.

The rest of the chapter is organized as follows. In Section 2.2, we define/formulate the masquerade detection problem. Existing methods available in the literature are covered in Section 2.3. Section 2.4 describes datasets that are considered to be standard for the study of command line based attacks. In Section 2.5, we describe about episode mining and discuss in length the episode based masquerade detection method with experimental results. The focus of Section 2.6 is a detection technique based on Interval Algebra (IA) network. Adaptive Naive Bayes method with experiments and discussion is covered in Section 2.7. Finally, Section 2.8 concludes the chapter.

2.2 Problem Formulation

Let us assume that a set of users $u_1, u_2, \ldots, u_U$, with their respective usage patterns (training sets) TR^1, TR^2, ..., TR^U are known. Given a sequence of commands $b = c_1, c_2 \ldots c_T$, assumed to be generated by a specific user u, the problem is to check whether b conforms to the known usage pattern of u, TR^u. The sequence b might have been genuinely generated by u or by a masquerader who is pretending to be u. If b conforms to TR^u then b is considered to be genuinely of u. Else, b is used by a masquerader and hence a masquerader is detected.

Jha et al. [53] formulate the masquerade detection problem in a slightly different manner. In this case, it is not known whether b is generated by a specific user u. Thus, the problem is to determine whether there is a user u, $1 \leq u \leq U$ who has generated the sequence b legitimately or, it is generated by a masquerader.

In this thesis, we adopt the first formulation of masquerade detection problem. This problem can be visualized as an anomaly detection problem where the task is to find whether a new trace from a user conforms with the past normal trace from the same user. But, most of the methods discussed in the next section can be applied for both the formulations of masquerade detection.

2.2.1 Generic Model

We describe here a *generic model* for masquerade detection problem. Given a set of users $u_1, u_2, \ldots, u_U$ and their corresponding training sets $TR^1, TR^2, \ldots, TR^U$, a profile for each user is prepared which captures the inherent characteristics of each user from its training data. Upon receiving a new block of commands b of a user u, the characteristics of b are compared with the known profile of u. Based on the extent of match and mismatch, b is classified as a normal or masquerade block.

2.3 Existing Methods

The earliest mention of masquerade detection in literature dates back to 1988 when Smaha et al. [106] list masquerade as one of the six attacks that his prototype intrusion detection system HAYSTACK planned to detect. HAYSTACK profiles each user in two ways: as an

individual and as part of a class of users. The hypothesis is that a single user may be able to train the individual profile to accept abusive behavior (hostile training), but will not be able to distort the entire class profile. HAYSTACK's class profile represents "generic notions of acceptable behavior for a group of users".

Lane and Brodley [64] present an approach to masquerade detection based on IBL (Instance Based Learning) and Hidden Markov Models (HMM) techniques. Schonlau et al. [99] study six different methods for masquerade detection and subsequently many other attempts employing different techniques such as Naive Bayes [75], Support Vector Machine (SVM) [59, 103, 119, 126], Non-negative Matrix Factorization (NMF) [121], Eigen co-occurrence Matrix (ECM) [83, 84], Bioinformatics [25, 26] are proposed. We categorize these methods into different classes as follows.

1. Frequency based methods: These groups of methods are based on frequency of individual commands.

2. Transition based methods: The methods in this category are based on, given a sequence of commands $c_1, c_2 \ldots, c_k$, what is the likelihood of transition to the next command c_{k+1}.

3. Sequence based methods: This class of methods deals with a sequence or a subsequence of commands as a whole.

4. Feature Vector based methods: This set of methods works with suitable features extracted from the given sequence.

Based on the above classes, the categorization of different masquerade detection methods are shown in Table 2. In order to describe these methods we introduce the following notations.

Let TR be the training data of all users; TR^u be the training data of user u; U be the total number of users; U_k be the number of users who have used command k in the training data. N_u is defined as the length of TR^u and N_{uk} is the number of times user u used the command k in the training data. Let b be a new observed command sequence $c_1, c_2, \ldots, c_T$

Table 2: Categorization of masquerade detection methods

Frequency based	Uniqueness, Naive Bayes
Transition based	Bayes One-step Markov, Hybrid Multistep Markov, Filtering Approach, Sequence Prediction, Compression
Sequence based	Sequence Match Sequence Alignment
Feature Vector based	Non-negative matrix factorization (NMF), Eigen co-occurrence matrix (ECM), Support vector machine (SVM) Weighted RBF Naive Bayes

Table 3: Definition of notations

Notation	Meaning
TR	Training data of all users
TR^u	Training data of user u
U	Total number of users
U_k	Number of users who have used command k in TR
N_u	Length of TR^u
N_{uk}	Number of times user u used the command k in TR^u
b	New observed command sequence $c_1, c_2, \ldots, c_T$ for a user u
T	Length of b
n_{uk}	Number of times user u used the command k in b
K	Number of distinct commands in TR and b together

for a user u (length of b is T); n_{uk} be the number of times user u used the command k in b; K be the number of distinct commands in TR and b together. Table 3 lists these definitions.

2.3.1 Frequency Based Methods

Frequency based methods are based on frequency of individual commands. These methods ignore the temporal order information of the sequence of commands and consider only the number of times a command appears in a sequence or subsequence thereof. There are many intuitions (or assumptions) on which these methods are developed. For instance, the more frequent a command is in TR^u the more indicative it is of a pattern of u. Similarly, by

another intuition, if a command is more often used by a user u than by any other users then it is stronger indicative of behaviour of u. In this category there are two known algorithms: Uniqueness and Naive Bayes methods.

2.3.1.1 Uniqueness

Uniqueness approach reported by Schonlau et al. [99, 100] is based on the concepts of *unique* and *unpopular* commands. A command is said to be *unique*, if only one user uses that command, and is *unpopular*, if it is used only by few users. The uniqueness method is based on the intuition that commands not previously seen in the training data, TR, is a good indicator of anomaly in b and the more an unpopular command is used in b the more indicative it is of intrusion. Schonlau et al. [99] define a *uniqueness* measure that encapsulates the degree of uniqueness as well as (un)popularity of a command k for user u. This measure for a command k for user u is defined as

$$W_{uk}(1 - \frac{U_k}{U})n_{uk}. \tag{1}$$

where,

$$\begin{aligned} W_{uk} &= \begin{cases} -v_{uk}/v_k, \text{ if } TR^u \text{ contains command } k \\ 1, \text{ otherwise} \end{cases} \\ v_{uk} &= \frac{N_{uk}}{N_u} \\ v_k &= \sum_u v_{uk} \end{aligned}$$

In other words, v_{uk} is the probability of command k in TR^u. In Equation 1, $(1 - U_k/U)$ acts as a uniqueness index; which is 0, if all users use command k, and 1, if none of the users uses this command during training. The weight parameter W_{uk} is 1, if command k is not used by user u in the training data; -1, if it is a command used only by user u; and between -1 and 1, if it is used by user u and also by other users. If k is not in b, the uniqueness measure of k for u is 0; if k is in b and k is not in training set, then it is n_{uk}; if k is in b as well as in training set then the uniqueness measure is between 0 and $-n_{uk}$. The uniqueness measure of b for user u is the sum of uniqueness measures of the individual commands in b

and is defined as follows.

$$unique(b,u) = \frac{1}{T}\sum_{k\in b} W_{uk}(1-\frac{U_k}{U})n_{uk}. \tag{2}$$

Thus, the uniqueness measure for a user u is low if the user uses commands similar to the ones used in the training data. The uniqueness score, $unique(b,u)$, is compared with a threshold value τ to check for abnormality.

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } unique(b,u) \geq \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

Although this method is based on a simple intuition, it yields good results compared to other more complex detectors on the same set of data. This implies that frequency property of the command sequence is an important indicator of intrusion. Recently, in [9], the uniqueness measure is used in a slightly different way and based on the uniqueness measure of b and TR^u, a uniqueness diversity measure, $unique\_div(b)$, is computed. $unique\_div(b)$ measures the deviation of uniqueness between an observed sequence and the training sequence of the same user u.

$$\begin{aligned} unique(b)' &= \frac{1}{T}\sum_{k=1}^{K}(1+W_{uk})(1-\frac{U_k}{U})n_{uk} &(3)\\ &= \sum_{k=1}^{K}(1+W_{uk})(1-\frac{U_k}{U})\frac{n_{uk}}{T} &(4)\end{aligned}$$

For the training data of a user u, the uniqueness measure can be defined as,

$$unique(TR^u)' = \sum_{k=1}^{K}(1+W_{uk})(1-\frac{U_k}{U})\frac{N_{uk}}{N_u} \tag{5}$$

Thus, the uniqueness diversity measure is,

$$unique\_div(b) = unique(b)' - unique(TR^u)' = \sum_{k=1}^{K}(1+W_{uk})(1-\frac{U_k}{U})\left|\frac{n_{uk}}{T} - \frac{N_{uk}}{N_u}\right| \tag{6}$$

Based on $unique\_div(b)$, b is classified as follows,

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } unique\_div(b) \geq \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

2.3.1.2 Naive Bayes

The Naive Bayes method is proposed by Maxion and Townsend [75]. This method assumes that the commands in b are independent of each other i.e., the occurrence of the next command in b is independent of the previous commands. Based on this independent assumption, it calculates the self and non-self probabilities of b. The self probability of b for a user u, $self(b,u)$, is the probability that b is generated by user u, and the non-self probability of b for user u, $nonself(b,u)$, is the probability that b is generated by some user other than u. We define $U_{\neq u}$ as the set of all users other than u.

$$self(b,u) = \prod_{i=1}^{T} P(c_i, u). \tag{7}$$

$$nonself(b,u) = \prod_{i=1}^{T} P(c_i, U_{\neq u}). \tag{8}$$

where,

$$P(c_i, u) = \frac{N_{uc_i} + \alpha}{N_u + (\alpha \times A^u)}.$$

and

$$P(c_i, U_{\neq u}) = \frac{\sum_{u_j \in U \wedge u_j \neq u} N_{u_j c_i}}{\sum_{u_j \in U \wedge u_j \neq u} N_{u_j}}.$$

A^u is the number of distinct commands in TR^u and α is a pseudocount, a real number larger than zero, and is added to ensure that there is no zero counts. The lower the pseudocount, the more sensitive the detector is to previously unseen commands. Maxion and Townsend take α to be 0.01 in their study. The larger the ratio of the self probability to the non-self probability, the greater the evidence in favour of b belonging to u. The ratio of the self probability to the non-self probability is compared with a threshold value τ to determine the abnormality of b.

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } \frac{self(b,u)}{nonself(b,u)} \leq \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

2.3.2 Transition Based Methods

In transition based methods, given a sequence of commands $c_1, c_2, \ldots, c_k$, the aim is to estimate the likelihood of the next command c_{k+1}. The occurrence of c_{k+1} is estimated

from the preceding k commands in the sequence where k can vary from 1 to the total number of commands in the sequence. In the following sections we outline methods which are based on this fundamental principle.

2.3.2.1 Bayes One-Step Markov

Bayes one-step markov is a method that attempts to study the frequency of one-step transition from one command to the next. Let P_{ujk} be the probability of next command being k when the current command is j for the user u. Using Bayesian logic, the probabilities P_{ujk} are computed from the conditional frequencies in the training data. It is to see if b is consistent with the transition probabilities generated by the training data.

This approach, proposed by DuMouchel [34], employs Bayes Factor (BF) for comparing two hypotheses, namely the null hypothesis and the alternative hypothesis. For a given b, the null hypothesis H_0 assumes that the transition probabilities of commands in b are in accordance with the transition in TR^u. H_0 also called as the historical transition probability is defined as

$$H_0 : P(k|j) = P_{ujk}$$

The alternative hypothesis H_1, on the other hand, is the transition probabilities of commands generated by a particular distribution, namely Dirichlet distribution.

$$H_1 : P(k|j) = Q_k$$

$(Q_1, \ldots, Q_K)$ are obtained from Dirichlet $(\alpha_{01}, \ldots, \alpha_{0K})$ [34]. Given the hypotheses H_0, H_1 and b as $c_1, c_2, \ldots, c_T$, BF is the ratio of marginal likelihood as given below,

$$BF = \frac{P(c_1, c_2, \ldots, c_T|H_1)}{P(c_1, c_2, \ldots, c_T|H_0)}. \tag{9}$$

It is argued in [99] that, BF on the log scale exhibits very nice property that evidence from two independent data sets is the sum of the individual evidence. Hence, it is used as the test statistics as $log(BF)$. It turns out that BF can be calculated as

$$BF = \frac{\prod_k (\alpha_{0k}(\alpha_{0k}+1)\ldots(\alpha_{0k}+n_{uk}-1))}{\alpha_0(\alpha_0+1)\ldots(\alpha_0+T-1)\prod_{j,k} P_{ujk}^{n_{ujk}}}. \tag{10}$$

where, α_{0k} and α_a are the Dirichlet distribution parameters such that $\alpha_0 = \sum \alpha_{0k}$, and n_{ujk} is the number of times command k follows command j in b. Given b we calculate the BF value and compare it with a threshold value τ to check for abnormality.

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } BF \leq \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

2.3.2.2 Hybrid Multistep Markov

Schonlau et al. [99] use the Hybrid Multistep Markov method, proposed by Ju et al. [56], for masquerade detection. It is a combination of two models namely multistep Markov model and an independence model.

The Multistep Markov Model: For a user u, $K_u - 1$ commands are selected such that it constitute at least 99% of TR^u. All other commands including those not appearing in TR^u form a category $other_u$. These $K_u - 1$ commands and $other_u$ constitute the Markov chain's state space M_u. According to the mixture transition distribution (MTD) model [88], given a command sequence $\{C_{ut}; t = 1, 2, \ldots, \}$ of user u, the transition probabilities of an l-step Markov chain is,

$$P(C_{ut} = c_0 | C_{ut-1} = c_1, C_{ut-2} = c_2, \ldots, C_{ut-l} = c_l) = \sum_{i=1}^{l} \lambda_{ui} r_u(c_0|c_i). \tag{11}$$

where, $R_u = \{r_u(i|j); i, j \in M_u\}$ and $\Lambda_u = \{\lambda_{ui}; i = 1, 2 \ldots, l\}$ satisfy certain positivity constraints. The log-likelihood of a command sequence for u is

$$\log L = \sum_{i_0 \in M_u} \cdots \sum_{i_l \in M_u} n(i_0, i_1, \ldots, i_l) \times \log(\sum_{j=1}^{l} \lambda_{uj} r_u(i_0|i_j)). \tag{12}$$

where, $n(i_0, i_1, \ldots, i_l)$ is the number of times the pattern $(i_0, i_1, \ldots, i_l)$ is observed in the command sequence.

The independence model: It assumes that the commands are independently generated from a multinomial random distribution. Thus,

$$P(C_{u1} = c_1, \ldots, C_{uT} = c_T | TR^u) = \prod_{t=1}^{T} P(C_{ut} = c_t | TR^u) = \prod_{t=1}^{T} q_{uc_t}. \tag{13}$$

where,

$$q_{uk} = \begin{cases} \frac{N'_{uk}}{N_u} & \text{if } N'_{uk} > 0 \\ \varepsilon & \text{otherwise} \end{cases} \tag{14}$$

N'_{uk} is a normalised representation of N_{uk}, which is computed as,

$$N'_{uk} = w_k N_{uk} + (1 - w_k)(\sum_u N_{uk} \frac{N_u}{\sum_u N_u}). \tag{15}$$

where, $w_k = 1 + 1/U - U_k/U$. This model assumes that unpopular commands have a greater power to distinguish different users than popular commands.

Combining the two models: The log-likelihood-ratio statistics for both the Markov and independence models are defined respectively as,

$$X_{1u} = \log \frac{\max_{v \neq u} L(c_1, \ldots, c_T | \hat{\Lambda_v}, \hat{R_v})}{L(c_1, \ldots, c_T | \hat{\Lambda_u}, \hat{R_u})} \tag{16}$$

$$X_{2u} = \log \frac{\max_{v \neq u} \prod_i q_{vc_i}}{\prod_i q_{uc_i}} \tag{17}$$

To bring both X_1 and X_2 into the same scale, X_1 is regressed on X_2 with no intercept based on least median of squares, thus satisfying $X_1 = \hat{\rho} X_2$. The hybrid model combines both the Markov and independence models such that it toggles between the two models based on the number of commands categorised as $other_u$ in the test data. Let s_u be the number of commands in b that are categorised as $other_u$. The hybrid model computes a score x_u as,

$$x_u = \begin{cases} X_{1u} & \text{if } s_u/T \leq \xi \\ \hat{\rho} X_{2u} & \text{otherwise} \end{cases} \tag{18}$$

b is classified based on the score x_u as follows,

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } x_u \leq \mu + 3\sigma \\ \text{normal}, & \text{otherwise} \end{cases}$$

where, μ and σ are the mean and standard deviation of the scores x_u estimated from TR.

2.3.2.3 Filtering Approach

A process X in discrete time $n = 0, 1, ...$, is simply a sequence of random variables $\{X_n\}_{n \geq 0}$, where X_n denotes the value at time n. The distribution of X_n depends on the values of the process at time $0, 1, \ldots, n-1$, and can be expressed as

$$p_X(c_n | c_{n-1}, \ldots, c_0). \tag{19}$$

If $p_X(c_n|c_{n-1})$ depends only on c_{n-1} and not on $c_0, \ldots, c_{n-2}$, the process is a *markov chain* and $p_X(c'|c)$ is its *transition function*. Jha et al. [53] propose a filtering approach for anomaly detection based on Markov chain.

The problem of anomaly detection is formulated as a mixture of two Markov chains, M_u and M_A. Where, M_u is a model of user u's normal profile, obtained from TR^u and M_A is a model of attack set. For simplicity, it is assumed that M_A is a chaotic process such that at each time there is an equal probability of generating each command.

Given an observation $b = c_1, c_2, \ldots, c_T$, the goal of anomaly detection is to determine whether b conforms to the model M_u. Filtering approach method assumes that b is a mixture of two models M_u and M_A with a random mixing rate α,

$$b = \alpha M_u + (1 - \alpha) M_A. \tag{20}$$

Based on this formulation, the conditional distribution of α, which signifies the strength of the normal set in b, is computed and if it goes below a specified threshold, an alarm is raised. The classification can be formulated as,

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } \alpha < \tau \\ \text{normal}, & \text{otherwise.} \end{cases}$$

2.3.2.4 Sequence Prediction

Sequence prediction algorithms predict the next command in an input sequence based on the past observations. Let $\sum$ be the alphabet of input commands. Let us assume that $b = c_1, \ldots, c_T$, $c_j \in \sum$ for $1 \leq j \leq T$, be a sequence of input commands of which the first i commands, $c_1, \ldots, c_i$ are known/observed. Based on the already observed i commands, a sequence prediction algorithm returns the probability for each command $k \in \sum$ that k is the next command in the input sequence. In other words, it returns the probability value $P(k|c_1, \ldots, c_i)$ for all $k \in \sum$.

One of the first algorithms for predicting the next user action is Incremental Probabilistic Action Modeling (IPAM) [29]. IPAM employs a first order Markov model, i.e. it is based on one-step command transition probabilities, just as Bayes one-Step Markov, estimated

from the training data. But, the transition probabilities are updated dynamically as and when new user action is observed. Consider that we have arrived at the current state via command c_i. Then, upon arrival of the new command, c_{i+1}, the probabilities associated with the transition from c_i to k, $k \in \sum$ are aged by multiplying them with α, $0 \leq \alpha \leq 1$ and $(1 - \alpha)$ is added to the most recent transition, i.e. from c_i to c_{i+1}. This procedure can be described in a recursive formula as shown in the following equation:

$$P_{ipam}(k|c_i) = \begin{cases} \alpha\ P_{ipam}(k|c_i) + (1 - \alpha) & \text{if } k = c_{i+1} \\ \alpha\ P_{ipam}(k|c_i) & \text{otherwise} \end{cases}$$

This approach for updating the probabilities of the transitions is called *aging*, since as time goes by the probabilities are geared towards transitions of frequent occurrence. According to some empirical results, Davison et al. [29] recommend a value of 0.8 for α. Thus, the recent commands have a greater impact on the predictions than the older commands. By assigning $\alpha = 0$, probabilities for all transitions, but the recent transition, are made zero which predicts what is seen most recently. Whereas, $\alpha = 1$ leads to no change in the transition probabilities, and thus the initial transition probabilities are carried forward.

Schonlau et al. [99] study IPAM for masquerade detection. Given a command, IPAM can predict the next command by choosing the one corresponding to the highest transition probability. To determine if an input sequence, b, is an attack, IPAM considers every command transition in b, one at a time. For any command, the transition is marked as *good* if the next command is one among the top four predicted commands. If the ratio of *good* versus the length of b falls below a threshold an alarm is raised.

2.3.2.5 Compression

The compression based classification has been popular recently in sequence data analysis such as bioinformatics. This technique originates from Normalized Information Distance (NID) which is considered to be a universal similarity metric based on Kolmogorov complexity [68]. It is well-known that NID is incomputable [68] and compression based distances are the computable simplification of NID. The similarity between two strings A and B is inversely proportional to the additional bytes required to compress A and B separately from

compressing A and B together (concatenating). Another way of looking at compression based method is for a sequence of commands A, which is compressed by a given compressor into a file of size α, if we add noise to this sequence and compress it, as the amount of noise increases, the compressor is less able to reduce the file size. In other words, a noisy sequence results in more number of bytes after compression than noise free sequence. So, in the present context, new data from a user compresses at about the same ratio as old data from that same user, and that data from a masquerading user will compress at a different ratio and thereby can be distinguished from the legitimate user. In [99], a compression score, $compress(b)$, is defined as the number of additional bytes needed to compress an observed sequence b, when appended to TR^u.

$$compress(b) = |compress(TR^u b)| - |compress(TR^u)| \tag{21}$$

where, $TR^u b$ is TR^u concatenated with b, and $|compress(x)|$ is the number of bytes of compressed data (after compressing x). The compression score, $compress(b)$ is compared with a threshold, τ, to check for the abnormality of b.

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } compress(b) \geq \tau \\ \text{normal}, & \text{otherwise.} \end{cases}$$

The results of the compression based method is not very encouraging [99]. However, in [13, 12], Bettacchini et al. observe that by using Normalized Compression Distance (NCD) the efficiency of compression method can be improved. The NCD is defined as,

$$NCD(b, b') = \frac{|compress(bb')| - \min\left(|compress(b)|, |compress(b')|\right)}{\max\left(|compress(b)|, |compress(b')|\right)}. \tag{22}$$

NCD is a normalized distance in the range $[0, 1]$, such that $NCD(b, b')$ tends to 0 when b and b' are maximally similar and $NCD(b, b')$ tends to 1 when b and b' are maximally dissimilar. The classification of b is defined as follows

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } NCD(b, TR^u) \geq \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

In [13], the authors use compression algorithms such as, `bzip`, `compress`, `zlib`, raw `zlib` with a custom dictionary with truncated command-line data and in another paper [12], they

also use Prediction by Partial Matching (PPM) compression algorithm and use enriched command-line data. It is seen that the NCD based on PPM performs better than the other compression methods.

PPM [125] is a lossless compression technique based on arithmetic encoding that attempts to make $|compress(x)|$ as close as possible to the ideal $-\log P(x)$ by allowing symbols within a message to be encoded with fractional quantities of bits. If the symbols in a message x follows some underlying distribution Q, so that the true probability of x is $Q(x)$, then the optimal lossless code will use $-\log Q(x)$ bits in describing x. Thus, arithmetic codes yield excellent lossless source coding performance when provided with a good estimate/prediction $P(x)$ for the true probability $Q(x)$ of x.

The arithmetic code in PPM rely on Markov model for forming the conditional probability estimates needed to code symbol x. PPM is based on the assumption that the probability distribution of current command relies only on the previous t commands in the stream, i.e. $P(c_n|c_{n-1}, c_{n-2}, \ldots, c_{n-t})$. The number of previous symbols, t, determines the order of the PPM model. Given an order t, PPM computes the likelihood of the next symbol, c_n, based on a context of size t, i.e. $P(c_n|c_{n-1}, c_{n-2}, \ldots, c_{n-t})$. If the prediction can not be estimated on the context of size t, it accumulates the probability of *escape character* at this level and the prediction is attempted on a context of lower size $t-1$, $t-2$ etc. This process continues till a prediction is made or no more symbols remain in context. If the prediction is achieved on a context of length q, $0 \leq q \leq t$, $P(c_n)$ is estimated to be the product of probabilities of escape characters at levels $t, t-1, \ldots, q+1$ multiplied by $P(c_n|c_{n-1}, \ldots, c_q)$. Different versions of PPM use different heuristics to determine the probability of escape character.

2.3.3 Sequence Based Methods

Sequence based methods deal with a sequence of commands rather than a single command at a time. These methods consider the temporal (sequencing) information of a command sequence unlike the frequency based methods which ignore the sequencing information by assuming command independence.

2.3.3.1 Sequence-match

Sequence-match method is based on command to command match between two command sequences of equal length. Schonalu et al. [99] use the sequence-match method proposed by Lane and Brodley [64]. It builds user's profile consisting of command sequences of length n extracted from the historical data of the user (in a overlapping window fashion). Thus, the profile is a set, $\{P\}$, and each instance $y \in \{P\}$ is a subsequence of length n. Given a test sequence $b = c_1, \ldots, c_T$; subsequences of same length are extracted, $x_i = c_i, \ldots, c_{i+n-1}$, $1 \leq i \leq (T - n + 1)$; and are compared with the known profile. The comparison is performed between all $y \in \{P\}$ and each x_i via a similarity measure $sim(x, y)$, which makes a point-by-point comparison of two sequences, x and y, counting matches and assigning greater weight to adjacent matches. Similarity to the profile $sim_{\{P\}}(x)$, is defined by: $sim_{\{P\}}(x) = \max_{y \in \{P\}} sim(x, y)$ [64]. A score μ is computed for b, which is defined as,

$$\mu = \frac{1}{T - n + 1} \sum_{i=1}^{T-n+1} sim_{\{P\}}(x_i) \tag{23}$$

if μ is below a threshold τ, then b is classified as masquerade, otherwise b is treated as normal.

2.3.3.2 Sequence Alignment

Sequence alignment is largely used in bioinformatics to find similarity between two biological sequences, such as DNA or protein sequences. It can be viewed as a generalization of the longest common subsequence problem [118] in which, given two strings $A = \alpha_1\alpha_2 \ldots \alpha_m$ and $B = \beta_1\beta_2 \ldots \beta_n$ $(n \leq m)$ over alphabet $\sum$, the goal is to find the maximal length lexically similar subsequences of A and B. It can be achieved by introducing *gaps* into the sequences so that the matching common subsequences are aligned. Sequence alignment uses a scoring approach to promote certain matches and to discourage others and to achieve this, each of the three cases *match*, *mismatch* and *gap* is given some weight.

In general, there are several modes of alignment, including global, semi-global, and local alignments. Global alignment, known as the Needleman-Wunch algorithm [82], tries to maximize the length of the subsequence over the entire length of both strings. It is

useful when both strings are of approximately the same length and the entirety of both sequences should be similar. Local alignment, known as the Smith-Waterman algorithm [107], focuses instead on finding the best aligned substrings of the two sequences over all possible substrings, rather than over the entire sequence. Local alignment is useful when searching for areas of functional similarity between sequences, or when one sequence is significantly longer than the other. In such cases the majority of the sequence can be assumed to be dissimilar except for the area of functional similarity, and so global alignment would provide a poor alignment. Finally, semi-global alignment allows for large areas of the sequences to be aligned as in global alignments, but also allows dissimilar prefixes and suffixes of the sequences to be ignored. Semi-global alignment is particularly useful in situations where the alignment of the entire length of the sequences should be dictated primarily by the alignment of several small, conserved subsequences.

Coull et al. [25] apply sequence alignment algorithm for masquerade detection. In the context of masquerade detection two command sequences of a user u are given, one is a training sequence $TR^u = a_1 a_2 \dots a_m$ (also called *signature*) and the other is an observed sequence $b = c_1 c_2 \dots c_T$ with $T \leq m$. The goal is to find the extent of dissimilarity between TR^u and b, and determine if this dissimilarity indicates a masquerade attack. It is equivalent to aligning the two strings and assuming that alignment with gaps or lexical mismatches may be indicative of possible masquerade attacks. Dynamic Programming is used to discover the optimal alignment among all possible alignments. It starts at the upper left corner of a grid/matrix, and iterates through each position whose value is determined through a choice of three transitions: diagonal step, vertical step, horizontal step and finally ends at the lower right corner. In [26], Coull et al. observe that scoring systems in bioinformatics are based on domain knowledge about the form of mutations and propose two models of mutation in command sequence data such as *Command Grouping* and *Binary Scoring*.

Command Grouping: It divides all commands into different groups based on their functionalities. In this scoring system, during a mismatch, the groups to which the two commands belong are compared. If the two commands belong to two different groups then it is an unexpected mutation and hence the mismatch is penalized by -1, otherwise if they

belong to same group it is an acceptable mutation with reward of +1.

Binary Scoring: It assumes that a user is prone to use the same set of commands as that of commands present in TR^u. Thus, any command in TR^u can replace any other command in TR^u, but use of previously unseen command is labeled as anomalous behavior. In other words, it allows various permutations of commands in TR^u without reducing the score significantly. Experimentally Coull et al. [26] show that binary scoring performs better then all other sequence alignment methods.

2.3.4 Feature Vector Based Methods

Feature vector based methods try to build a profile of a user or a group of users from the normal command usage data of the user(s) such that it captures the unique usage pattern of each user. Such user profiles are represented as vectors which are based on extracted features from the training data of users. We describe feature vector based masquerade detection methods such as Non-negative Matrix Factorization, Eigen Co-occurrence Matrix, Support Vector Machine and weighted RBF Naive Bayes in the following sections.

2.3.4.1 Non-negative Matrix Factorization

Non-negative Matrix Factorization (NMF) is a method developed by Lee and Seung [65]. NMF expresses a feature matrix V as the product of two non-negative matrices W and H.

$$V_{ij} \approx (WH)_{ij} = \sum_{\mu=1}^{r} W_{i\mu} H_{\mu j} \tag{24}$$

Where the dimensions of the matrices V, W and H are $n \times m$, $n \times r$ and $r \times m$ and $r \leq \frac{mn}{m+n}$. The idea is to compute r basis vectors such that columns of V can be expressed as non-negative linear combination of the basis vectors as given in Equation 24. Each column of matrix W contains a basis vector while each column of H contains encoding coefficients needed to approximate the corresponding column in V. Thus, W represents the profile of columns of V. The following iterative learning rules are used to find the linear decomposition [74]:

$$H_{\mu j} \leftarrow H_{\mu j} \sum_i (V_{ij}/(VH)_{ij}) W_{i\mu}$$

$$W_{i\mu} \leftarrow W_{i\mu} \sum_j (V_{ij}/(WH)_{ij}) H_{\mu j} \tag{25}$$
$$W_{i\mu} \leftarrow W_{i\mu} / \sum_a W_{a\mu}$$

The unsupervised multiplicative learning rules, as shown in Equation 25, are used iteratively to update W and H. The initial values of W and H are fixed randomly. With these iterative updates, the quality of the approximation of Equation 24 improves monotonically with a guaranteed convergence to a locally optimal matrix factorization [65].

NMF is applied by Wang et al. [121] to build normal behavior models in anomaly intrusion detection systems where the user behaviors are profiled by the frequency property. Let TR^u consists of n unique commands. TR^u is divided into m sessions and is represented as a matrix V^u such that the entry V^u_{ij} is the number of times the i^{th} command appears in the j^{th} session in TR^u. The NMF generates factors W^u and H^u for V^u such that the columns of W^u represent the basis profiles for u and the columns of H^u represent the encoding coefficients.

Given the test sequence b, it is represented as a column vector b^u, such that the ith entry in b^u represents the frequency of the ith command in b. b^u is approximated as the linear combination of columns of W^u.

$$\hat{b}^u = W^u \, t^u \tag{26}$$

It is expected that the encoding coefficients of $\hat{b}^u$ with respect to W^u, i.e. t^u, should not deviate much from the encoding coefficients of the training data of u, i.e. H^u. In order to measure this quantity we determine C, by linear regression such that,

$$y = C^u H^u \tag{27}$$

where $y = 1_{1\times m}$. If b deviates significantly from those of the normal behaviors learned in TR^u, then $|C^u \, t^u - 1| > \tau$, which indicates that the observed session b belongs to a masquerader. So the classification rule can be expressed as follows,

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } |C^u \, t^u - 1| > \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

Mex-Perera et al. [74] propose an improved NMF method for masquerade detection by including a normalization phase of the matrix V^u. During normalization, they apply different weights to modify the entries of V^u. For instance, it gives lower weights to common commands than rare commands. The normalization is done as,

$$\hat{V}^u_{ij} = \alpha_{ij} \cdot V^u_{ij} \tag{28}$$

where, the weights α_{ij} are calculated as,

$$\alpha_{ij} = -\log(P_i)/(L - N_j) \tag{29}$$

where, $P_i = N_{uk}/N_u$, L is the length of the jth session, and N_j is the number of times that commands not seen in TR^u appear in the jth session. During training $\hat{V}^u_{ij}$ is used instead of V^u_{ij} as input to NMF. For classification, a score is computed as,

$$d = \|b^u - \hat{b}^u\|^2(N_j + 1) \tag{30}$$

Score d is compared with a predefined threshold τ, and if $d > \tau$, then b is classified as masquerade else b is considered as a legitimate sequence of u.

2.3.4.2 Eigen Co-occurrence Matrix (ECM) Method

Eigen co-occurrence matrix (ECM) method profiles the normal usage patterns of users by extracting the casual relationship features embedded in the training sequences of events (commands). The casual relationship is extracted by correlating an event with any following events that appear within a certain distance (*scope size* s). The strength of the correlation between two events is defined by the distance between the events and the frequency of their occurrence. The closer the distance between two events is, or the more frequent an event pair is, the stronger their correlation becomes. Oka et al. [84, 83] uses ECM for masquerade detection where they consider the strength of the correlation by considering only the frequency component of event pairs.

Let $O = \{o_1, o_2, \ldots, o_m\}$ be the set of unique events (commands) appearing in TR. The first step of ECM involves preparing *event co-occurrence matrices* of dimension $m \times m$ using windows of size w and a scope of size s. It divides TR into non-overlapping windows of

size w and prepares an event co-occurrence matrix for each such window. Each element of an event co-occurrence matrix M_{ij} contains the number of times event o_j follows event o_i within a distance of s (scope size) in the whole window. Let there be L such event co-occurrence matrices constructed from TR: $M_1, M_2, \ldots, M_L$. The mean event co-occurrence matrix M_0 is computed and is subtracted from each $M_k, 1 \leq k \leq L$.

$$A_k = M_k - M_0 \tag{31}$$

Where,

$$M_0 = \frac{1}{L} \sum_{k=1}^{L} M_k \tag{32}$$

and A_k is the kth co-occurence matrix with the mean subtracted. The covariance matrix is constructed as,

$$P = \sum_{k=1}^{L} \hat{A_k} \hat{A_k^T}. \tag{33}$$

where $\hat{A_k}$ is the vector representation of A_k, prepared by concatenating its elements into a single vector. The matrix size of P is $m^2 \times m^2$, and each of its elements represents the correlation between two event pairs.

Principal Component Analysis (PCA) is applied on P to calculate the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors can be ranked in order of descending eigenvalues, namely $(v_1, v_2, \ldots, v_{m^2})$, thus ranking the principal components according to significance. Each eigenvector v_i can be represented as a $m \times m$ matrix and is called an *Eigen co-occurrence matrix* denoted as V_i. The first N out of m^2 eigenvectors are chosen to represent the *event co-occurrence matrix space*. Oka et al. [84] select the value of N such that the contribution rate is about 90%, which is defined as,

$$contributionrate = \frac{\sum_{i=1}^{N} \lambda_i}{\sum_{i=1}^{m^2} \lambda_i} \tag{34}$$

where λ_i denotes the ith largest eigenvalue.

The feature vector of a co-occurrence matrix, M, is obtained by projecting it onto the defined co-occurrence matrix space. The feature vector $F^T = [f_1, f_2, \ldots, f_N]$ is obtained by the dot product of vectors v_i and $\hat{A}$, where f_i is defined as,

$$f_i = v_i^T \hat{A} \quad \text{for } i = 1, 2, \ldots, N. \tag{35}$$

Where $\hat{A} = M - M_0$ and each f_i represents the contribution of ith eigen co-occurrence matrix.

In [84], Oka et al. create the profile of each user u from his training data TR^u. TR^u is divided into windows of size w, and an event co-occurrence matrix is created for each such window with a scope size of s. Each co-occurrence matrix is then projected onto the eigen co-occurrence matrices to obtain the feature vectors as per Equation 35. If TR^u is divided into t windows of length w each, then u's profile is composed of t feature vectors $F_1, F_2, \ldots, F_t$. Given b, a feature vector F is obtained in the same manner and is compared with the profiled feature vectors $(F_1, F_2, \ldots, F_t)$ by a simple Euclidean distance measure. If the distance between F and any of the profiled feature vector exceeds a threshold, then b is classified as a masquerade command sequence.

In [83], Oka et al. extends the ECM method to profile each user as a layered network. Once a feature vector F is obtained from a co-occurrence matrix, it is converted into a layered network. The ith layer of a network is obtained by multiplying the corresponding ith eigen co-occurrence matrix V_i by the ith element f_i of F. The ith network layer (for $i = 1, \ldots, N$) is constructed by connecting the elements in $f_i V_i$. $f_i V_i$ is represented in relation to the average co-occurrence matrix by separating it as,

$$\sum_{i=1}^{N} f_i V_i = \sum_{i=1}^{N} (X_i + Y_i) = \sum_{i=1}^{N} X_i + \sum_{i=1}^{N} Y_i \tag{36}$$

where X_i (or Y_i) denotes an adjacency matrix whose elements are determined by the corresponding positive (or negative) elements in $f_i V_i$. The networks obtained from X_i and Y_i are called positive and negative networks respectively. In [83], only positive network is used to define a user's profile $(\psi_1, \ldots, \psi_t)$. Given b, a feature vector F is obtained in the same manner and is multiplied with the eigen co-occurrence matrices to obtain a layered network (ψ_b). ψ_b is compared with the profiled network of u to obtain a score μ, defined as,

$$\mu = \max_i \ sim(\psi_i, \psi_b) \tag{37}$$

and

$$sim(\psi_i, \psi_j) = \sum_{k=1}^{N} \Gamma(T_k(i), T_k(j)) \tag{38}$$

where, $T_k(i)$ is the network obtained at the kth layer of ψ_i and $\Gamma(T_k(i), T_k(j))$ is the number of subnetworks that are common in both $T_k(i)$ and $T_k(j)$. If μ is below a threshold then b is classified as masquerade, otherwise b is assumed as normal.

2.3.4.3 Support Vector Machine

Support Vector Machine (SVM) is a collection of machine learning algorithms for binary classification. The fundamental principle of SVM is to map feature vectors to a high dimensional space and to compute a hyperplane that not only separates the training vectors for different classes, but also maximizes this separation by making the margin as large as possible. Thus, SVM aim at minimizing generalization error by maximizing the margin between training sets of two classes. The training examples that define the maximum margin are called support vectors. There have been some proposal to employ SVM for masquerade detection [119, 59, 103, 126].

In [59], Kim et al. use SVM to train the classifier by treating training data of each user as negative example and data of other users as positive examples. It represents the training set as vectors containing command frequencies in fixed window. To reduce the dimension of the feature vectors a concept of *common commands* are used. A set of commands used by more than X number of users at the rate exceeding $Y\%$ is tagged as common commands. They use RBF kernel well-known in SVM literature for training. Given b, it is divided into overlapping windows with a sliding window of size 10. Feature vectors are prepared for each of the window and the SVM predictor determines if each such window is normal or not. If the number of masquerade windows in b exceeds a threshold, b is tagged as masquerade.

Seo et al. [103] observe that RBF kernel used by Kim et al. [59] is not suitable for sequence data as it uses only frequencies of commands without considering sequence information. Thus, they propose to use kernels that have sequence information such as, *string kernel* and *K-gram kernel* for masquerade detection.

String Kernel: String kernel, originally proposed by Lodhi et al. [71], compares two strings according to the non-contiguous substrings they contain. Let $\sum$ be a finite alphabet (command). Any $s \in \sum^k$, $k = 0, 1, \ldots$ is called a string and $\sum^*$ represents the set of all

non-empty strings. Let $|s|$ denotes the length of string $s = s_1, \ldots, s_{|s|}$, $s[i:j]$ denotes substring $s_i, \ldots, s_j$. We say that u is a substring of s if there exists indices $\mathbf{i} = (i_1, \ldots, i_{|u|})$ with $1 \leq i_1 < \ldots < i_{|u|} \leq |s|$ such that $u = s[\mathbf{i}]$. The length $l(\mathbf{i})$ of the subsequence in s is $i_{|u|} - i_1 + 1$. The string kernel as proposed by Lodhi et al. [71] is defined as,

$$K_n(s,t) = \sum_{u \in \sum^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}. \tag{39}$$

which is same as,

$$K_n(s,t) = \sum_{u \in \sum^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{i_n+j_n-i_1-j_1+2}. \tag{40}$$

The string kernel has two parameters: n which stands for the number of non-contiguous sequence and $\lambda \in (0,1)$ which is a parameter that controls the penalization of the interior gaps in the substrings.

K-gram Kernel: K-gram kernel transforms strings into high dimensional feature vectors where each feature corresponds to a continuous substring. The K-gram kernel used by Seo et al. [103] is a special case of the string kernel defined by Vishwanathan et al. [116], which is shown in Equation 41.

$$K(s,t) = \sum_{x \in \sum^*} \text{num}(x,s)\ \text{num}(x,t) w_x. \tag{41}$$

where $\text{num}(x,y)$ denotes the number of times substring x occurs in y. This kernel counts the number of occurrences of every substring x in both s and t and weight it by w_x. The K-gram kernel used by Seo et al. [103] is given in Equation 42. It adds a 1 to num() to avoid the zero occurrence case and assigns $w_x = 1$.

$$K(s,t) = \sum_{x \in \sum^n} (\text{num}(x,s)+1)(\text{num}(x,t)+1). \tag{42}$$

The string kernel deals with n unconnected features, whereas the K-gram kernel handles n connected features. Seo et al. experimented with both the kernel methods. They use two-class SVM to profile a user u, where TR^u is treated as self data and training data of all other users is treated as nonself data. Once profile of u is built, SVM classifier examines b to determine whether its normal or masquerade.

Wang et al. [119] argue that for masquerade detection problem, it is probably impossible and unreasonable to estimate how an attacker would behave. Thus, treating sets of other users' data as positive examples provides a substantive bias (to those users' behavior who probably were not behaving maliciously). Thus, they propose to use one-class SVM (OCSVM) for detection of masqueraders.

One-class SVM (OCSVM): OCSVM is an outlier detection technique, originally proposed by Scholkopf et al. [98], where examples of only one-class (*self* data) is used for training the classifier. The simplest way to express the one-class SVM is to envision a sphere or ball, and the objective is to squeeze all of the training data into the tightest ball feasible. Let TR^u consists of vectors $x_1, x_2, \ldots, x_n$. Training of OCSVM is formulated as a optimization problem (dual form) as follows.

$$\min_{\alpha} \ \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j), \tag{43}$$

$$\text{such that } 0 \leq \alpha_i \leq \frac{1}{vn}, \quad \sum_i \alpha_i = 1, \ \ i = 1, 2, \ldots, n.$$

where, $K(.)$ is a kernel function, α_i is a Lagrange multiplier and $v \in (0, 1)$ is a parameter that controls the trade-off between maximizing the number of data points contained by the hyperplane and the distance of the hyperplane from the origin. After solving for α_i, the decision function to classify a new data x is,

$$f(x) = \text{sgn } (\sum_i \alpha_i K(x_i, x) - \sum_i \alpha_i K(x_i, x_j)). \tag{44}$$

$f(x) = +1$ means x is *self* (normal) and $f(x) = -1$ means x is *nonself* (masquerade).

Yang et al. [126] propose to use OCSVM with string kernel defined by vishwanathan et al. [116]. They modify the string kernel defined by vishwanathan et al. shown in Equation 41, to derive two different kernel functions, which are as follows.

Standard Kernel: For $k > 0$,

$$w_x = \begin{cases} 1, & |x| \leq k \\ 0, & |x| > k \end{cases}$$

Thus, only contributions of substring of length k or less are considered.

Simple Kernel: It defines the kernel function as,

$$K(s,t) = \sum_{x \in \sum^*} I(\text{num}(x,s)) I(\text{num}(x,t)) w_x. \tag{45}$$

where, the indicator function I(y) is defined as,

$$I(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases}$$

and the weight function w_x is same as defined in standard kernel. Thus, it only takes into account whether the substring occurs or not, rather than the number of occurrences.

2.3.4.4 Weighted RBF Naive Bayes

Sharma et al. [104] use a weighted RBF similarity measure with Naive Bayes classifier for masquerade detection. They represent the training set of each user as a vector a_j using the frequency information of the training commands. Given b from user u, it is represented as a vector of same size, let it be v. A score r is computed as shown in Equation 46.

$$r = \gamma(1 - \lambda(v, a_u)) \tag{46}$$

where,

$$\gamma = \frac{\log self(b,u)}{\log nonself(b,u)}$$

and $\lambda = \mu(vb, ab_u)k(v, a_u)$. vb and ab_u are the binary representation of the vectors v and a_u respectively. k is Gaussian radial basis function and is defined as $k(v, a_u) = \exp(-\frac{1}{2}\|v - a_u\|^2/\|a_u\|^2)$. μ is a weight parameter defined as,

$$\mu(vb, ab_u) = \frac{\sum \text{and}(vb, ab_u)}{\sum \text{or}(vb, ab_u)} \tag{47}$$

A high value of r indicates dissimilarity between b and a_u, and low value signifies similarity. Thus, the decision about b is made as,

$$Decision(b) = \begin{cases} \text{masquerade}, & \text{if } r > \tau \\ \text{normal}, & \text{otherwise} \end{cases}$$

2.3.5 Summary of methods

Masquerade detection is a form of anomaly detection because the behaviour of a masquerader is unknown and is assumed to be different from that of a real user. Thus, the methods discussed above profile the users based on their past command usage data (behaviour). Profiling generally means finding some pattern of activity/usage which can distinguish a user from others efficaciously. A user can be profiled independently, as seen in most of the above methods, or a group of *similar* users are profiled together, as mentioned by Smaha et al. [106]. During profiling importance can be given to unique and uncommon commands to effectively reflect diverse command patterns, as done in uniqueness method [100, 99] and in SVM based method proposed by Kim et al. [59]. Some of the methods assume that the commands generated by a user are independent of each other like the Naive Bayes method [75], whereas some other methods assume dependency among commands such as in Bayes one-step markov [34], Hybrid multistep markov [99], Filtering approach [53], Sequence prediction [29, 99]. Methods like Uniqueness [100, 99], Naive Bayes [75], Non-negative matrix factorization [121], Eigen co-occurrence matrix [83, 84], SVM based methods proposed by Kim et al. [59], Wang et al. [119] considers the frequency information of commands and ignore the sequence information. Whereas, methods like Compression [99, 13, 12], Sequence match [99], Sequence alignment [25, 26], String kernel based SVM methods [103, 126] take into account the sequence information in user generated commands.

At this stage, it is apt to study the data sets that are considered to be standard for the study of command line based attacks. In the next section, we discuss about datasets for the study of masquerade attack and its detection.

2.4 Dataset

There are three standard datasets available for studying the characteristics of masquerade attack. One is provided by Schonlau et al. [99], which is a truncated command dataset collected from Unix *acct* auditing mechanism and is commonly called as SEA (derived from **S**chonlau **e**t **a**l.) dataset. The other two are provided by Lane and Brodley [64] and Greenberg [45]. These datasets are explained in the following sections.

2.4.1 SEA Dataset

SEA dataset is provided by Schonlau et al. which consists of 15000 commands for each of 50 users. These commands are logically divided into blocks of 100 commands each. So, each user's data consists of 150 blocks of commands. As described by Schonlau et al. [99] the dataset is prepared as follows.

User commands are collected using the Unix *acct* auditing mechanism which includes information such as command name, user name, terminal, start time, end time, real (sec), CPU (sec) and memory usage. Due to privacy reasons only two fields are taken - *command name* and *user*. Examples of commands are: `sed`, `eqn`, `troff`, `dpost`, `echo`, `sh`, `cat` and so forth. The first 15000 commands for each of about 70 users are collected over a period of several months. Out of which 50 users are randomly selected to serve as the intrusion targets and the remaining 20 users as masqueraders. For simplicity, each user's data is decomposed into 150 blocks of 100 commands each. The first 50 blocks (5000 commands) of all 50 users are kept aside as training data (legitimately issued by the users). The rest 100 blocks (51 through 150) are randomly contaminated with blocks from the 20 masquerading users, such that a block is contaminated completely, "dirty" block, or not at all, "clean" block. The task is to classify the "dirty" blocks as masquerade blocks and the "clean" blocks as normal/self blocks. Two different detection configurations are adopted for experiments using this dataset which are described below.

SEA Configuration: This configuration is originally due to Schonlau et al. [99], which uses the first 5,000 commands of each of the 50 users for training the classifier, and the rest of the 10,000 commands to test the classifier. The entire test data of each user is divided into blocks of 100 commands each and each block is classified as being masquerade or legitimate. This is the configuration which naturally follows from the way that the dataset is generated.

1v49 Configuration: Maxion and Townsend [75] propose another configuration called 1v49 configuration. In this configuration, the first 5,000 commands of a user are used to build a model for that user, and the first 5,000 commands of the remaining 49 users are used as masquerader data for this user. This is contrary to SEA configuration where each user's last 10,000 commands containing simulated masquerade blocks are used as test data.

However, 1v49 configuration has a richer source of masquerade data in that 2450 (i.e., 50 blocks, 49 users) command blocks are used as masquerader data.

2.4.2 Lane and Brodley Dataset

Lane and Brodley [64] collect user command information from different authorized users of UNIX hosts at the Purdue Millennium Machine Learning Lab. It contains 9 sets of sanitized user data drawn from *tcsh* history files of 8 UNIX computer users over the course of up to 2 years (USER0 and USER1 are generated by the same person working on different platforms and different projects). The data is parsed and sanitized to remove file names, user names, directory structures, web addresses, host names, and other possibly identifying items. In this dataset users' data are called token streams, where a token means a command name or a processed command parameter. Sessions are concatenated by date order and tokens appear in the order issued within the shell session, but no timestamps are included in this data. The important feature of this dataset is that not only command name but also the number of parameters of the command are also included. A sample of the data is shown in Table 4. The amount of data available varied among the users from just over 15,000 tokens to well over 100,000 tokens, depending on their work rates and duration of study.

Table 4: A sample of user data from the Lane and Brodley dataset

```
**SOF**
setenv
<2>
xhost
<1>
rlogin
<1>
finger
<1>
**EOF**
**SOF**
vi
<1>
ls
vi
```

2.4.3 Greenberg Dataset

Another dataset collected by Greenberg [45] contains data of 168 different users of University of Calgary. The users are divided into 4 groups namely, Novice Programmers, Experienced Programmers, Computer Scientists and Non-programmers according to their computer experience and needs. The complete command line data submitted by a user is captured as a chunk after it is entered and processed by UNIX *csh* command interpreter. It also includes extra information known to *csh*, including the use of history and alias and the current working directory of the users. Login sessions are distinguished by a record that notes the start and end time of each session. Command lines (C) entered during this period are then listed in following records, each annotated with the current working directory (D), alias substitution (if any) (A), history use (H) and error status (X). A sample of the data is shown in Table 5.

Table 5: A sample of user data from Greenberg dataset

C bye D /user/cpsc500/l01b91/xxxxxx A logout H NIL X M 09
C rwho | more D /user/cpsc500/l01b91/xxxxxx A NIL H NIL X NIL

Most of the techniques proposed in the literature based on truncated command-line data use SEA dataset accepting it as a benchmark dataset for masquerade detection. In that sense, anomaly detection in SEA dataset has become the problem of masquerade detection. The following sections describe our proposed methods for masquerade detection.

2.5 Episode Mining

Most of the masquerade detection techniques that are proposed in recent years depend on truncated or enriched command line data. It is assumed that individual commands play a

very important role in determining legitimate or questionable behaviour of a user. But to the contrary, the usage by a user consists of a set of commands for a particular task, rather than isolated commands. For example, `dvips` and `gview` can be viewed as one interrelated pair of commands and natural to occur together. Thus, identifying an user to be legitimate or masquerader based on individual commands may become robust enough for successful identification but may generate many false positives also. Thus, it is more natural to have a masquerade detection technique based on *episodes*, meaningful subsequences of commands, rather than with individual commands. To accomplish this, it is necessary to determine meaningful subsequence of commands that may define a single task or purpose. We illustrate a motivating example in the following section.

2.5.1 Motivating Example

While experimenting with the existing techniques for different datasets, we found many interesting cases that motivated the current research. Let us consider the following example. For User 10 of SEA dataset, the training set contains many unique commands – commands which are unique to this user. These are `op-newus, pg, where, fls_star, arch_uni, op_mko, test.m2., m3_flse` etc. Interestingly, in the 24^{th} block in the test data of this user none of these commands appears. It is, however, a legitimate block for this user. On the other hand, this block contains commands like `netscape` and `telnet`, which are used by this user (in the training set) much less frequently than other users' use. It may be noted that `netscape` is used 9 times (6 times in a row) by User 10 and `telnet` is used 7 times (4 times in a row). Thus, the ratios of *self-probability* to *nonself-probability*, as defined in Equations 7 and 8 respectively, of these two commands are very low and because of their repeated use, Naive Bayes technique identifies this as a masquerade block – a *false positive* – even when there is strong evidence of user 10 signature in commands like `launchef`.

In another instance, Killourhy et al. [58] observe that Naive Bayes consistently fails to detect masquerade instance in two of the users data of the SEA dataset. On analysis it is observed that those blocks contain commands which are not previously seen in TR and are named as never-before-seen commands (NBSCs). According to the formulation of self and

nonself in Naive Bayes method, the NBSCs result in increasing the self score keeping the nonself score constant, thus forcing the classifier to detect it as a normal block.

The observations by Killourhy et al. [58] validates our findings that few individual commands can greatly affect the overall score of a block irrespective of the presence of a large number of commands with high ratio.

It is to be noted that a command with low ratio, if used repeatedly for as low as 10% times, may drag the overall probability of the block to a low value yielding a false positive. This may happen even if there are other commands which have reasonably high ratio. On the other hand, if we identify episodes, the continuous stream of 6 `netscape` commands of the 24^{th} block of User 10 can be an episode. This subsequence would be detected as a masquerade episode. There may be nearly 20 episodes for this block of User 10 and only few of them are detected as masquerade. One such masquerade episode is the consecutive sequence of `netscape` commands. The total number of commands in all masquerade episodes is much less in proportion to the full length of the block and hence the block can be recognized as a legitimate block. Thus, if a user deviates from its usual commands and uses the less frequent commands for few times, then existing algorithms may raise a false alarm. On the other hand, it would be better to recognize the episode as masquerade and if the user does not deviate too much from his own pattern set of episodes then episode based technique can still recognize the block as a legitimate one. In this way it can also take care of the momentary *concept drift* by a user.

The proposed method is based on the foregoing discussion. We show in Figure 1 the block diagram outlining the major modules of the proposed episode based masquerade detection method.

In the next section we describe the episode discovery algorithm.

2.5.2 Episode Discovery Algorithm

In this section, we propose an algorithm to extract meaningful subsequences, *episodes*, from a continuous sequence of commands. In other words, it involves breaking a given sequence at

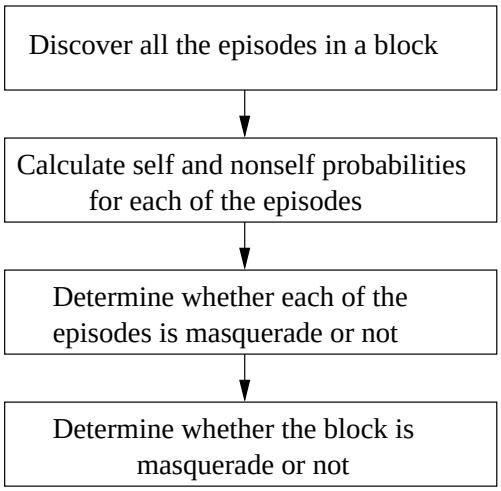


Figure 1: Block Diagram of episode based masquerade detection algorithm

appropriate places/positions to get meaningful subsequences (statistical markers of boundaries of meaning carrying subsequences.). So, it essentially involves finding appropriate positions in a sequence to break it.

We adopt the *Voting-Experts* paradigm proposed in [23] for this purpose. The proposed episode discovery algorithm is concerned with assigning score to every element of the sequence so that higher value of the score indicates more likelihood of the position being an appropriate position for a break point. The scores for each position are accumulated by sliding a window of fixed length over the given sequence. While the window slides from left to right, we accumulate score based on *entropy* and *frequency* information. Positions with local maximum scores are slected as break points. The main intuition behind the entropy and frequency information is the following.

Entropy remains low inside meaningful units and increases at their boundaries. In a subsequence, if any element precedes many distinct elements then it is difficult to determine any pattern of occurrence of the pair of elements. Hence, the entropy at this element has a very high value. On the other hand, if there is any specific pattern of occurrence then the entropy would be low. Similarly, we assign high score when the subsequence is very frequent, which is attributed to being more meaningful than low-frequency ones. In order to compute these scores efficiently, it is proposed to compile the sequence data in the form of a *trie* of n-grams. This data structure is then used to determine the scores at every

location. Thus, the episode discovery algorithm can be divided into three major steps as follows:

- Construction of trie
- Calculation of boundary entropy using the trie
- Finding episodes using the n-gram trie structure

We describe each of these steps in detail in the following sections.

2.5.2.1 Construction of trie

A *trie* is an ordered tree of depth d such that each distinct subsequence of length $d-1$ is a path from root node to a leaf node in this tree. Two subsequences having common prefix share common ancestors representing the prefix fragment. At every node, the frequency indicates the frequency of the subsequence represented by the path from root to the current node i.e., it represents the number of times the subsequence appears in the whole sequence. In Figure 2, we provide the algorithm for construction of the trie.

We illustrate the concept with the following example.

Example 2.5.1. Let us consider the sequence of six commands: `xrdb, cpp, sh, cpp, sh, mv`. For this sequence a trie of depth 3 can be generated as shown in Figure 3 using the algorithm depicted in Figure 2. We can observe that the leaf node labeled `sh` (second from left) represents the subsequence {`cpp, sh`} and hence the number 2 at this node indicates the frequency of {`cpp, sh`} in the given sequence. Similarly, each of the subsequences {`xrdb, cpp`}, {`sh, cpp`} and {`sh, mv`} is present once. The two subsequences {`sh, cpp`} and {`sh, mv`}, have a common prefix {`sh`}, which is also the common ancestor for the corresponding nodes. □

Nodes at level k of the trie refers to all the subsequences with length k. For example nodes at level 2 in example 2.5.1 refers to subsequences with length 2, i.e. {`xrdb, cpp`}, {`cpp, sh`}, {`sh, cpp`}, {`sh, mv`}. It can be observed that the sum of the frequencies of all the nodes at level k is equal to $T-k+1$. For example the sum of the frequencies of all

Algorithm: To build a trie for a given command sequence b
Input: Sequence of commands $b = c_1, c_2, \ldots, c_T$ and depth d

1. Initialize: root = NULL, $n = d - 1$
2. **for** $i = 1$ to T **do**
3. **if** root has a child node labeled c_i **then**
4. increment frequency of this child node by 1
5. **else**
6. add a new child node to root labeled c_i with frequency 1
7. **endif**
8. **for** $j = i - n + 1$ to $i - 1$ **do**
9. **if** $j > 0$ **then**
10. $S = c_j, \ldots, c_{i-1}$
11. **if** S has a child node labeled c_i **then**
12. increment frequency of this child node by 1.
13. **else**
14. add a new child node to S labeled c_i with frequency 1.
15. **endif**
16. **endif**
17. **enddo**
18. **enddo**

Figure 2: Algorithm for construction of a trie for a given command sequence

nodes at level 1 in example 2.5.1 is 6 (=1+2+2+1) and for level 2 it is 5. In a trie of depth d build from a sequence b, we can find all the subsequences up to length $d - 1$ present in b, along with the frequency of their occurrence. In other words, the trie contains all the n-grams (1-gram, 2-gram, ..., (d-1)-gram) available in b with their frequency information.

2.5.2.2 Calculation of boundary entropy using the trie

The entropy of a node refers to the entropy of the sequence from the root node to the concerned node. Let $f(x)$ be the frequency of the node x. Let x_i be a node and $parent(x_i)$ be the parent node of x_i. Let us assume that $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m$ are the siblings of x_i.

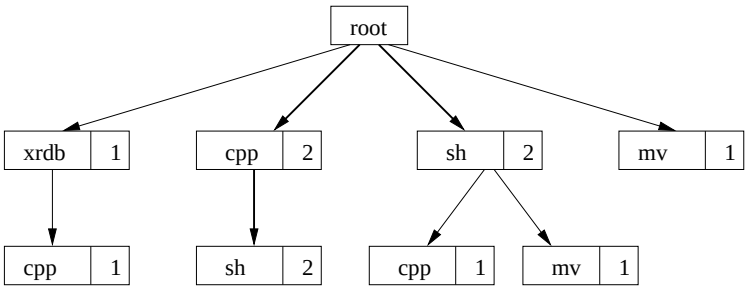


Figure 3: Trie for Example 1 with d=3. The thickness of edges indicates the frequency

The probability of a node x_i, denoted as $p(x_i)$, is given by

$$P(x_i) = \frac{f(x_i)}{f(parent(x_i))} \tag{48}$$

$P(x_i)$ is the probability of the subsequence represented at node x_i (subsequence obtained by traversing the path from the root node to node x_i). Let the subsequence represented at node x_i be $c_1, c_2, \ldots, c_p$ (node x_i is labeled with command c_p). One can see that $P(x_i)$ is equivalent to probability of command c_p being the next command given $c_1, c_2, \ldots, c_{p-1}$, i.e., $P(x_i) = P(c_p|c_1, c_2, \ldots, c_{p-1})$.

Similarly, the entropy of $parent(x_i)$ is given by

$$e(parent(x_i)) = -\sum_{j=1}^{m} P(x_j) \log P(x_j) \tag{49}$$

When the node is a leaf node the entropy is assumed to be zero.

Thus, each node of the trie has two parameters, one is frequency and the other is entropy (of course except the root node). For each level k, we calculate the mean frequency (f_k), mean entropy (e_k), standard deviation using frequency parameter (σ_{fk}) and standard deviation using entropy parameter (σ_{ek}). Let $x_1, x_2, \ldots, x_m$ be the nodes present at level k. We calculate f_k, e_k, σ_{fk}, σ_{ek} as follows.

$$\begin{aligned} f_k &= \frac{\sum_{i=1}^{m} f(x_i)}{m} \\ e_k &= \frac{\sum_{i=1}^{m} e(x_i)}{m} \\ \sigma_{fk} &= \sqrt{\frac{\sum_{i=1}^{m} (f(x_i) - f_k)^2}{m-1}} \\ \sigma_{ek} &= \sqrt{\frac{\sum_{i=1}^{m} (e(x_i) - e_k)^2}{m-1}} \end{aligned}$$

We then standardize the frequency and entropy values of each node at level k as,

$$f(x_i) = \frac{f(x_i) - f_k}{\sigma_{fk}}, \text{ and } e(x_i) = \frac{e(x_i) - e_k}{\sigma_{ek}}, \ 1 \leq i \leq m. \tag{50}$$

The intuition behind standardizing frequency and entropy is, we want to capture how unusual these frequencies and entropies are *relative to* other subsequences of the same length.

2.5.2.3 Finding episodes using the n-gram trie structure

To find episodes from a given command stream, it is necessary to find the correct boundaries within the stream. Both the frequency and entropy parameters contribute equally in finding the possible boundary by assigning scores to the probable boundary positions. The above trie data structure helps us in efficiently computing the entropy and frequency of subsequences at different boundary positions.

We slide a window of size n ($n + 1$ is the depth of the trie) on b and examine the subsequence within the window for possible boundary positions. For example, if an instance of the window contains the commands $c_1, c_2, \ldots, c_n$, then we examine the entropy and frequency at each location i, $1 \leq i \leq n$ within the window as follows.

The entropy at location i is the entropy of the node along the path $c_1, \ldots, c_i$ which is present at level i in the trie. After finding entropy at all locations i, $1 \leq i \leq n$, the location corresponding to the highest entropy is identified and its score is incremented by 1.

The frequency at location i is calculated by adding the frequencies of the subsequences $c_1, \ldots, c_i$ and $c_{i+1}, \ldots, c_n$. The location with the highest frequency is identified and its score is incremented by 1. In this case, our goal is to maximize the sum of the frequencies of the left and right subsequences of the probable boundary.

After sliding the window across the whole command sequence (b), we end up with scores for different locations in the sequence. A location is likely to accrue a locally-maximum score if it is repeatedly identified for boundary for different positions of the sliding window. We choose the location with the local maximum of score as boundary of the episode.

Example 2.5.2. We consider the 51^{st} block of user 1 of SEA dataset consisting of the following 100 commands.

java, .java_wr, expr, expr, dirname, basename, egrep, egrep, egrep, egrep, egrep, java, aacdec, cat, aiffplay, sh, aacdec, cat, aiffplay, sh, aacdec, cat, aiffplay, sh, netscape, netscape, netscape, netscape, netscape, netscape, netscape, aacdec, cat, aiffplay, sh, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, hostname, id, nawk, getopt, true, grep, date, lp, find, mkdir, expr, generic, cat, file, post, awk, cat, post, rm, generic, ln, ln, generic, lp, sh, getpgrp, LOCK, true, ls, sed, FIFO, cat, date, generic, generic, date, generic, gethost, download, tcpostio, tcpostio, tcpostio, tcpostio, cat, generic, ls, generic, date, generic, rm.

Taking depth of the trie as 5, 16 episodes are obtained as follows.

1. java, .java_wr, expr, expr, dirname, basename
2. egrep, egrep, egrep, egrep, egrep
3. java, aacdec, cat, aiffplay, sh
4. aacdec, cat, aiffplay, sh
5. aacdec, cat, aiffplay, sh
6. netscape, netscape, netscape, netscape
7. netscape, netscape, netscape
8. aacdec, cat, aiffplay, sh
9. netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape, netscape
10. hostname, id, nawk, getopt, true, grep, date, lp, find, mkdir, expr, generic, cat, file, post, awk, cat, post
11. rm, generic
12. ln, ln, generic, lp, sh, getpgrp, LOCK, true

13. `ls, sed, FIFO, cat, date, generic`

14. `generic, date, generic`

15. `gethost, download, tcpostio, tcpostio, tcpostio, tcpostio`

16. `cat, generic, ls, generic, date, generic, rm`

It can be seen that it has correctly identified {`aacdec, cat, aiffplay, sh`} to be an episode which occurs three times in the given sequence. The second episode in the list groups all the `egrep` commands occurring continuously thus forming a meaningful episode. □

2.5.3 Episode Based Masquerade Detection Method

We know that the behaviour of a masquerader cannot be identified just by one single command or instantaneously. And even a legitimate user may deviate from his normal behaviour for sometime. Thus, it is more appropriate to doubt a user who is consistently deviating from the normal signature rather than a user who deviates momentarily and returns to the normal pattern. Thus, it is necessary to detect the episodes which may be masquerade and take into account the length of such episodes to determine the block to be masquerade. We make use of Naive Bayes algorithm given in [75] for detecting masquerade episodes.

Let C be the set of unique commands in TR. Let C_j be the set of unique commands used by the user u_j during training and $f(c, u_j)$ be the frequency with which the command c is entered by the user u_j during training. We define the probability of use of a command c by user u_j as,

$$P(c, u_j) = \frac{f(c, u_j) + \alpha}{\sum_{c_i \in C_j} f(c_i, u_j) + \alpha|C|}$$

Similarly, we define probability of use of command c for a group of users except user u_j as

$$P(c, U_{\neq j}) = \frac{\sum_{u_k \in U \wedge k \neq j} f(c, u_k) + \alpha}{\sum_{u_k \in U \wedge k \neq j} \sum_{c_i \in C_j} f(c_i, u_k) + \alpha|C|}$$

where α is a pseudocount, and $|C|$ is the number of distinct commands in the data.

The pseudocount can be any real number larger than zero (0.01 in this study), and is added to ensure that there are no zero counts. The lower the pseudocount, the more

sensitive the detector is to previously unseen commands. The pseudocount term in the denominator compensates for the addition of a pseudocount in the numerator [75].

For a test block b of a user u_j we find all the episodes in b using the episode discovery algorithm as described in Section 2.5.2. For each of the episode e_k in b, we calculate the *self* and *nonself* probabilities of e_k as,

$$Self(e_k, u_j) = \prod_{c \in e_k} P(c, u_j) \tag{51}$$

$$Nonself(e_k, u_j) = \prod_{c \in e_k} P(c, U_{\neq j}) \tag{52}$$

Based on the ratio of *self* and *nonself* probabilities, we flag each e_k as masquerade (flagged as 1) or normal (flagged as 0) as shown in equation 53.

$$Masq(e_k) = \begin{cases} 1, & \text{if} \quad \frac{Self(e_k,u_j)}{Nonself(e_k,u_j)} \leq \lambda \\ 0, & \text{otherwise} \end{cases} \tag{53}$$

where λ is a threshold. Then we compute the score of b as,

$$score(b) = \sum_{e_k} |e_k| Masq(e_k) \tag{54}$$

where $|e_k|$ is the length of e_k. $score(b)$ represents the number of commands present in masquerade episodes of b. If this value is greater than a threshold (θ), then the block b is declared as a masquerade block. The algorithmic representation of the method is given in Figure 4.

2.5.4 Experimental Results

We experiment our algorithm with the widely used SEA dataset. The first 50 blocks (5000 commands) of each of the user is used for training and the remaining 100 blocks of data of each user is used for test purpose. As per the dataset, the test data consists of 5000 blocks (from all 50 users), out of which 4769 blocks are normal and the rest 231 blocks are masquerade bocks. During training we find 635 unique commands from the training data of all users. We compute $P(c_i, u_j)$ and $P(c_i, U_{\neq j})$ for each of these 635 commands, for all users.

Training Phase

1. **for** each user $u_j \in U$ **do**
2. **for** each unique command c_i in the training data of user u_j **do**
3. calculate $P(c_i, u_j)$ and $P(c_i, U_{\neq j})$
4. **enddo**
5. **enddo**

Testing Phase

1. **for** each block of commands b **do**
2. find all episodes in b
3. **for** each episode e_k in b **do**
4. calculate $Self(e_k, u_j)$ and $Nonself(e_k, u_j)$
5. **if** $(Self(e_k, u_j)/Nonself(e_k, u_j)) \leq \lambda$ **then**
6. $Masq(e_k) = 1$
7. **else**
8. $Masq(e_k) = 0$
9. **endif**
10. **enddo**
11. compute $score(b) = \sum_{e_k} |e_k| Masq(e_k)$
12. **if** $score(b) > \theta$ **then**
13. mark block b as *masquerade*
14. **endif**
15. **enddo**

Figure 4: Algorithmic representation of the Episode Based Masquerade Detection method

During the test phase, we apply the episode discovery algorithm on each test block of 100 commands to find the episodes. Each episode is then tested for masquerade using Equation 53. The threshold λ and the ratio of self vs non-self probabilities are used to determine whether an episode is masquerade or not. Based on the number of commands in all the masquerade episodes of a block we determine whether the block is masquerade or not. A block of 100 commands is detected to be masquerade if total number of commands in masquerade episodes exceeds θ. In our initial experiments we take θ as 40.

The classification result is reported in terms of *detection rate* (DR) and *false positive*

rate (FPR). DR is defined as the number of masquerade blocks detected divided by the total number of masquerade blocks (231). Similarly, FPR is equal to the number of normal blocks classified as masquerade blocks divided by the total number of normal blocks (4769). By changing the value of the threshold λ, we get different pairs of *detection rate* (DR) and *false positive rate* (FPR). The result of our method is shown in Table 6.

Table 6: Results of Episode based masquerade detection technique with $\theta = 40$

Detection Rate	False Positive Rate	Detection Rate	False Positive Rate
0.090909091	0	0.865800866	0.1214091
0.194805195	0.000209688	0.87012987	0.121828476
0.207792208	0.001048438	0.874458874	0.137345355
0.311688312	0.008387503	0.878787879	0.142797232
0.372294372	0.011323129	0.883116883	0.143426295
0.402597403	0.014887817	0.887445887	0.149297547
0.445887446	0.016984693	0.891774892	0.157685049
0.458874459	0.021597819	0.896103896	0.161878801
0.476190476	0.022855945	0.904761905	0.168379115
0.515151515	0.025581883	0.909090909	0.170475991
0.519480519	0.026001258	0.917748918	0.177185993
0.545454545	0.028098134	0.922077922	0.180331306
0.593073593	0.036485636	0.926406926	0.181589432
0.614718615	0.038372824	0.930735931	0.18263787
0.636363636	0.041937513	0.935064935	0.216607255
0.662337662	0.043824701	0.939393939	0.218284756
0.670995671	0.050115328	0.943722944	0.220591319
0.675324675	0.052841267	0.948051948	0.259593206
0.696969697	0.059970644	0.952380952	0.262948207
0.705627706	0.066680646	0.956709957	0.266722583
0.718614719	0.071922835	0.961038961	0.274481023
0.735930736	0.075277836	0.965367965	0.28077165
0.761904762	0.086181589	0.974025974	0.287062277
0.796536797	0.100650031	0.982683983	0.307192283
0.818181818	0.106101908	0.987012987	0.339064793
0.826839827	0.108827847	0.991341991	0.354791361
0.835497835	0.109037534	0.995670996	0.418117006
0.83982684	0.120989725	1	0.533235479

The accuracy of the method starts with a lowest DR of 9.09% and FPR of 0% and attains a maximum DR of 100% with 53.32% FPR. We plot a ROC (Receiver Operating Characteristics) curve as depicted in Figure 5 to show the overall performance of our method. ROC curve is a 2-dimensional plot for classification performance, with DR (also called true

positive rate) on the y-axis and FPR on x-axis. It can be seen from the ROC curve that our algorithm outperforms many of the earlier algorithms.

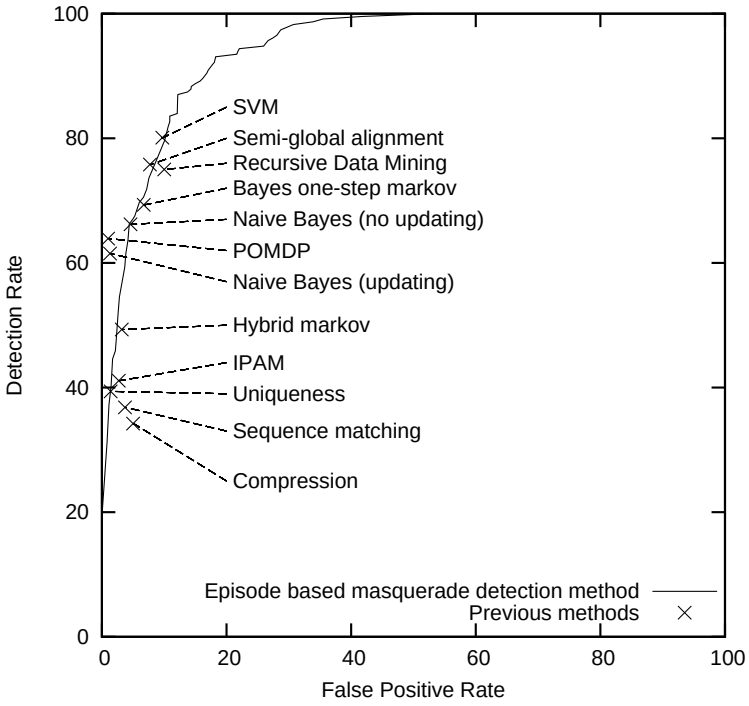


Figure 5: ROC curve of the Episode based masquerade detection method and its comparison with other methods

The best or optimal performance of a classifier can be examined by a cost function, such as,

$$Cost = \alpha(1 - DR) + \beta(FPR) \tag{55}$$

The cost of a false alarm in terms of a missed detection vary from system to system and depends more on the overall objective of the detection mechanism. Thus, a general notion to find the cost is by assigning 1 to both α and β, which penalizes both the false alarm and the misses equally. In this scenario, the optimal result obtained by our method is 93.07% DR with 18.26% FPR.

Maxion and Townsend in [75] propose the values of α and β to be 1 and 6 respectively, thus penalizing the false alarms by 6 times more than the missed cases. For $\alpha = 1$ and $\beta = 6$, the episode based method attains a lowest cost of 60.05 with a DR of 66.23% and

FPR of 4.38%.

We also test with different values of θ, such as 30, 50 and 60. But we observe that the deviations in the results are very small. We show the optimal results obtained for different values of θ in Table 7. We use $\alpha = \beta = 1$ to find the cost. From Table 7, we can see that for $\theta = 60$, the value of the cost function is the lowest with a DR of 91.77% and FPR of 16.71%.

Table 7: Optimal results obtained for different values of θ.

θ	DR (%)	FPR (%)	Cost=1-DR+FPR
30	92.21	18.16	25.95
40	93.07	18.26	25.19
50	93.07	18.05	24.98
60	91.77	16.71	24.93

In the next section we propose a masquerade detection technique based on constraint programming.

2.6 Constraint Programming

The detection of a masquerader relies on a user signature, a sequence of commands collected from a legitimate user. The underlying assumption is that the signature captures detectable patterns in a user's sequence of commands. In this section, we propose a novel way of modeling user behavior and the detection of masqueraders. We model a user as a binary constraint network such that each node represents an episode of commands and binary relationship between a pair of episodes is encoded as the disjunction of the Allen's Interval relations. We describe the concept of Interval Algebra in the following section.

2.6.1 Interval Algebra

Allen in his landmark paper [1] propose *Interval Algebra* (IA), with 13 basic relations to relate any pair of time intervals in which events could occur. This initiated a substantial research activity in Artificial Intelligence front to devise practical systems, which reason about time. A time interval A is an ordered pair of real valued time points, represented as (A^-, A^+), such that $A^- < A^+$. A^- and A^+ are the start and end points of interval A. Given two time intervals, the relation between them can be represented by exactly one out

of 13 basic relations of IA. The set of all basic relations in IA is represented by, `I` = {`b`, `eq`, `m`, `o`, `d`, `s`, `f`, `bi`, `mi`, `oi`, `di`, `si`, `fi`}. These relations are namely, *before*(`b`), *equals*(`eq`), *meets*(`m`), *overlaps*(`o`), *during*(`d`), *starts*(`s`), *finishes*(`f`), *after*(`bi`), *met-by*(`mi`), *overlapped-by*(`oi`), *includes*(`di`), *started-by*(`si`), *finished-by*(`fi`). These relations are exhaustive and are pairwise disjoint. Figure 6, gives the semantics of these basic relations.

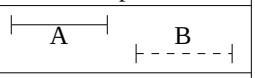
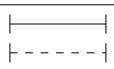
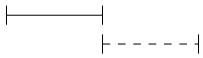
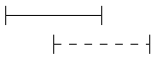
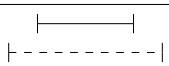
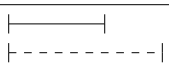
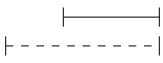
Relation	Symbol	Inverse	Example
A before B	b	bi	
A equals B	eq	eq	
A meets B	m	mi	
A overlaps B	o	oi	
A during B	d	di	
A starts B	s	si	
A finishes B	f	fi	

Figure 6: 13 basic relations in IA

When the relation between a pair of intervals is indefinite, it is expressed as disjunction of basic relations and is represented as a set. For example, the relation {`m`, `o`, `s`} between any two events A and B represents the disjunction

$$(A \texttt{ meets } B) \vee (A \texttt{ overlaps } B) \vee (A \texttt{ starts } B).$$

At any instance of time only one of the disjuncts holds true. There are $2^{13} = 8192$ possible ways to relate a pair of intervals which includes the empty and the universal set. An IA network is a graphical representation of this information where the vertices represent events and directed edges are labeled with sets of basic relations. The main reasoning tasks in this framework include, checking consistency of the given information and finding the feasible relations among all the variables in the network. The temporal information represented in terms of a collection of qualitative relations constrains time intervals and the reasoning tasks therefore reduce to the standard Constraint Satisfaction Problem (CSP). A CSP consists of a set of constraints over a set of variables, where each variable is associated with its domain

of values.

An IA network is a network of binary constraints where the variables represent time intervals, the domain of the variables are the end points of the variables and the binary constraints between variables are represented implicitly by the sets of basic relations.

Determining the feasible relations for example can be viewed as determining the deductive consequences of the given temporal information. For example, from the information, (A `meets` B), (B `meets` C), we could derive that (A `before` C). The main inference technique (path consistency) in this framework is based on constraint propagation.

Consider 3 intervals I, J, L with constraints ($I\ R_{ij}\ J$), ($J\ R_{jl}\ L$) and ($I\ R_{il}\ L$). Compute relational composition and intersect with the direct relation.

$$I\ R_{il}\ L = (I\ R_{ij}\ J \otimes J\ R_{jl}\ L) \cap I\ R_{il}\ L.$$

Continue until fixed relation. This path consistency algorithm is used as inference algorithm for Allen's Interval Algebra.

2.6.2 Proposed Method

We observe that while a user shows a consistent behavior over a long period of time, it may happen that due to some requirement of temporary nature, the same user may type in few different commands. In such situation, we get interleaving of command sequence i.e. during user's usual command sequence, there may be some other command sequence, arisen due to temporary requirement. Under such conditions, mere command sequence matching may not be very suitable technique to apply. We, therefore, propose to use interval algebra to capture interleaving of different command subsequences. We consider user command data as time series and apply various temporal relations, depicted in Figure 6, to find the relation among various command subsequences (which we call as *episodes*). Based on these relations we model the behavior of a user in terms of an IA network.

2.6.2.1 User modeling

We apply the episode discovery algorithm as described in Section 2.5.2 on the training data of each of the users $u_j \in U$, individually. Let E_j be the set of episodes for user u_j. We

select the most frequent N episodes (the set E) from all the episodes $(E_j, \forall j)$. These N most frequent episodes are used to profile the users as a binary constraint network. For each u_j we construct a directed graph G_j such that each of the N frequent episodes represents a node in G_j. For each $e \in E_j$ we find the interleaving of episodes from TR^{u_j} by using the 13 relations shown in Figure 6. The sets of relations among episodes that are satisfied in TR^{u_j} constitute the edges of the directed graph G_j. The edges of the graph determine the binary relationship between a pair of nodes encoded as the disjunction of the Allen's interval relations. Thus, for each u_j we have a directed graph G_j representing the user's normal behavior. We illustrate the above method by taking an example.

Example 2.6.1. Let a user's command sequence be (`ls pwd cd ls grep pwd cat ls cd grep`). Let us assume the following three episodes as frequent episodes.

(`pwd cd`), (`ls ls`) and (`ls grep`)

The interleaving of these episodes in the user's command sequence is shown in Figure 7. From Figure 7, we can see that the episode (`pwd cd`) occurs *during* the episode (`ls ls`) and also *after* (`ls ls`), thus generating the relations {`d`, `bi`}.

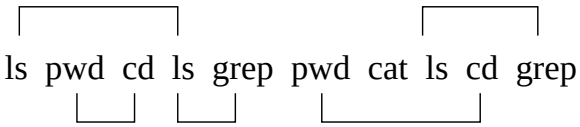


Figure 7: The interleaving of the episodes in user command sequence.

On the basis of interleaving shown in Figure 7, we profile the command behavior of the user as a directed graph as shown in Figure 8. □

2.6.2.2 Masquerade Detection

During test phase, given a command sequence b corresponding to user u_j, we form a directed graph G'_j using the episodes in E in the same manner as described above. Thus, G'_j captures the interleaving of episodes (of E) in b. Both G_j and G'_j have the same nodes, but the relation among the edges of G_j are derived from the normal behavior of the user, whereas that of G'_j is obtained from the new sequence b. G'_j is compared with the user's

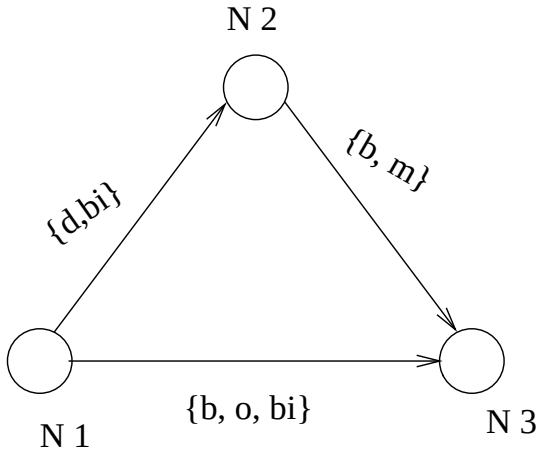


Figure 8: User's profile shown as a IA network, where each node corresponds to one episode, $N1$=(`pwd cd`), $N2$=(`ls ls`) and $N3$=(`ls grep`), and each edge denoted the set of constraints satisfied by corresponding nodes.

original profile represented by G_j for consistency. To do so, we compare the set of relations between corresponding edges of the two directed graphs. The following expression is used for comparison.

$$(I - edge_i(G_j)) \cap edge_i(G'_j) = \text{NULL}, \forall i \tag{56}$$

where $edge_i(G_j)$ represents the binary relations for the edge $edge_i$ in G_j. If Equation 56 holds (i.e. true), then the command sequence, b, belongs to user u_j. The intuition behind the above equation is that, if b is indeed coming from the genuine user, then it should also form the same directed graph and in such case the expression $I - edge_i(G_j)$ consists of relations not belonging to genuine user, whose intersection with incoming sequence, thus, gives information about the genuineness of b. If the above relation (Equation 56) does not hold then we go for *path consistency* check by using the Qualitative-Path-Consistency algorithm [31]. If G'_j is consistent with G_j, the incoming command sequence belongs to u_j otherwise, the incoming command sequence does not belong to u_j. As a special case, if b does not have at least two episodes of E, then we can not have any relations between the edges of G'_j. In such case, we flag b as masquerade sequence without comparing with G_j.

2.6.3 Experimental Results

We use the SEA dataset for experimentation. We apply episode discovery algorithm as described in Section 2.5.2 on the training data (first 5000 commands) of each user separately. In total, we find 9770 episodes from the training data of all users. Out of these 9770 episodes, we select those episodes that occur at least 1000 times in the training data, with the intuition that more frequent episodes can better represent the characteristics of all users. From this set of episodes, we discard those episodes which are multiples of some smaller episode. After this preprocessing, we get 20 ($= N$) most frequent episodes which are shown in Table 8.

Table 8: List of 20 episodes used for modeling users' normal behavior as IA network.

1.	netscape, netscape
2.	popper, popper
3.	sendmail, sendmail
4.	date, generic
5.	cat, cat
6.	ls, ls
7.	sh, sh
8.	tcpostio, tcpostio
9.	expr, expr
10.	uname, nawk
11.	egrep, egrep
12.	gcc, gcc
13.	generic, date, generic
14.	ls, sed
15.	sed, FIFO
16.	rdistd, tcsh
17.	rdistd, tcsh, rshd
18.	ls, sed, FIFO
19.	launchef, sh
20.	cpp, cc1

This set of 20 episodes is used for constructing each user's profile G_j, for $1 \leq j \leq 50$. For each block of test data of the user j, we construct G'_j and measure its consistency with G_j as per the proposed method in Section 2.6.2.

To measure the accuracy, we define the following measures of accuracy.

$$\mu_j = \frac{\text{\# of normal block detected as normal}}{\text{total \# of normal blocks}}$$
$$\lambda_j = \frac{\text{\# of masquerade block detected as masquerade}}{\text{total \# of masquerade blocks}} \tag{57}$$

The above expression (Equation 57) is calculated for each user u_j. Once it is calculated, the total accuracy of the method is calculated as follows.

$$TotalAccuracy = \frac{\sum_{j=1}^{U} (\mu_j + \lambda_j)}{2U} \tag{58}$$

It should be noted that the total accuracy given by the Equation 58 incorporates *true negatives* (μ_j) and *true positives* (λ_j) in the parlance of intrusion detection. We also observe that though the total number of profiled users is $U = 50$, some of them do not contain any masquerade blocks. Such users are excluded while calculating the value of total accuracy. Based on the accuracy measure, defined in Equation 58, we obtain an accuracy of 0.76 on the test data.

In the next section we propose an adaptive naive bayes approach for masquerade detection.

2.7 Adaptive Naive Bayes Approach

We observe from the existing methods that the performance of the algorithms are evaluated mostly on Schonlau data and on two models namely, *SEA configuration* and *1v49 configuration*. Both the configurations evaluate the performance based on the command blocks. For instance, the Schonlau data provides 50 blocks of 100 commands each for each user for the training data, and 100 blocks of 100 commands each for the test data. The masquerade data is generated by inserting an illegitimate block. Thus, in a sense the behaviour of the algorithm is tested in a very specific fashion that is guided by the nature of the standard dataset.

A legitimate user may deviate from his normal behaviour for a short duration. Thus, the behaviour of the masquerader cannot be identified instantaneously by examining one single

command. It is more appropriate to doubt a user who is consistently deviating from the normal signature rather than a user who deviates momentarily and returns to the normal pattern. We formalize this particular concept as *Deferred Detection*, which is described in the next section.

2.7.1 Deferred Detection

We propose to identify a block of commands into three categories- *legitimate*, *doubtful* and *masquerade*. If a block does not match with the corresponding user signature then we designate the block as *doubtful* and wait for subsequent blocks. If the *doubtful* status continues for 2 to 3 consecutive blocks then it is justified to conclude that the deviation is due to masquerading. Moreover, when one block is already *doubtful*, the subsequent block has to pass a more stringent legitimacy test. This is because once we know that a block of commands is a questionable block, the following block would have a higher likelihood of being *doubtful*.

Let b_0, b_1, b_2 be three consecutive blocks. Let us assume that b_0 is detected as *legitimate* and b_1 is detected as *masquerade* by any of the detection algorithms. When we test block b_2, it is justified to believe that b_2 is more likely to be a *masquerade*, and hence we put more stringent condition on b_2 to pass the legitimacy test. In case b_2 successfully passes the legitimacy test, then based on our knowledge that b_0 and b_2 are *legitimate*, b_1 is labeled as *legitimate* in spite of its failure to pass legitimacy test. This failure can be attributed to momentary deviation. Similarly, if b_2 is identified as masquerade block (it fails to pass the stringent legitimate test), then the block b_3 which immediately follows b_2, is subjected to still harder legitimacy test. In the context of Naive Bayes algorithm this condition can be implemented by changing the *threshold* progressively. One may argue that a block of 100 commands by itself is long enough and detecting a masquerade after two or three consecutive blocks (200 to 300 commands) can prove to be too late for online masquerade detection. But in the real life applications, the block size can be decided to be reasonably small depending on the context. In the case of real situation, if it is known that the smallest corrupted command string is of size k, then the block size can be chosen to be reasonably

smaller than k i.e. $k/4$ or $k/5$.

Motivated by this observation, we propose here a new algorithm for masquerade detection. We make use of Naive Bayes algorithm given in [75], and use a progressively varying threshold for consecutive masquerade blocks.

2.7.2 Adaptive Naive Bayes

We use Naive Bayes algorithm [75] to compute the self and nonself probabilities and find $Score(b, u_j)$ as given in Equation 59.

$$Score(b, u_j) = \frac{Self(b, u_j)}{Nonself(b, u_j)} \tag{59}$$

In traditional Naive Bayes method [75], a block b is determined to be *masquerade* or *legitimate* depending on whether $Score(b, u_j)$ falls below a threshold α or exceeds the threshold respectively.

In our Adaptive Naive Bayes algorithm, a block is detected as *legitimate*, *doubtful* or *masquerade* based on the corresponding status of two blocks preceding it. We determine the status of the current block by the following set of tests. It may be noted that we often look back to revise the status of some of the earlier blocks. Let b_0, b_1, b_2 be three consecutive blocks.

- If b_0 is *legitimate* then b_1 is marked *legitimate* only if $Score(b_1, u_j) \geq \alpha$, else, b_1 is marked *doubtful*.
- If b_0 is *legitimate* and b_1 is *doubtful*, then b_2 is marked *legitimate* if $Score(b_2, u_j) \geq \alpha + \frac{\alpha}{4}$, or else, it is marked *doubtful*.
- If b_2 is marked *legitimate* by the above test, then the marking of b_1 is revised to *legitimate*.
- If b_0 is *doubtful* and b_1 is *doubtful*, then b_2 is marked *legitimate* if $Score(b_2, u_j) \geq \alpha^{'} + \frac{\alpha^{'}}{4}$, where $\alpha^{'} = \alpha + \frac{\alpha}{4}$, or else, it is marked *masquerade*.
- If b_2 is marked *masquerade* by the above test then the markings of b_0 and b_1 are revised to *masquerade*.

- If b_2 is marked *legitimate* by the above test then the markings of b_0 and b_1 are revised to *legitimate*.

The above procedure is summarized in Figure 9. The theme of this method is shown in a tabular form in Table 9. It shows, the sequence observed (the first column), what action is taken by the algorithm (second column) and how the threshold changes (third column). 1 indicates a block which failed the legitimacy test condition, and 0 otherwise. "?", "N" and "M" indicate that the block is marked as doubtful, normal (legitimate) and masquerade respectively. Thus, based on these interpretations, the first row of the table (1, ?, $\alpha + \alpha/4$) means that, if the new block fails the test condition, then the algorithm marks it as doubtful ("?"), and increases the threshold by 25%. Similarly, the second row (? 0, N N, α) indicates that, if the current block passes the test condition, whereas its preceding block is labeled doubtful ("?"), then the Adaptive Naive Bayes algorithm marks both the blocks as normal (legitimate) and α is changed to its original value.

Table 9: Theme of the Adaptive Naive Bayes method

Sequence	**Action Taken**	**Threshold (α)**
1	?	$\alpha + \alpha/4$
? 0	N N	α
? 1	? ?	$\alpha' + \alpha'/4$
? ? 0	N N N	α
? ? 1	M M M	$\alpha' + \alpha'/4$

The proposed adaptive naive bayes method for masquerade detection is a joint work and part of this appears in the MTech thesis of KS Reddy [93]. In the current thesis, an extensive experimentation of the proposed method is done with all the three standard datasets for both SEA and 1v49 configurations.

2.7.3 Experimental Results

We experiment our algorithm with three existing standard datasets namely Schonlau dataset [99], Lane and Brodley dataset [64] and Greenberg dataset [45] which are described in Section 2.4. Experiments with these datasets are as follows.

1. **for** each block of commands b **do**
2. Calculate $Self(b, u_j)$ and $Nonself(b, U_j)$
3. Calculate $Score(b, u_j)$
4. **if** $Score(b, u_j) \leq \alpha^{'}$ **then**
5. **if** previous two blocks are *doubtful* **then**
6. Mark all the three blocks as *masquerade*
7. $\alpha^{'} = \alpha + \frac{9\alpha}{16}$
8. **else if** previous block is *masquerade* **then**
9. Mark block b as *masquerade*
10. $\alpha^{'} = \alpha + \frac{9\alpha}{16}$
11. **else**
12. Mark block b as *doubtful*
13. $\alpha^{'} = \alpha + \frac{\alpha}{4}$
14. **end if**
15. **else**
16. Mark block b and any previous *doubtful* blocks as *legitimate*
17. $\alpha^{'} = \alpha$
18. **endif**
19. **end for**

Figure 9: Adaptive Naive Bayes Classifier

2.7.3.1 Schonlau dataset

We conduct our experiments on the Schonlau dataset for both SEA and 1v49 configurations, with different threshold values. Figure 10 summarizes the experimental results as the ROC curve for SEA configuration. It can be seen from the ROC curve in Figure 10 that, except for two cases *POMPD* and *Naive Bayes updating*, points corresponding to all earlier experiments lie below the curve of the *Adaptive Naive Bayes Technique*. Even in these cases, the detection rate is much below 70% whereas the detection rate of our algorithm is above 80%. The best result that we obtained is with the detection rate of 83.9% with a false alarm rate of 8.8%.

We also test our algorithm taking block sizes of 100, 50 and 25 commands, results of

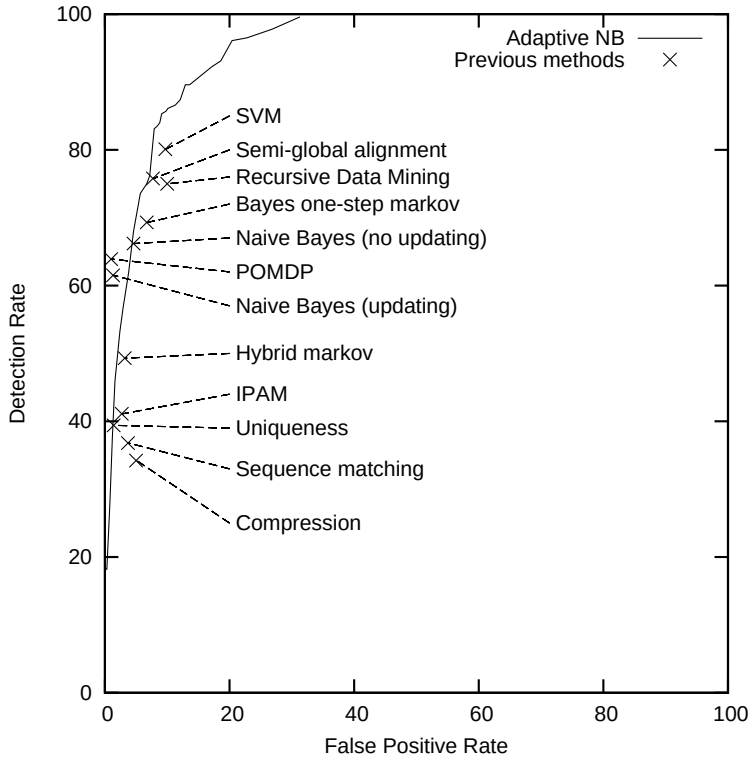


Figure 10: ROC curve of the Adaptive NB Algorithm and comparison with other algorithms

Table 10: Results for Adaptive Naive Bayes method taking block size of 100, 50 and 25

Test Configuration	Detection Rate	False Positive
SEA configuration with block size 100	84.0%	8.8%
SEA configuration with block size 50	85.3%	8.5%
SEA configuration with block size 25	85.2%	10.6%
1v49 configuration	90.7%	1.0%

which are depicted in Table 10. Figure 11 gives a comparison of the performances of our algorithm using various block sizes. One interesting observation is that although the block size is reduced to 50 and 25, the performance of the algorithm doesn't degrade much. In fact, the algorithm performs best when the block size is 50, as depicted in the figure. This means that a masquerader can be identified in just the time of 150 commands as opposed to 300 when using a block size of 100. When using a block size of 25, though the performance is not as good as when using a block size of 100 or 50, it can be observed from Figure 11 that the three curves are very close to each other. Thus, when using block size as 25, we can detect a masquerade in just 75 commands, which is quicker than what we can do

using other methods. We also experiment our algorithm with the 1v49 configuration. In

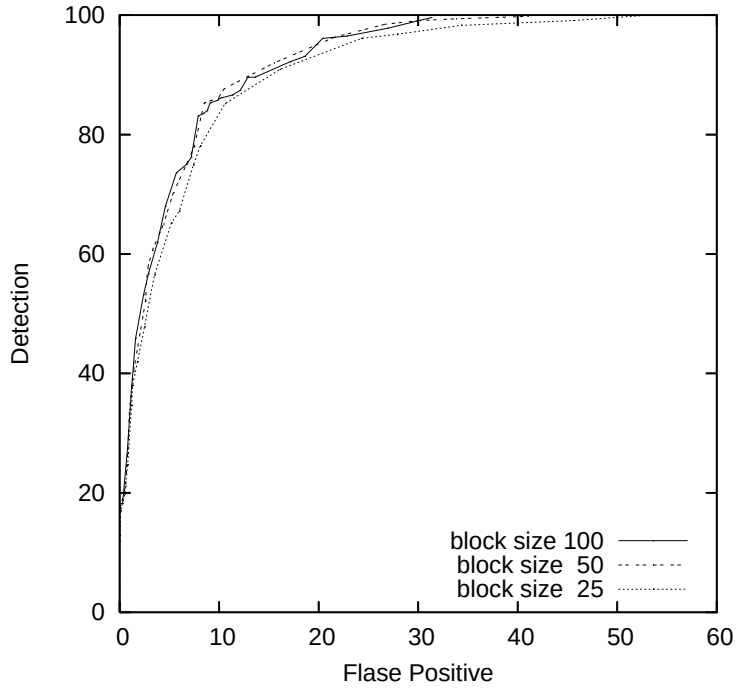


Figure 11: ROC curve of Adaptive Naive Bayes method with different block sizes

this configuration, the algorithm could detect masquerades at 90.7% accuracy with a false alarm of 1.0%.

2.7.3.2 Lane and Brodley dataset

As described in Section 2.4.2, this dataset contains session-wise normal data of 9 users. We preprocess the data by removing argument and session parameters, to get a continuous stream of commands. Table 11 shows a sample of the original and processed data.

As the dataset contains data from only nine users, we choose 1v8 configuration, a similar kind of approach as 1v49 configuration. Each user's own data is used for training and the other eight users' data are used for testing. As the number of commands available for each user are different, we have taken only first 5000 commands of each user for experimentation. We experimented both Naive Bayes and our proposed method, Adaptive Naive Bayes, on this processed dataset with three different block sizes of 100, 50 and 25. The results in terms of detection rate (DR) and false positive rate (FPR) are shown in Table 12.

Adaptive NB method performs better than Naive Bayes method consistently in all three

Table 11: A sample of the original and the processed data from the Lane dataset

Original data	Processed data
SOF	setenv
setenv	xhost
<2>	rlogin
xhost	finger
<1>	vi
rlogin	ls
<1>	vi
finger	
<1>	
EOF	
SOF	
vi	
<1>	
ls	
vi	

Table 12: Detection rate (DR) and False positive rate (FPR) by Naive Bayes and Adaptive NB methods on processed Lane and Brodley dataset for 1v8 configuration

Block size	Naive Bayes		Adaptive NB	
	DR	FPR	DR	FPR
100	100	1.78	100	0
	99.69	0	100	0
50	100	18.33	100	6.33
	97.38	0	99.78	0
25	100	31.61	100	14.11
	92.51	0	99.09	0

cases. When the blocks are smaller, the superiority of our method is more evident.

2.7.3.3 Greenberg dataset

As described in Section 2.4.3, Greenberg dataset consists of 168 users' data. We process the raw data to get clean data for each user, a sample of which is shown in Table 13. If *alias* is present for a command then the processed data includes both the command and its alias separated by <#>. The total of 168 users are divided into two sets as victim (V) and masquerade (M) by the method as given in Figure 12.

Coincidentally, the number of users in the victim set comes to be 50 which is same as the number of users in the Schonlau dataset. To have a greater diversity, all the remaining 118 users ($=|M|$) are taken as source of masquerade commands. The data for the victim

Table 13: Raw data and its corresponding processed form from the Greenberg dataset

Raw data	Processed data
C bye D /user/cpsc500/l01b91/xxxxxx A logout H NIL X M 09	bye<#>logout
C rwho | more D /user/cpsc500/l01b91/xxxxxx A NIL H NIL X NIL	rwho | more

1. **Initialize:** $V = \{\}$, $M = \{\}$
2. **for** $j = 1$ to 168
3. **if** (#commands(U_j) $\geq$ 2000 and #commands(U_j) $<$ 5000) **then**
4. $V = V \cup \{U_j\}$
5. **else**
6. $M = M \cup \{U_j\}$
7. **end if**
8. **end for**

Figure 12: Algorithm used to obtain victim and masquerade set of users from Greenberg dataset

users are truncated to 2000 commands out of which 800 commands are used for training purpose. The remaining 1200 commands are divided into 40 blocks of 30 commands each. 10 blocks of masquerade data is inserted into the test data, for which, 10 places are selected randomly from the possible 41 places. For a victim, if same place is selected more than once then it indicates consecutive occurrence of masquerade blocks. Occurrence of consecutive masquerade blocks is represented by selecting a continuous stream of commands from the same masquerade user. Masquerade blocks are randomly selected from the command pool of M and inserted at the selected places. So, at the end the test data consists of 1500 commands logically grouped into 50 blocks.

As our method delays the declaration of masquerade by 3 blocks, we take the block size to be 10 (1/3rd of the original block size) during the testing phase. So the new scenario is, the test data consisits of 150 small blocks out of which 30 are masquerades. Our algorithm

achieves a hit rate of 84.13% with a false positive of 9.4%. We have also experimented our algorithm with 1v49 configuration on this dataset which gives a detection rate of 97.38% with a false positive of 0%.

2.8 Conclusions

In this chapter we formulate the problem of Masquerade attack and propose three new algorithms for masquerade detection.

The episode based detection method emphasizes on finding meaningful and frequent episodes in a block and based on the legitimacy of the episodes, terms the block to be normal or masquerade. A valid user may sometime deviate from its normal usage pattern. Any small deviation in the command sequence is localized to episodes and thus its effect to the legitimacy of the entire block is reduced. This results in an improved detection rate and reduced false alarm. As indicated by our experimental results, our algorithm performs better than many standard methods. Moreover, our method is more realistic in the sense that it does not generate false alarm for any momentary deviations.

We also investigate the applicability of Interval Algebra (IA) in the scenario of masquerade detection. We observe that user's command line data has some variability from time to time for a short period of time. To capture the interleaving of various command sequences, we make use of 13 temporal relations and represent the user's profile as a graph. Each new incoming command sequence is converted into a similar graph and is compared with the corresponding user's profiled graph. If the new graph is not consistent with the profiled graph, we flag the new command sequence as masquerade. Though the performance of the method is not good in comparison to other methods, but the main contribution here is the novel use of constraint network for intrusion detection problem.

We also propose a variation of the classical Naive Bayes approach using a deferral model. Our method uses a deferred detection technique which delays the decision of a block by a few blocks. Our method is simple to implement and also realistic to the case of momentary deviations, which may be due to user concept drift. Even though the method defers the decision to later blocks, a masquerader can be detected in reasonably quick time. We have

demonstrated that the method can detect masqueraders fairly quickly by choosing a smaller block size. We provide extensive experimental results on all the three standard datasets and show that the method performs better than most of the methods available in the literature.

CHAPTER III

APPLICATION OF LOCALLY LINEAR EMBEDDING TO ABNORMAL PROCESS DETECTION

3.1 Introduction

There are many different levels on which an IDS can monitor system behavior. It is critical to monitor and profile behavior at a level that is both robust to variations in normal and perturbed by intrusions. In the work reported in this chapter, we chose to monitor behavior at the level of processes.

The range of behaviors of processes is limited compared to the range of behaviors of users. Processes usually perform a specific, limited function, whereas users can carry out a wide variety of actions. The behavior of processes is relatively more stable over time, especially compared to user behavior. Not only do users perform a wider variety of actions, but the actions performed may change considerably over time, whereas the actions (or at least functions) of processes usually do not vary much with time. For host based intrusion detection system (HIDS), system call sequences have been acknowledged as the most effective set of data that can capture the behavior of a process [39].

System calls are functions, which a process calls to invoke different operating system routines/services. When a program executes, depending upon its use of operating system services, the corresponding process generates a *trace* - an ordered list of system calls. Such trace can be collected by logging the system calls using operating system utilities, e.g. Linux *strace* or Solaris *Basic Security Module* (BSM). *truss* is another such tool which is available for both the Linux and Solaris platforms and can do the same thing. By monitoring the trace, much can be learned about the behavior of the process being monitored.

In HIDS, it is assumed that any normal execution of a process follows a pattern and hence the normal behavior of a process can be profiled by a set of predictable sequences of

system calls. Any deviation from these sequences of system calls is termed as intrusion in the framework of anomaly based IDS. Thus, the problem of identifying intrusive processes, which are measurably different from the normal behavior, becomes detecting anomalous sequence of system calls. Thus, it can be viewed as a supervised classification problem, where given a new sequence of system calls, the task is to find whether it is anomalous or normal.

Anomaly detection based systems face a problem of getting normal data for profiling the normal behavior. Normally the data contains some background activities, termed as noise, which sometimes degrades the overall performance of the system. Also, to detect the presence of attack, a good amount of knowledge is required. In other words, a lot more data need to be processed so that the attack can be manifested in this. But the larger the amount of data to be processed the larger will be the cost in terms of time and space associated with it, which delay the timely detection of attack. Thus, it is practically desirable to process as less data as possible which adequately captures all information required for the manifestation of the attack. In the present work, we focus on preprocessing the data to be analyzed for detection and investigate a method of reducing the dimensionality of the data without degrading the performance of the system by showing rigorous experimental analysis. We make use of Locally Linear Embedding (LLE), a nonlinear dimensionality reduction technique, to reduce the size of the input data derived from the sequence of system calls.

The rest of the chapter is organized as follows. In Section 3.2, we describe different ways of representation of system call sequences for intrusion detection. Section 3.3 gives a brief overview of research on IDS using these representations. Section 3.4 covers the datasets which are used for system call based research. Dimensionality reduction and in particular Locally Linear Embedding (LLE) technique is described in Section 3.5. In Section 3.6, we use LLE for intrusion detection. The experimental setup and results are shown in Section 3.7 followed by conclusions in Section 3.8.

3.2 Representation of Sequences

The length of the system call trace generated by processes varies from process to process. It can range from a few system calls to tens of thousands of system calls. Research on HIDS based on system call data is concerned with two types of representation of system calls. One is *term frequency* representation, which is based on frequency of system calls and the other is *decision table* representation, which is based on subsequences of system calls. The aim is to represent the sequences in a form suitable for different classification algorithms.

Let $S = \{S_1, S_2, \ldots, S_m\}$ (say, $|S| = m$) be the set of system calls made by all training processes. Let P be the set of training processes. The ℓ^{th} process P_ℓ is represented as $< P_1^\ell$, P_2^ℓ, ..., $P_n^\ell >$, where P_j^ℓ is the j^{th} system call in P_ℓ.

In the following sections we describe both the term frequency and decision table representations of system calls.

3.2.1 Term Frequency Representation

Term frequency representation draws an analogy between text categorization and intrusion detection, such that each system call is treated as a word and a set of system calls generated by a process as a document. In this approach, a process, P_ℓ, is represented by an ordered list $< c_1^\ell, c_2^\ell, \ldots, c_m^\ell >$ where, c_j^ℓ denotes the frequency of system call S_j in the process P_ℓ.

Example 3.2.1. Let S = {`access audit chdir close creat exit fork ioctl`}. Let two processes be P_1 = <`access close ioctl access exit`>, and P_2 =<`ioctl audit chdir chdir access`>. We can represent the two processes in terms of frequencies as, P_1 = <2, 0, 0, 1, 0, 1, 0, 1> and P_2 = <1, 1, 2, 0, 0, 0, 0, 1>. □

This representation formulates each process as a vector of m-dimension (in Example 3.2.1 m=7). But, it is to be seen that, this form of representation of a process does not preserve the relative order of system calls in the sequence. We also have to know all the system calls of a process and their frequencies in order to represent it in this form. In other words, we have to wait till the termination of the process to represent it. This approach is not suitable for on-line detection as the frequency cannot be ascertained until after the process terminates [122].

3.2.2 Decision Table Representation

In supervised classification, the posteriori knowledge is expressed by one distinguished attribute called decision attribute. A table wherein one of the attributes is decision attribute is called a *decision table*. Every other column represents an attribute that can be measured for each object. A decision table is said to be *consistent* if each unique row has only one value of decision attribute. We describe here the representation of system call data in the form of a decision table.

Using a sliding window of size D ($D \leq n$), subsequences of length D are extracted from each of the training processes. Thus, the i^{th} subsequence of P_ℓ is given by,

$$< P_i^\ell, P_{i+1}^\ell, \ldots, P_{i+D-1}^\ell >.$$

Each such subsequence of a normal process is labeled as normal. In case of abnormal process, it is found that not all parts are responsible for intrusion. Hence, not all of the subsequences of an abnormal process are abnormal and many of them will be identical to those occurred in normal processes. Thus, a subsequence corresponding to an abnormal process matching with any of the normal subsequences is discarded; otherwise it is labeled as abnormal. It should be noted that by removing duplicate subsequences, we get a consistent decision table because no subsequence can belong to normal as well as abnormal classes.

Example 3.2.2. Let P_1 and P_2 be normal and abnormal processes respectively.
P_1 = <`fcntl, close, close, fcntl, close, fcntl, close, open`> and
P_2 = <`fcntl, close, fcntl, close, open, open`>
We transform P_1 into a set of subsequences using a sliding window of length 5 (=D). We get 4 subsequences from P_1 and label each of them as normal. While calculating the subsequences of P_2, the first subsequence <`fcntl, close, fcntl, close, open`> matches with the last subsequence of P_1 and therefore it is discarded. The second subsequence <`close, fcntl, close, open, open`> is labeled as abnormal and added to the decision table. The final decision table is shown in Table 14. It can be seen that the decision table, thus created, is consistent. □

Table 14: Representation of subsequences

Objects	A_1	A_2	A_3	A_4	A_5	Decision
1	fcntl	close	close	fcntl	close	normal
2	close	close	fcntl	close	fcntl	normal
3	close	fcntl	close	fcntl	close	normal
4	fcntl	close	fcntl	close	open	normal
5	close	fcntl	close	open	open	abnormal

3.3 Research on IDS using System Calls

Many researchers have proposed different methods of anomaly process detection by profiling the system call sequences in either of the two ways of representation. Table 15 shows different detection methods for each of the representations. In the subsequent sections we describe each of these methods.

Table 15: Detection methods for different representations

Sequence Representation	Method
Decision Table	tide stide DFA based scheme RIPPER t-stide Wespi scheme Rough set based scheme
Term Frequency	k-NN based scheme BWC scheme Kernel based scheme SVD based scheme

3.3.1 tide

Inspired by the natural immune systems, Forrest et al. [39] propose a behavior (anomaly) based intrusion detection system based on monitoring the system calls invoked by processes. In this approach, called *time-delay embedding* (tide), the profile for normal behavior is built by extracting short sequences (also called subsequences) of system calls from process trace. It uses a sliding window algorithm to populate a *table* with the positional relationships between system calls. The term *table* refers to a compressed representation of the normal

profile of an application where the current system call is used to index the table and there are at most N rows in the table, where N is the number of unique system calls used by the application being profiled. If during the creation of the table, two explicit windows with the same current system calls are encountered, then the rows are collapsed into one (the rows corresponding to `open` and `mmap` system calls in Table 16 are the examples). In other words, the normal profile consists of a list for each system call with the system calls that follow it at offset 1,2 up to k.

Forrest et al. [39] use a sliding window of size $k+1$ to record which system calls succeed or precede each other at offsets 1 through k. This implementation is said to have a "forward" lookahead because while matching with a test process, the current system call is used as the index to the table and anomalies are found by performing a pair wise comparison between the current system call and each system call that follows at offsets 1 through k. If the test process has a matching system call in the table at each offset, then it is considered as matching; otherwise it is a mismatch. The number of mismatches are recorded as a percentage of total number of mismatches. For a sequence of length L and lookahead of k, there are $(L-k)$ subsequences each with k positions to match. Thus, the total mismatches for subsequences with k positions are $k(L-k)$. Also there are $k-1$ subsequences (of length $k-1, k-2, \ldots, 1$ each) in the table. Therefore, the maximum number of pairwise mismatches for a sequence of length L with a lookahead of k is:

$$k(L-k) + (k-1) + (k-2) + \ldots + 1 = k(L - \frac{(k+1)}{2}) \tag{60}$$

If the number of mismatches for a test process crosses a predefined threshold, the process is flagged as anomalous. The following example illustrates the approach.

Example 3.3.1. Let us consider the following trace to profile the normal behavior with $k = 3$.

`open`, `read`, `mmap`, `mmap`, `open`, `getrlimit`, `mmap`, `close`

Sliding window of size 4 yields five subsequences each of length 4, and three subsequences of length 3, 2 and 1 each. The forward lookahead table representation is shown in Table 16 which comprises the normal behavior. Let the following sequence of system calls be the

Table 16: Table representing the normal profile

current call	offset 1	offset 2	offset 3
open	read getrlimit	mmap	mmap close
read	mmap	mmap	open
mmap	mmap open close	open getrlimit	getrlimit mmap
getrlimit	mmap	close	
close			

test trace.

`open`, `read`, `mmap`, `open`, `open`, `getrlimit`, `mmap`, `close`.

Subsequences are extracted from the test sequence by sliding a window of same size and are matched with the subsequences in the table. While comparing, the test sequence generates four mismatches and the corresponding subsequences are:

`open`, `read`, `mmap`, *`open`*

`read`, `mmap`, *`open`*, `open`

`open`, *`open`*, *`getrlimit`*, `mmap`

In the above three subsequences, the mismatch offsets are indicated by *italic* system calls. Using Equation 60, total number of mismatches are 18. Therefore, the percentage of mismatch is 22%. □

3.3.2 stide

Inspired by the above mentioned work, *tide* approach is extended by Hofmeyr et al. [50] by using a technique called *sequence time-delay embedding* (stide). In this approach, the normal profile consists of fixed-length, unique subsequences extracted from the process trace. Thus, for a sequence of length L and a window of size k, the normal profile can consists of at most $(L-k+1)$ subsequences. This normal profile is then used to monitor the ongoing behavior of the process during test phase. It extracts all the overlapping subsequences of length k from the new trace and matches them against the subsequences present in the normal profile, by calculating Hamming distance between the two subsequences. Thus, for each

subsequence s_i in the new trace S_{new}, it computes minimal Hamming distance $D_{min}(s_i)$ as follows.

$$D_{min}(s_i) = \min\{D(s_i, s_j) \ \ \forall \ \text{ normal subsequences } s_j\}. \tag{61}$$

Where, $D(s_i, s_j)$ is the Hamming distance between the two subsequences s_i, s_j which is the number of positions in which the two subsequences differ. D_{min} value represents the strength of the anomalous signal and the higher the value the more likely it is intrusive. Thus, for S_{new}, the anomalous signal strength is calculated as,

$$\hat{S}_{new} = (\max\{D_{min}(s_i) \ \ \forall \ \text{ subsequences } s_i \text{ of } S_{new}\})/k. \tag{62}$$

If $\hat{S}_{new} \geq \tau$, where $0 < \tau \leq 1$, then S_{new} is classified as anomalous else normal. For experimentation Hofmeyr et al. [50] traced `sendmail`, `lpr`, `ftp` programs. It has been observed that for $k < 6$, `decode` intrusion is not detected, which supports the observation made by Hofmeyr et al. about minimal sequence length of 6.

3.3.3 DFA based scheme

Kosoresow et al. [61] suggest the use of deterministic finite automata (DFA) to capture the temporal ordering of events in a sequence of system calls. They demonstrate that a program structure can be learnt by constructing DFA using *macros*, which are variable-length patterns of system calls. Repeated patterns of system calls are represented efficiently and compactly using DFA. The procedure followed by Kosoresow et al. for construction of DFA for a process is as follows. First, each process is divided into three parts: a prefix, a main portion, and a suffix. For each such part, the longest common prefix and suffix is found. Then, frequently occurring system calls and common short patterns of system calls in the main portion are derived. These iteratively formed strings of system calls are formed into macros. For `sendmail` program, a total of 24 macros are calculated by Kosoresow et al. Each process is then reduced to a new string made of these macros. Using these macros, the behavior of a process is represented by constructing DFA. When applied on `sendmail`, 147 processes are reduced to 26 different process description. Each new process is matched with these macros, using the *locality frame count*, defined in [50]. The process is declared

as normal or abnormal on the basis of some threshold. The main advantage of this scheme is that this has resulted in shorter normal profile. But identifying all the possible states, to be used in DFA, is very difficult and time consuming. If a sequence enters a state, not represented in DFA, the DFA can not move further. In other words, DFA is not flexible enough to capture all the states. If, however, all the states are incorporated in DFA, then it becomes too general to identify anomalous behavior.

3.3.4 RIPPER

The above approaches use sequential string matching in one or the other forms. Though these are simple to understand and easier to implement, these approaches may be time consuming in case of a very big data set. Lee et al. [67] investigate the idea of applying rule-based approach to find the rules for normal and abnormal classes. For their approach, Lee et al. make use of a rule learner called Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [24] to form the rules for classification. RIPPER produces rules for the 'minority' class which, in this case, is abnormal class. A default rule is produced which classifies rest of the data as belonging to normal class. They derive derive the training data from UNM `sendmail` traces. To detect whether a new trace S_{new} is normal or not, it extracts all the subsequences in a similar way and based on learned RIPPER rules classify them as "normal" or "abnormal". It uses a sliding window of size $2k+1$ and a sliding step of k to scan the predictions made by RIPPER. Within this region of length $2k+1$, if more than k predictions are abnormal, then the current region is predicted as "abnormal" region. If the percentage of abnormal regions in S_{new} is above a threshold, then S_{new} is classified as an intrusion.

In another set of experiments, Lee et al. learn to predict the n^{th} system call after seeing the first $(n-1)$ system calls. This approach tries to learn the co-occurrence of system calls under normal execution of a process. RIPPER outputs the rules with a confidence value associated with them. The confidence value is calculated as the number of matched examples divided by the sum of matched and unmatched examples. Whenever a rule is violated, its score is incremented by 100 times its confidence. The average score of entire

trace is used to decide whether an intrusion has occurred.

3.3.5 t-stide

A simple addition to *stide*, called *stide with frequency threshold* (t-stide) is proposed to test the premise that rare sequences are suspicious [122]. The normal profile is built in the same way as in *stide*, but in addition to the subsequences, their frequency of occurrence in training data is also recorded. Subsequences from test trace are compared to those in the normal profile, just like *stide*. Rare sequences, as well as those not included in the database, are counted as mismatches. It introduces the concept of *Locality Frame Count* (LFC) to classify the test trace. At each point in the test trace, it checks whether the current subsequence is a mismatch, and keep track of how many of the last n subsequences are mismatches. This number, called LFC, conveys the anomaly strength. A threshold is set on the LFC, below which the test trace is considered as normal, otherwise it is classified as abnormal.

3.3.6 Wespi scheme

Most of the above mentioned approaches model the behavior of processes in terms of fixed-length contiguous subsequences of system calls. One potential drawback of this approach is that the size of the database that contains fixed-length contiguous subsequences increases exponentially with the length of the subsequences. Wespi et al. [123] observe that there are many cases in which very long sequences are repeated frequently and also most of the sequences begin and end with similar subsequences. Thus, they propose a detection technique based on variable length subsequences. Wespi et al. use Teiresias algorithm [95] to derive *maximal variable-length* patterns from training data. A pattern p is said to be *maximal* if there is no other pattern q that contains p as a subsequence and has same number of occurrences as p.

Let there be a set of strings $S = s_1, s_2, \ldots, s_n$ over the alphabet $\sum = c_1, c_2, \ldots, c_n$. A character $c \in s$ is said to be covered by the pattern p if $c \in p$ and p is a substring of s. A string s is said to be covered by a set of patterns P if for each character $c, c \in s$, there is a pattern $p, p \in P$, such that c is covered by p.

The function $bCover(p, s)$ returns the number of characters covered at the beginning and end of a sequence s by a pattern p. The boundary coverage of a pattern p and a string set $S = s_1, s_2, \ldots, s_n$, written as $bCover(p, S)$, is defined as

$$bCover(p, S) = \sum_{i=1}^{n} bCover(p, s_i) \tag{63}$$

An algorithm is devised to further reduce the set P of maximal variable-length sequences. The reduced pattern set R is constructed as follows. μ denotes the minimal pattern length that is used to generate the set of maximal variable-length patterns.

1. If $P = \phi$, then add all $s \in S$ to R and exit.
2. For each $p \in P$ calculate $bCover(p, S)$.
3. Select a pattern $r \in P$ for which $bCover(r, S)$ is maximal, i.e. there is no other pattern $p \in P$ for which holds:
   - $bCover(p, S) > bCover(r, S)$ or,
   - $bCover(p, S) = bCover(r, S) \wedge |p| > |r|$.
4. Add r to R and remove it from P.
5. Remove all matching substrings adjacent to the beginning or end of a string, i.e., remove strings of the form $s = r^+$, and replace strings of the form $s = r^+s'$, $|s'| > \mu$ or $s = s''r^+$, $|s''| > \mu$, with s' or s'', respectively.
6. Remove the matching substrings that are not adjacent to the beginning or end of a string, i.e., as long as there is an $s \in S$, $s = s'rs''$, $|s'| \geq \mu$, $|s''| \geq \mu$, replace s with the two strings s' and s''.
7. If there is an $s \in S$ with length $|s| < 2 \times \mu$, remove s from the set of strings S and add it to the pattern set P.
8. If $S = \phi$, go to Step 1, otherwise exit.

In the pattern matching phase, The incoming stream is matched with the patterns in R. Whenever the number of unmatched system calls crosses a threshold, an alarm is raised. The pattern matching is done as follows:

1. Set the look-ahead parameter to a value $\delta > 0$, and set the threshold for the number of consecutive uncovered characters to a value $\tau > 0$.

2. Set the counter of consecutive uncovered characters, κ, to 0.

3. When there is a sufficient number of characters in the input stream I, find a pattern $p \in R$ that covers the beginning of the input stream I. If no pattern can be found, go to Step 6.

4. Find patterns $q_1, q_2, \ldots, q_\delta$, $(\delta > 0)$ such that the string $t = pq_1q_2 \ldots q_\delta$ covers the beginning of the stream. If there are ε patterns $q_1, q_2, \ldots, q_\varepsilon$, $0 < \varepsilon < \delta$, that cover the entire input sequence, set $t = pq_1q_2 \ldots q_\varepsilon$.

   - If t matches the entire input sequence, remove it and go to Step 2.
   - If δ patterns can be found that cover the beginning of the input stream, remove pattern p from the input stream, and go to Step 2.

5. Determine all pattern combinations that match the beginning of the input stream. If there is a match, select the pattern combination that covers the longest input sequence, remove it from the input stream, and go to Step 2.

6. Skip one character, and increase κ by 1.

7. If $\kappa = \tau + 1$, raise an alarm.

8. Go to Step 2.

Using this approach, the size of database of normal pattern is reduced by almost 10 times. Wespi et al. show experimental results on `ftpd` data and compare with *stide* approach.

3.3.7 Rough set based scheme

Rawat et al. [89] propose a rough set based approach for detection of abnormal activities in processes. They extract subsequences of fixed length from program traces and represent them in a decision table labeling the subsequences as normal if it is from a normal process, and abnormal, if it is from an abnormal process and not present in a normal process. They derive IF-THEN decision rules from the decision table using LEM2 algorithm [46]. For the example decision table given in Table 14, the IF-THEN rules produced by LEM2 algorithm is shown in Table 17.

Table 17: Representation of IF-THEN rules

1	$(A_2 = \texttt{close}) \Rightarrow$ (Dec=normal)
2	$(A_1 = \texttt{close}) \wedge (A_2 = \texttt{fcntl}) \wedge (A_3 = \texttt{close}) \wedge (A_4 = \texttt{fcntl}) \Rightarrow$ (Dec=normal)
3	$(A_1 = \texttt{close}) \wedge (A_2 = \texttt{fcntl}) \wedge (A_3 = \texttt{close}) \wedge (A_4 = \texttt{open}) \Rightarrow$ (Dec=abnormal)

Rawat et al. further introduce a default rule which says "any subsequence which is not covered by any of the rule is abnormal". With the inclusion of the default rule, the final rule set represents the case of a hybrid IDS because it has rules for normal and abnormal processes i.e. misuse-based system and rule for anything which deviates from learnt behavior i.e anomaly-based system. These learned rules are used to classify new process. Subsequences from a new (test) process are classified based on these learned rules. If at least one of the subsequences is identified as abnormal then the whole process is tagged as abnormal.

3.3.8 k-NN based scheme

An approach based on *k-NN classifier* is proposed by Liao and Vemuri [69] where the frequencies of system calls used by a program (process), rather than their temporal ordering, are used to define program's behavior. Their approach draws an analogy between text categorization and intrusion detection, such that each system call is treated as a word and the set of system calls generated by a process as a document. The processes under normal execution are collected from the DARPA data and thereafter converted into vectors, consisting of frequencies of system calls made by the processes during the normal execution.

Let S (say, $|S| = m$) be the set of system calls made by all the processes under normal execution. From all the normal processes a matrix $A = [a_{ij}]$ is formed, where a_{ij} denotes the frequency of i^{th} system call in the j^{th} process. In order to categorize a new process P_{new} into either normal or abnormal class, P_{new} is first converted into a vector. The *k-NN classifier* then compares it with all the processes A_j in A to determine the k nearest neighbors, by calculating the cosine similarity, $CosSim(P_{new}, A_j)$, as given in Equation 64.

$$CosSim(P_{new}, A_j) = \frac{P_{new} \cdot A_j}{||P_{new}|| \cdot ||A_j||} \tag{64}$$

The average similarity value of the k nearest neighbors is calculated and a threshold is set. When the average similarity value is above the threshold, process P_{new} is considered as normal; otherwise abnormal.

3.3.9 BWC scheme

Rawat et al. [90] observe that the cosine similarity used by Liao et al. [69] does not accord weight to processes having more number of common system calls. And, since the cosine similarity measure considers only the frequencies of the system calls, it may sometimes produce erroneous results.

As an improvement to the k-NN based method by Liao et al. [69], Rawat et al. [90] propose a new similarity metric that significantly improves the detection results. They measure the similarity between two processes using a metric that considers two factors - occurrence of system calls shared by the two said processes and the frequency of system calls in the processes and thus term this similarity metric as Binary Weighted Cosine (BWC) metric. For this metric, each process is represented in two forms, as a vector with frequency of system calls and a binary vector having information about the presence or absence of system calls. The BWC metric between two processes P_i and P_j is defined as:

$$BWC(P_i, P_j) = \mu(Pb_i, Pb_j) \cdot CosSim(P_i, P_j) \tag{65}$$

where, Pb_i and Pb_j are the binary vector representation of corresponding processes P_i and P_j, and $\mu(Pb_i, Pb_j)$ is a similarity score between the binary vectors, defined as:

$$\mu(Pb_i, Pb_j) = \frac{\sum_{n=1}^{m}(Pb_i \wedge Pb_j)_n}{\sum_{n=1}^{m}(Pb_i \vee Pb_j)_n} \tag{66}$$

where the summation runs over n, which is a subscript on the elements of the processes Pb_i and Pb_j and m is the length of each process vector. The symbols $\wedge$ and $\vee$ are the bitwise AND and OR operators.

The motive behind multiplying μ and $CosSim$ is that $CosSim(P_i, P_j)$ measures the similarity based on the frequency and $\mu(Pb_i, Pb_j)$ is the weight, which tunes the similarity score $CosSim(P_i, P_j)$ according to the number of similar and dissimilar system calls between the two processes.

3.3.10 Kernel based scheme

Sharma et al. [105] introduce different kernel based similarity measures for intrusion detection based on system call sequences using text processing technique (term frequency approach). They use the same detection algorithm as that of Liao et al. [69] and Rawat et al. [90], but apply four different kernel based similarity measures for finding the similarity between processes. The four kernel based similarity measures used are radial basis function (RBF), smooth RBF (SRBF), binary weighted RBF (BWRBF), smooth BWRBF (SBWRBF) as shown in Equations 67–70.

$$\text{RBF}(P_{new}, A_j) = \exp(-\frac{1}{2} \parallel P_{new} - A_j \parallel^2 / \parallel A \parallel^2) \tag{67}$$

$$\text{SRBF}(P_{new}, A_j) = \exp(-\frac{1}{2} \parallel P_{new} - A_j \parallel^2 / \parallel A \parallel) \tag{68}$$

$$\text{BWRBF}(P_{new}, A_j) = \mu(Pb_{new}, Ab_j) * \exp(-\frac{1}{2} \parallel P_{new} - A_j \parallel^2 / \parallel A \parallel^2) \tag{69}$$

$$\text{SBWRBF}(P_{new}, A_j) = \mu(Pb_{new}, Ab_j) * \exp(-\frac{1}{2} \parallel P_{new} - A_j \parallel^2 / \parallel A \parallel) \tag{70}$$

Where $\mu(Pb_{new}, Ab_j)$ is same as defined in Equation 66. Sharma et al. [105] show that by using a binary weighted kernel similarity measure the detection accuracy is improved.

3.3.11 SVD based scheme

In order to make an IDS fast enough to detect an attack early, Rawat et al. [91] investigate the applicability of a dimensionality reduction technique as a preprocessing step. In this study, each process in the training data is represented as a vector of fixed size as per term frequency representation. Let $A = [p_{ij}]$ be the matrix such that each element p_{ij} in A

represents the frequency of i^{th} system call in j^{th} process, $i = 1 \ldots m$ and $j = 1 \ldots n$. Thus each process is represented as a vector in m-dimensional space. Then by Singular Value Decomposition (SVD) A can be represented as,

$$A = T_0 S_0 D_0' \tag{71}$$

where T_0 and D_0 are the matrices of left and right singular vectors and S_0 is the diagonal matrix of singular values (square roots of eigenvalues of $A^T A$ or AA^T) in decreasing order of A. To reduce the processes into a lower dimension l, the first l largest values of S_0 are selected and accordingly the size of matrices T_0 and D_0 are adjusted as follows.

$$A_l = T_l S_l D_l' \tag{72}$$

where T_l is $m \times l$ matrix, S_l is $l \times l$ matrix and D_l is $l \times n$ matrix. Any new test process q is similarly represented as a row vector of size $1 \times l$. This new vector is mapped into the new space as follows:

$$\hat{q} = q T_l S_l^{-1} \tag{73}$$

This transformed query is compared against the columns of A_l to find most similar processes using cosine similarity metric and classification is done using $k - NN$ method. Empirically it is shown that by reducing the dimension of the input data, a better and fast detection is achieved.

3.4 Datasets

Most of the methods discussed above are based on learning techniques from various disciplines like data mining, machine learning etc. It is well known that a learning algorithm is as good as the data used for training. A good data set not only facilitates the researchers to evaluate their proposals, but also provides a common platform to compare their work with others. In this section, we discuss the datasets that are widely used for the study of system call based anomaly detection.

3.4.1 UNM Dataset

Forrest et al. [39] first proposed the idea of system call tracing for program behavior. They established the analogy between human immune system and intrusion detection system. In order to test the proposed idea, the team led by Forrest collected the system call data for various privileged unix programs [114]. Some of the data collected are synthetic and others are live. Synthetic traces are collected by running a prepared script, solely for the purpose of exercising the program and not to meet any real user's requests. Live normal data are traces of programs collected during normal usage of a computer system. Each trace is a list of system calls issued by a single process from the beginning of its execution to the end and is given in a separate file. Each trace file lists pairs of numbers, one pair per line. The first number in a pair is the PID of the executing process, and the second is a number representing the system call. The mapping between system call numbers and actual system call names is given in a separate file. However, using this mapping limits the use of *any program-any attack* combination. One has to use the corresponding attack files only. The system call traces are available for the following programs:
`synthetic sendmail`, `synthetic ftp`, `synthetic lpr`, `live lpr`, `xlock`, `live named`, `login and ps`, `inetd`, `sendmail`.

3.4.2 DARPA Dataset

MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) sponsorship, has collected and distributed the first standard corpora for evaluation of computer intrusion detection systems [28]. To date, there are three data sets available for evaluation, DARPA'98, DARPA'99 and DARPA'2000. The approach to create datasets is to simulate normal and attack traffic on a private network using real hosts, live attacks and live background traffic. The corpus was collected from a simulation network that was used to automatically generate realistic traffic including attempted attacks. The attacks that are included can broadly be classified into the following categories - *Probing* (information gathering attacks), *Denial-of-Service* (DoS, deny legitimate requests to a system), *User-to-Root* (U2R, unauthorized access to local super-user or

root), and *Remote-to-Local* (R2L, unauthorized local access from a remote machine). The DARPA'98 dataset contains, in all, over 300 attacks in the 9 weeks of data collected for the evaluation. These 300 attacks were drawn from 32 different attack types and 7 different attack scenarios. The attack types covered the different classes of computer attacks and included older, well-known attacks, newer attacks that have recently been released to publicly available forums, and some novel attacks developed specifically for this evaluation. Network based data is provided in tcmdump files and host based data in BSM audit log files. In DARPA'99 dataset, some additional attacks were included and NT audit log files were also provided.

In response to the DARPA'98 data set, McHugh wrote a rather scathing critique of the evaluation [77]. While he presents many good points on how an evaluation of IDSs should be performed, he also criticizes numerous shortcomings in the challenge without acknowledging how difficult addressing some of the issues is. For example, it is noted that the generated data was not validated against real traffic to ensure that it had the same rates of malicious traffic versus nonmalicious anomalous traffic that caused false positives. Doing so would, however, require more insight into real traffic than we can possibly obtain (in particular, intent). Further, modelling of traffic at that scale is still an area with much research left to be done. Some of his more directly applicable feedback was used for the IDS challenge the following year-DARPA'99.

3.5 Dimensionality Reduction

Dimensionality reduction basically refers to mapping a multidimensional space into a space of fewer dimensions. In mathematical terms, it can be stated as: given a D-dimensional vector, $x = (x_1, x_2, \ldots, x_D)$, find a low-dimensional representation of it, $y = (y_1, y_2, \ldots, y_d)$ with $d < D$, that captures the content in the original data, according to some criterion [37]. The reduction becomes effective if the loss of information due to mapping to a lower-dimensional space is less than the gain due to simplifying the problem.

Dimensionality reduction techniques can be broadly categorized into two types: linear and non-linear. Linear techniques use a linear transformation weight matrix $\mathbf{W}_{D\times d}$ to

represent the reduced components as a linear combination of the original components.

$$y_i = x_1 w_{i,1} + x_2 w_{i,2} + \ldots + x_D w_{i,D}, \text{ for } i = 1, \ldots, d \quad \text{or} \tag{74}$$

$$y = x\mathbf{W} \tag{75}$$

Two popular and widely used methods of dimensionality reduction for continuous data are Principal Component Analysis (PCA)[55] and Multi-Dimensional Scaling (MDS)[27]. Both these methods share certain common features as both these are eigenvector methods trying to model linear variation in the original high dimensional space. However PCA tries to retain the maximum amount of variance in the reduced dimensionality space whereas MDS tries to preserve the inter-point pairwise distances. If the pairwise distances in MDS are taken to be Euclidean distance then it becomes equivalent to PCA. PCA and MDS are extensively used for linear dimensionality reduction. However, these methods are inadequate for embedding nonlinear manifolds.

It is possible that the high-dimensional data can have a low dimensional manifold structure and that cannot be captured by the linearity assumption. Recently, several nonlinear dimensionality reduction techniques, such as Isomap [110], Locally Linear Embedding (LLE)[96], Laplacian eigenmaps [10], and Hessian eigenmaps [33], have been proposed that overcome this linearity limitation. Most of the non-linear techniques assume that the data lies on an embedded non-linear manifold within the higher dimensional space. It is shown that LLE has been very useful for the purpose of visualization and unsupervised classification of high dimensional data [14, 18, 48, 52, 94, 97].

In one of our work, we apply manifold learning and more specifically LLE to project high-dimensional categorical data into low dimensional Euclidean space. We demonstrate the effectiveness of this method by clustering the embedded low-dimensional points (and comparing it with clustering of the original high-dimensional categorical data points). We observe that the clustering results are equally efficient as well-known categorical clustering algorithms. A brief description of this work and the corresponding results are given in Appendix A.

In this chapter, we explore the applicability of LLE for intrusion detection using system

call data which are essentially categorical in nature. In the next section we briefly explain the LLE process.

3.5.1 Locally Linear Embedding (LLE)

Recently, Roweis and Saul [96] proposed locally linear embedding algorithm, an unsupervised learning algorithm that can compute low dimensional, neighborhood-preserving embedding of high dimensional data. The basic idea of LLE is global minimization of the reconstruction error of the set of all local neighbors in the dataset, i.e. to find "a low dimensional embedding with the property that nearby points in the high dimensional space remain nearby and similarly co-located with respect to one another in the low dimensional space" [97]. Like PCA and MDS, it is simple to implement, but can compute nonlinear embedding of high dimensional data. The steps of LLE are given below.

Input: A set $X = \{x_1, x_2, \ldots, x_N\}$ of N data points in R^D.

Output: A set $Y = \{y_1, y_2, \ldots, y_N\}$ of N data points in R^d, where $d \ll D$ and each $y_i \in R^d$ corresponds to $x_i \in R^D$, $i = 1, 2, \ldots, N$.

Step 1: The K nearest neighbors of each data point x_i are computed.

Step 2: The reconstruction weights W_{ij}^* for each x_i are computed by minimizing the following error function

$$\psi(W) = \| x_i - \sum_{j=1}^{N} W_{ij} x_j \|^2 \tag{76}$$

subject to the constraint $\sum_{j=1} W_{ij} = 1$, and $W_{ij} = 0$, if x_j is not a neighbor of x_i.

Step 3: Using the optimal reconstruction weights W_{ij}^*, the embedding y_i's are computed by minimizing the following error function.

$$\varphi\,(Y) = \sum_{i=1}^{N} \| \, y_i - \sum_{j=1}^{N} W_{ij}^* y_j \, \|^2 \tag{77}$$

where $W^* = arg\ \min_w \psi(W)$. The error function is minimized subject to the constraints $\sum_i y_i = 0$ and $\sum_i y_i y_i^T / n = I$. The embedding is given by $Y^* = arg\ \min_Y \varphi(Y)$.

The weight W_{ij} stores the contribution of x_j to the linear construction of x_i. These weights characterize the local geometric properties of the data set around each input. The optimal weights for each input can be efficiently calculated by solving the system of linear

equations $Q_i \mathbf{W}_i = e$ for every x_i where the $j\ell^{th}$ entry of Q_i is

$$Q_i(j,\ell) = {d_{ij}}^2 + {d_{i\ell}}^2 - {d_{j\ell}}^2. \tag{78}$$

where, d_{ij} is the distance between the i^{th} and j^{th} points. A more detailed explanation on this is provided in Appendix A. We define d_{ij} for both the categorical and numeric data differently, later in appropriate sections.

The Q_i matrix might sometime be singular. To overcome this problem the matrix is regularized as $Q_i = Q_i + r_i Tr(Q_i)$ where r_i is a small regularization parameter. The optimal reconstruction weights can be obtained in closed form as

$$W_i = \frac{Q_i^- e}{e^T Q_i^- e} \tag{79}$$

where Q_i^- is the generalized inverse of Q_i.

In LLE, it is assumed that a $(K-1)$-dimensional hyperplane passes through the K data points which constitute the K-nearest neighbors of x_i. The hyperplane is represented by the affine combination of these points. When x_i (assumed to be outside the hyperplane) is projected onto the hyperplane, the coefficients of the affine combination for the projection gives the reconstruction weights.

3.5.2 Incremental LLE

LLE lacks generalization to new data points. This makes LLE less attractive in a dynamic environment, where a complete rerun of the algorithm becomes prohibitively expensive. The problem is, given the set $X = \{x_1, x_2, \ldots, x_N\}$ and its corresponding embedding $Y = \{x_1, x_2, \ldots, x_N\}$, how to find the low dimensional embedding y_{N+1} of a new data point x_{N+1}. Kouropteva et al. [62] propose two generalizations to efficiently find the embedding of new data points. These generalizations are discussed below.

- Find the K nearest neighbors of x_{N+1} and let us denote it as a matrix $X_{N+1} = \{x_{N+1}^1, x_{N+1}^2, \ldots, x_{N+1}^K\}$. Let $Y_{N+1} = \{y_{N+1}^1, y_{N+1}^2, \ldots, y_{N+1}^K\}$ be the corresponding embedding of these K nearest neighbors. By using the assumption that the manifold is locally linear, the following equation is approximately true: $Y_{N+1} = X_{N+1} Z$, where Z

is an unknown linear transformation matrix of size $D \times d$, which can be determined as $Z = (X_{N+1})^{-1} Y_{N+1}$. Because X_{N+1} is the neighborhood of x_{N+1} and LLE preserves local structures, the new projection can be found as $y_{N+1} = x_{N+1} Z$.

- Find the K nearest neighbors of x_{N+1}. Then, compute the linear weights w_{N+1}, that best reconstruct x_{N+1} from its neighbors using Equation 76 with the sum-to-one constraint: $\sum_j w_{N+1j} = 1$. Finally, compute the new embedding as $y_{N+1} = \sum_j w_{N+1j} y_j$, where the sum is over the y_j's corresponding to the K nearest neighbors of x_{N+1}.

In the context of IDS, we need to project vectors of new observed processes (test trace, which are high dimensional in nature) into lower dimensional space before classifying them. For this purpose, we make use of the incremental LLE methods proposed by Kouropteva et al. [62].

3.6 Dimensionality Reduction For Intrusion Detection

System call data has been used for different data mining techniques by representing it as vectors, invariably high dimensional vectors. We explore the effectiveness of applying dimensionality reduction for these high dimensional vectors before applying any classification technique. We address the following problem in this study. Is it worthwhile to reduce the dimension by some dimensionality reduction technique for successful intrusion detection?

We observe that in the context of intrusion detection, the need for dimensionality reduction can not be less emphasized. There have been some attempts based on Singular Value Decomposition (SVD) [91] in this direction earlier. However, realizing that manifold learning, Locally Linear Embedding (LLE) in particular, provides a better and robust dimensionality reduction technique, we in this chapter investigate LLE for HIDS using system call data. As system call data can be represented in two different forms, as shown in Section 3.2, we in this present work propose two new algorithms for HIDS based on dimensionality reduction.

3.6.1 Term Frequency Approach (TFLLE)

In term frequency approach, we convert processes under normal execution into vectors, as described in Section 3.2.1. From all the normal processes, a matrix $X = [c_j^\ell]$ is formed, where c_j^ℓ denotes the frequency of S_j, the j^{th} system call of S, in P_ℓ, the ℓ^{th} process, where $1 \leq j \leq m$ and $m = |S|$. We apply the LLE method, as summarized in Section 3.5.1, on X to get a dimensionally reduced matrix Y, where Y_ℓ represents the reduced vector of vector P_ℓ in X. As a part of the LLE method, in Equation 78 we need to compute the distance between vectors (processes) of X. In this case, X is a matrix consisting of numerical values so, the distance between two processes P_k and P_ℓ of X is defined as the Euclidean distance as shown in Equation 80.

$$d_{k\ell} = \sqrt{\sum_j \left(c_j^k - c_j^\ell\right)^2} \tag{80}$$

The interpretation of d as given in Equation 80 is used in Equation 78 to calculate the value of Q.

In order to categorize a new process p into either normal or abnormal class, we first check if the new process p contains a new system call which does not belong to set S. In that case we flag it as abnormal, with the assumption that a system call that does not appear in the normal set is suspicious and thus the process is abnormal. Otherwise, we represent p in the vector form using frequency of system calls. We then compute the embedding of p on the manifold structure by using the first incremental LLE method, which gives the corresponding reduced vector Y_p. We calculate similarity between Y_p and all the processes Y_ℓ in Y using the cosine formula [90], given as,

$$CosSim(Y_p, Y_\ell) = \frac{Y_p \cdot Y_\ell}{\parallel Y_p \parallel \cdot \parallel Y_\ell \parallel} \tag{81}$$

where $\parallel A \parallel$ refers to the norm of the vector A. If any one of $CosSim(Y_p, Y_\ell)$, $\forall \ell$, is equal to 1, it implies that the projection of p is same as the projection of one of the normal process. Thus, p is detected as normal. Otherwise, we find the average similarity value by taking the K highest similarity values. When the average similarity value is above some threshold (λ), process p is considered as normal, and if not, abnormal. The algorithmic representation of the proposed scheme (TFLLE) is given in Figure 13.

Training Phase

1. Form matrix $X = [c_j^\ell]$ from a set of normal processes
2. Apply LLE on X to find the corresponding reduced dimensioned representation Y

Testing Phase

1. **For** each process p in the testing data **do**
2. **if** p has some system calls which does not belong to S **then**
3. p is abnormal; goto step 1 for next process
4. **endif**
5. find Y_p
6. **For** each process P_ℓ in the training data **do**
7. compute $CosSim(Y_p, Y_\ell)$
8. **if** $CosSim(Y_p, Y_\ell)$ equals 1.0 **then**
9. p is normal; goto step 1 for next process
10. **end if**
11. **end do**
12. find average similarity value $AvgSim$ by taking K highest $CosSim$
13. **if** $AvgSim < \lambda$ **then**
14. p is abnormal
15. **else**
16. p is normal
17. **end if**
18. **end do**

Figure 13: Algorithmic representation of TFLLE method

3.6.2 Decision Table Approach (DTLLE)

We represent normal processes in the form of a decision table by extracting subsequences of length D as described in Section 3.2.2. Each of these subsequences is labeled as normal. So, now it can be interpreted as a matrix $X = [x_{ij}]$, where x_{ij} refers to the j^{th} system call of the i^{th} subsequence, where $1 \leq j \leq (D+1)$, and the last column is the class attribute. One thing to note here is, each row of X consists of a sequence of categorical values. So, we define the similarity between two subsequences χ_k and χ_ℓ as,

$$sim(\chi_k, \chi_\ell) = \frac{\chi_k \otimes \chi_\ell}{2D - (\chi_k \otimes \chi_\ell)} \tag{82}$$

where, $(\chi_k \otimes \chi_\ell)$ refers to the number of corresponding attributes in χ_k and χ_ℓ having same value and is defined in Equation 83.

$$\chi_k \otimes \chi_\ell = \sum_{j=1}^{D} x_{kj} \odot x_{\ell j} \tag{83}$$

$$x_{kj} \odot x_{\ell j} = \begin{cases} 1 & \text{if } x_{kj} = x_{\ell j} \\ 0 & \text{otherwise} \end{cases} \tag{84}$$

It is to be noted that, we ignore the class attribute while calculating the similarity between two subsequences. We define the distance between χ_k and χ_ℓ in Equation 85.

$$d_{k\ell} = 1 - sim(\chi_k, \chi_\ell) \tag{85}$$

Example 3.6.1. Given two subsequences as χ_1 = <`fcntl open open fcntl close`> and χ_2 = <`open open fcntl close close`>, we can compute,
$(\chi_1 \otimes \chi_2) = 2$.
$sim(\chi_1, \chi_2) = \frac{2}{(10-2)} = \frac{1}{4}$.
$d_{12} = 1 - \frac{1}{4} = \frac{3}{4}$. □

We apply the LLE method, as described in Section 3.5.1, on X to obtain Y, which is a dimensionally reduced *numerical representation* of X. The distance function defined in Equation 85 is used in Equation 78 to calculate the value of Q for this approach (DTLLE).

Given a new process p for classification, we first extract all the possible subsequences of length D from it. If all of the subsequences are present in the training pool i.e. X, it implies that p does not have any abnormality and thus we flag p as a normal process. Otherwise, for each subsequence $p_{(i)}$ of p which is not present in X, we find its corresponding $Y_{p(i)}$, by the second incremental LLE method described in Section 3.5.2. In this case, X is a matrix consisting of categorical data and Y contains numeric data. Thus, finding the linear transformation matrix Z is not possible, hence we can not use the first incremental LLE method.

We calculate similarity between $Y_{p(i)}$ and each of the vectors Y_ℓ in Y using Equation 81. Taking the K highest similarity values we find the average similarity value for $Y_{p(i)}$. If the average similarity value of any of the subsequence is below some threshold (λ), then the

subsequence is abnormal and so the whole process is flagged as abnormal. One measure advantage of this method is, to classify a process we need not have to wait till the termination of the process, but can extract and test the subsequences as the process executes. The algorithmic form of the proposed scheme, DTLLE, is presented in Figure 14.

Training Phase

1. Construct a decision table X from the normal processes with length of the subsequence as D
2. Apply LLE on X to find the corresponding reduced dimensioned representation Y

Testing Phase

1. **For** each process p in the testing data **do**
2. **For** each subsequence p_i of p **do**
3. **if** $p_i \notin X$ **then**
4. find $Y_{p(i)}$
5. compute $CosSim(Y_{p(i)}, Y_\ell)\ \forall Y_\ell \in Y$
6. find average similarity value $AvgSim$ by taking K highest $CosSim$
7. **if** $AvgSim < \lambda$ **then**
8. p is abnormal
9. goto step 1 for next process
10. **end if**
11. **end if**
12. **end do**
13. p is normal
14. **end do**

Figure 14: Algorithmic representation of DTLLE method

3.7 Experimental Setup and Results

Most of the decision table representation based methods mentioned in the literature such as *tide*, *stide*, *t-stide* use UNM dataset for experimentation. In these cases, for a particular program, say *sendmail*, different instances of its normal execution are run and the traces are collected to form the normal profile. So, in other words, the normal profile consists of normal subsequences from a single program. During test phase, the same program is executed and the new traces obtained are compared with the normal profile for classification.

But, on the other hand, almost all the methods based on term frequency representation are experimented on the BSM audit data from DARPA dataset. It consists of traces from different programs including normal and abnormal programs with appropriate labels as normal or abnormal. We, in this work, use DARPA BSM audit data to test our proposed dimensionality reduction approach for two reasons. First of all, to be able to make a comparative analysis of our results with that of the previous methods. Secondly, for the case of decision table representation based method (DTLLE), we intend to build a normal profile from a group of normal programs and use this normal profile to test new traces of both normal and abnormal programs.

As mentioned in Section 3.4.2 BSM audit logs contains 9 weeks of data. For each day of the BSM audit data, a separate BSM file is provided with the 'BSM List File'. Each line of this file contains the information about one session such as time, service, source IP and destination IP. A '0' at the end of the line shows that the session is normal and the presence of a '1' at the end of the line declares the session intrusive. All the intrusive sessions are labeled with the name of the attacks launched during the corresponding sessions. On analyzing the entire set of BSM logs (list files), we locate the five days which are free of any type of attacks - Tuesday of the third week, Thursday of the fifth week and Monday, Tuesday and Wednesday of the seventh week.

We use the same dataset configuration that is used by Rawat et al. [90] and Sharma et al. [105] for experimentation. The first four days of normal data is used for training. It consists of about 2000 normal sessions and we extract 606 distinct processes from these sessions which are used as training data. The data of the fifth day (Wednesday of the seventh week) is used to test the methods. There are 412 normal sessions on the fifth day and we extract 5285 normal processes from these sessions. These 5285 normal processes are used as testing data to measure the false positive rate. We do not remove duplicate processes from the test set, as we do not require the test processes to be distinct in order to count false alarms for a day [69]. Thus, *false positive* is equal to the number of normal processes detected as abnormal divided by the total number of normal processes.

In order to test the detection capability of our methods, we incorporate the same 55

intrusive sessions as used by Liao et al. [69], Rawat et al. [90] and Sharma et al. [105], into our test data. Table 18 lists these attacks. A number in the beginning of the name denotes the week and day and the later part denotes the name of the session (attack). For example, the attack name *3.1_it_ffb_clear* means that the attack was launched in the 3^{rd} week, on the 1^{st} day viz. Monday and the name of the attack is ffb (in clear mode). These attack sessions consist of almost all types of attacks launched on the victim Solaris machine during seven weeks of training and two weeks of testing period and that can be detected using BSM logs.

Table 18: List of 55 attacks used in test data set

1.1_it_ffb_clear, 1.1_it_format_clear, 2.2_it_ipsweep, 2.5_it_ftpwrite, 2.5_it_ftpwrite_test, 3.1_it_ffb_clear, 3.3_it_ftpwrite, 3.3_it_ftpwrite_test, 3.4_it_warez, 3.5_it_warezmaster, 4.1_it_080520warezclient, 4.2_it_080511warezclient, 4.2_it_153736spy, 4.2_it_153736spy_test, 4.2_it_153812spy, 4.4_it_080514warezclient, 4.4_it_080514warezclient_test, 4.4_it_175320warezclient, 4.4_it_180326warezclient, 4.4_it_180955warezclient, 4.4_it_181945warezclient, 4.5_it_092212ffb, 4.5_it_141011loadmodule, 4.5_it_162228loadmodule, 4.5_it_174726loadmodule, 4.5_it_format, 5.1_it_141020ffb, 5.1_it_174729ffb_exec, 5.1_it_format, 5.2_it_144308eject_clear, 5.2_it_163909eject_clear, 5.3_it_eject_steal, 5.5_it_eject, 5.5_it_fdformat, 5.5_it_fdformat_chmod, 6.4_it_090647ffb, 6.4_it_093203eject, 6.4_it_095046eject, 6.4_it_100014eject, 6.4_it_122156eject, 6.4_it_144331ffb, test.1.2_format, test.1.2_format2, test.1.3_eject, test.1.3_httptunnel, test.1.4_eject, test.1.5_processtable, test.2.1_111516ffb, test.2.1_format, test.2.2_xsnoop, test.2.3_ps, test.2.3_ps_b, test.2.5_ftpwrite, test.2.4_eject_a, test.2.2_format1

On analysis, we find that there is one intrusive session, namely *test.1.5_processtable*, that is exactly similar to one of the process trace in the training data and hence we remove it from the test data. Thus, we consider only 54 intrusive sessions for test purpose. An intrusive session is said to be detected if any of the processes associated with this session is classified as abnormal. Thus, *detection rate* is defined as the number of intrusive sessions detected, divided by the total number of intrusive sessions.

3.7.1 TFLLE

The training data consists of 606 distinct processes. After analysis, we find 50 unique system calls that appear in the training data which constitute the set S (with $m = 50$). The list of 50 unique system calls is shown in Table 19.

All the 606 processes are then represented in vector form to constitute the matrix X of

Table 19: List of 50 unique system calls

```
access, audit, auditon, chdir, chmod, chown, close, creat, execve,
exit, fchdir, fchown, fcntl, fork, fork1, getaudit, getmsg, ioctl,
kill, link, login, logout, lstat, memcntl, mkdir, mmap, munmap,
nice, open, pathconf, pipe, putmsg, readlink, rename, rmdir,
setaudit, setegid, seteuid, setgid, setgroups, setpgrp, setrlimit,
setuid, stat, statvfs, su, sysinfo, unlink, utime, vfork
```

dimension 606×50. We use the test data set consisting of 5285 normal processes and 54 intrusive sessions to test the performance of TFLLE method. We apply LLE on X to get the corresponding reduced representation Y. We experiment with $K = 5$, 10, 15 (K-NN value in LLE) and $d = 10$, 20, 30 (reduced dimension), but the optimal accuracy is obtained for $K = 10$ and $d = 20$. Table 20 shows the results in terms of false positive rate and detection rate. We get 100% detection with almost 0% false positive. In this way, we achieve a case near to ideal one i.e 100% detection with 0% false positive.

Table 20: Detection Rate (DR) and False Positive Rate (FPR) with Term Frequency method

Threshold	Detection rate	False positive rate
0.30	0.351852	0.000000
0.40	0.388889	0.000000
0.50	0.851852	0.000378
0.55	0.888889	0.000378
0.60	1.000000	0.000378
0.65	1.000000	0.000378

We plot the ROC curve of our method along with that of Liao & Vemuri [69], Rawat et al. [90], as shown in Figure 15. From the figure, it is clear that our method dominates all other methods.

For unbiased evaluation of our approach, we also use ten-fold cross-validation. In this cross-validation experiment, we partition the entire data into ten disjoint sets of nearly equal size and select one partition as testing set and the remaining nine are combined to form the training set. This process is repeated 10 times, each time selecting one set as testing set and the other nine as training set, to cover all the data in a circular manner. There are around 2412 normal sessions reported during the five days of data. We extract 676 processes in

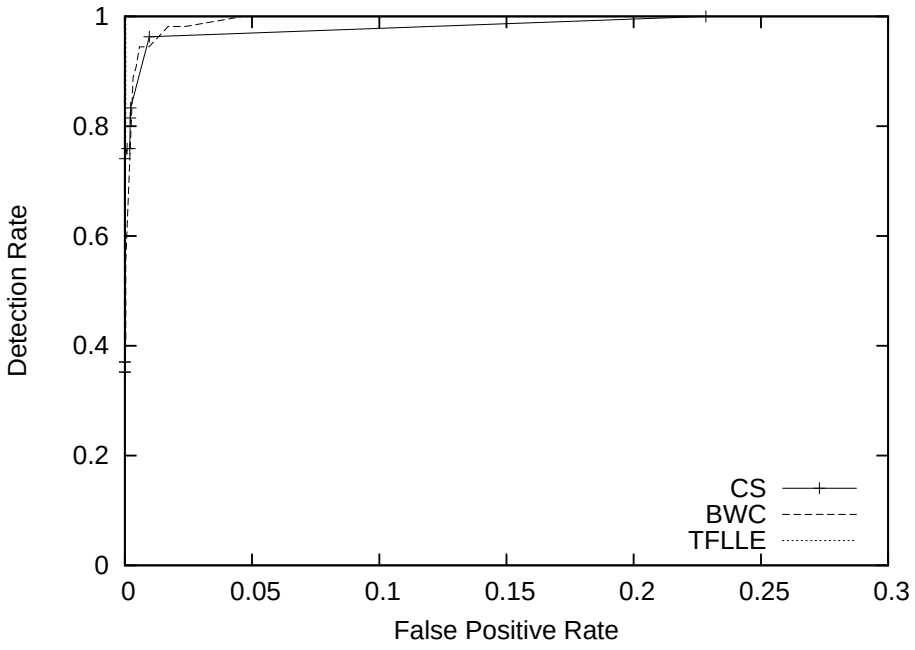


Figure 15: ROC curve of CS, BWC and TFLLE methods

total, occurring during these days. So, each time we train with 608 processes and test with 68 processes, which gives the *false positive* rate.

There are 50 unique system calls in the whole set of 676 normal processes which are listed in Table 19. Every run of cross validation has a different subset of 608 normal processes of training data. Let S ($|S| = s$) be the set of unique system calls in the training set. During experimentation we find that, the set S contains the same 50 unique system calls for all 10 runs, but in general this may not happen. All the 608 training processes are then represented as s dimensional vectors. LLE is applied on these vectors while varying K (no. of nearest neighbours) for values 5, 10, 15 etc. and d (target dimension) for values 10, 20, 30 etc. We test with the remaining normal processes and 52 attacks as explained above. We get the optimal accuracy for $K = 10$ and $d = 20$.

In ten runs of our method for ten-fold cross-validation, we get ten different sets of detection rates and false positive rates. We take the vertical average to obtain a combined result. The combined average result is given in Table 21 for different values of threshold and the ROC curve is shown in Figure 16. Our method attains 100% detection rate with 45.13% false positive and with substantial decrease in the false positive rate (with false positive of

29.13%) we can get a near perfect detection rate of 99.81%.

Table 21: Detection Rate and False Positive Rate with Term Frequency method

Threshold	Detection rate	False positive rate
0.40	0.365385	0
0.45	0.367308	0.0029412
0.50	0.446154	0.0102942
0.55	0.5519231	0.022059
0.60	0.7326924	0.0368568
0.65	0.8461539	0.0649817
0.70	0.9519231	0.1063419
0.75	0.9807691	0.152206
0.80	0.9942307	0.2111212
0.85	0.9980769	0.2912684
0.90	0.9980769	0.4246325
0.91	1.0	0.451287

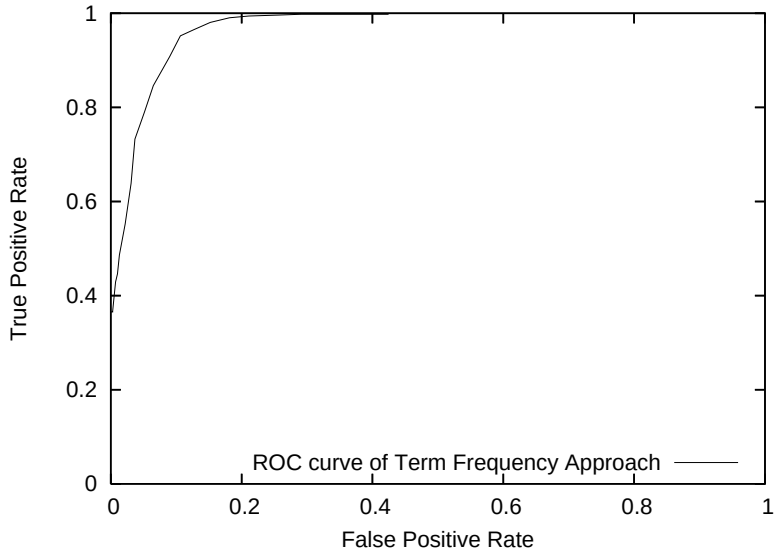


Figure 16: ROC curve of TFLLE with ten-fold cross validation

Most of the experimental results reported in the literature are carried out for distinct training set and test set and these methods did not adopt n-fold cross-validation where the training set is split in the ratio of $n-1:1$ to get the test set. We rerun the SVD based method proposed by Rawat et al. [91] with ten-fold cross validation to compare its performance with our proposed method. Table 22 shows a comparative analysis of our results with that of the SVD based method. Third column of the table reports the best (maximum) detection rate obtained keeping the false positive as 0 and the fourth

column reports the best (minimum) false positive keeping the detection rate as 1. And finally the area under the curve (AUC) value is shown in the last column. For both the categories, namely, *without cross-validation* and *with cross-validation*, LLE approach gives better results than the SVD based method. For instance, the method of LLE without adoption of cross-validation (tagged as R_{1a} in Table 22) performs better, as revealed from the AUC value, than the earlier method by Rawat et al. [91] (tagged as R_{0a}). Similarly, from the same table it can be seen that our method of LLE with cross-validation (tagged as R_{1b}) reports an overall better result in terms of AUC than the SVD method [91] with cross-validation (tagged as R_{0b}).

Table 22: Comparative analysis for Term Frequency representation

Tag	Method	Max DR with FP=0	Min FP with DR=1	AUC
R_{0a}	SVD wo CV	0.37	0.048	0.9930
R_{0b}	SVD w CV	0.626	0.5789	0.9321
R_{1a}	TFLLE wo CV	0.403846	0.000378	0.9999
R_{1b}	TFLLE w CV	0.3653	0.4513	0.9696

3.7.2 DTLLE

Our dataset consists of 606 normal processes as training data; 5285 normal processes and 54 intrusive sessions as test data. We construct a decision table (X) from the training data by extracting subsequences of length 10 ($=D$). Selecting a value for D is important for two reasons. First, if D is small then it gives a compact signature which is very much practical to check for online processes. Conversely, if it is large then it may increase the complexity of detection. Second, small value of D increases the granularity and may thus divide the actual signature into many parts. Considering these cases, we take the value of D to be 10 in our experiments. The decision table thus created contains 5496 distinct subsequences.

To have a comparative analysis, we apply *stide* [50] method on this dataset and the results are shown in Table 23.

In stide method, the abnormality score of a test process (p) is directly proportional to the most dissimilar subsequence of p with respect to X and is not affected by any other subsequence of p. In other words, it works on 1-NN (one nearest neighbour) classification

Table 23: False Positive Rate vs. Detection Rate for stide method

Threshold	False positive rate	Detection rate
0.7	0	0.203704
0.6	0.000189	0.462963
0.5	0.001514	0.648148
0.4	0.003595	0.944444
0.3	0.006055	0.981481
0.2	0.008325	1

principle. But in the proposed DTLLE method we use K-NN classifier which makes the abnormality score dependent upon the K nearest subsequences. Thus, intuitively we modify the *stide* algorithm to have a K-NN based stide algorithm (*k-stide*) as shown in Figure 17. We use the same dataset on *k-stide* for $K = 5$ and 10 and the results are reported in Table 24 and 25 respectively.

Training Phase

1. Construct a decision table X from the normal processes with length of the subsequence as D

Testing Phase

1. **For** each process p in the testing data **do**
2. **For** each subsequence p_i of p **do**
3. **If** $p_i \notin X$ **then**
4. $D_{min}(p_i)$ = average of k $\min\{D(p_i, x_j) \forall x_j \in X\}$
5. **Else**
6. $D_{min}(p_i) = 0$
7. **End if**
8. **End do**
9. find $\hat{S} = (\max\{D_{min}(p_i)\})/D$
10. **If** $\hat{S} \geq \tau$ **then**
11. p is abnormal
12. **Else**
13. p is normal
14. **End if**
15. **End do**

Figure 17: Algorithmic representation of k-NN based stide (*k-stide*) method

We experiment our proposed method, DTLLE, on the same dataset by taking k=10,

Table 24: False Positive Rate vs. Detection Rate for *k-stide* method with k =5

Threshold	False positive rate	Detection rate
7.0	0	0.203704
6.5	0.000189	0.203704
6.0	0.000189	0.462963
5.2	0.000757	0.462963
5.0	0.001514	0.648148
4.8	0.001703	0.925926
4.4	0.003406	0.925926
4.2	0.003595	0.944444
3.8	0.004352	0.944444
3.6	0.00492	0.962963
3.4	0.005676	0.962963
3.0	0.006244	0.981481
2.8	0.007379	0.981481
2.6	0.007569	1

Table 25: False Positive Rate vs. Detection Rate for *k-stide* method with k =10

Threshold	False positive rate	Detection rate
0.7	0	0.203704
0.69	0.000189	0.203704
0.57	0.000189	0.462963
0.55	0.000757	0.462963
0.53	0.000946	0.5
0.52	0.000946	0.518519
0.5	0.001703	0.666667
0.49	0.002081	0.925926
0.45	0.003595	0.944444
0.39	0.004352	0.944444
0.38	0.00492	0.962963
0.35	0.005676	0.962963
0.33	0.005866	0.981481
0.31	0.007379	0.981481
0.29	0.007379	1

and the results in terms of false positive rate and detection rate is shown in Table 26. As can be seen from Table 26, we change the threshold very intricately to capture the precise accuracy of false positive rate (FPR) vs. detection rate (DR). At FPR=0%, the detection rate is 18.519% and at DR=100%, the false positive rate is 0.77578% (41 false alarms out of 5285 normal processes). But the transition of accuracy in between these two extreme points is also important and to capture the overall performance of the method, we use a Receiver

Table 26: False Positive Rate vs. Detection Rate for DTLLE method with k =10

Threshold	False positive rate	Detection rate
0.040949	0	0.18519
0.027197	0.00018921	0.18519
0.02408	0.00018921	0.2963
0.022588	0.00037843	0.2963
0.017168	0.00037843	0.5
0.016123	0.00056764	0.5
0.015546	0.00056764	0.62963
0.014503	0.00075686	0.62963
0.0040209	0.00075686	0.88889
0.0038911	0.0013245	0.88889
0.0035208	0.0013245	0.90741
0.0028646	0.0015137	0.90741
0.0027944	0.0015137	0.92593
0.002402	0.0032167	0.92593
0.0023774	0.0032167	0.96296
0.00067468	0.0068117	0.96296
0.00047146	0.0068117	0.98148
0.00019247	0.0077578	0.98148
0.00018254	0.0077578	1

Operating Characteristic (ROC) curve. We plot the ROC curve for the above three results in Figure 18, and report the Area Under the Curve (AUC) value in Table 27.

Table 27: Area under the curve (AUC) value for different methods

Method	AUC
stide	0.99874908
k-stide (k=5)	0.9988945
k-stide (k=10)	0.99883318
DTLLE	0.99919058

From Figure 18 we observe that none of the methods seems to be perfectly superior to any other. In terms of AUC value, the methods can be sorted as DTLLE (0.99919058), k-stide with k=5 (0.9988945), k-stide with k=10 (0.99883318), stide (0.99874908). Thus, by comparing the AUC values, we can conclude that DTLLE performs better than all others. It can also be seen that, the modified version of stide method (k-stide) proposed by us, performs better than stide method for both the experimented values of k.

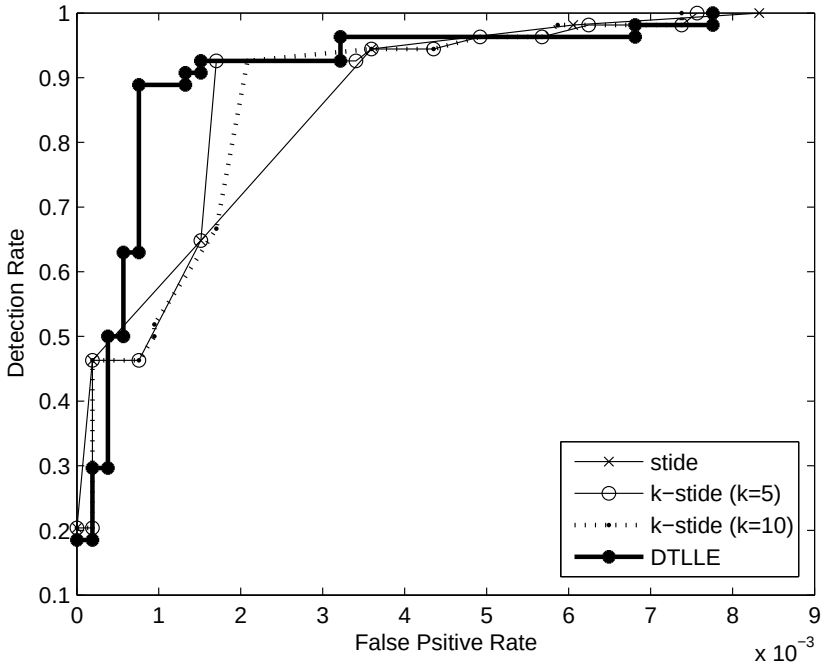


Figure 18: Performance of stide, k-stide and DTLLE methods expressed in ROC curves

3.8 Conclusions

In this chapter, we explore the usability of LLE method in the arena of HIDS. we observe that the existing methods represent system calls data in either of the two formats – term frequency and decision table representations. This results in a high dimensional representation of the system calls data which may have a low dimensional embedding which cannot be captured by any linearity assumption. The motivation of the present work is the requirement to have a fast and accurate HIDS. Instead of presenting a new method of classifying data as normal or abnormal, the present study focuses on preprocessing of the data. We propose two methods to achieve the aim based on LLE. The first method, TFLLE, is based on frequencies of system calls, and the second method, DTLLE, is based on subsequences of system calls obtained from process trace. In each of these methods, we apply LLE to map the high dimensional data into a low dimensional embedding. The advantage with LLE is that it tries to preserve the local neighborhood structure of the original data in the reduced dimension. Given a test process we convert it to corresponding representation and map it to low dimension using incremental LLE method. In the low dimensional space, we

use a K-NN classifier to classify the test case. One important thing to be noted is that for decision table representation, we are able to map a categorical dataset into a numerical low dimensional space.

For the TFLLE method we obtain a detection rate of 100% with a marginal false positive rate of 0.0378% which is much better than the results reported by earlier methods. We also compare this result with another dimensionality reduction based method proposed by Rawat et al. [91] and show that our method performs better.

We modify the stide [50] method to obtain its k-stide variant and empirically show that the k-stide method performs better than the original stide method. But experimental results show that our proposed DTLLE method gives better results than both the stide and the k-stide methods.

The use of LLE enhances the overall performance of IDS mainly due to two reasons. First, it reduces the dimension, thereby making the computational efforts less. Second reason being the reduction of noise in the data. By reducing the noise, we can expect a better classification of normal and abnormal data. Our future work will be focusing on analyzing more on system call data and in particular DARPA BSM data. As a part of this, we are focusing on the properties of data for which LLE yields better results.

CHAPTER IV

NEW MALICIOUS CODE DETECTION USING VARIABLE LENGTH N-GRAMS

4.1 Introduction

Malicious code is any code added, changed, or removed from a software system to intentionally cause harm or subvert the system's intended function [76]. According to its propagation methods, malicious code is usually classified into viruses, worms, trojan horses, back doors and spyware. *Viruses* are self-replicating pieces of malicious code that attach themselves to other program and propagate when the infected program executes. *Worms* self-replicate across a network by carrying out programmed attacks to jump from machine to machine across a network. *Trojan horses* masquerade as useful programs, but contain malicious code to attack the system or leak data. *Back doors* allow remote access and control of the host over a network by subverting the local security policies, thus opening the system to external entities. *Spyware* is a useful software package that also transmits private user data to an external entity. Combining two or more of these categories can lead to fatal attack tools. For example, a worm can contain a payload that installs a back door to allow remote access. When the worm replicates to a new system (via email or other means), the back door is installed on that system, thus providing an attacker with a quick and easy way to gain access to a large set of hosts [21].

Any computer system is vulnerable to malicious code whether or not it is attached to other systems. Malicious codes can affect the secrecy, the integrity, the data and control flow, and the functionality of a system. Therefore, their detection is a major concern within the computer science community as well as within the user community [11]. The *Gartner* report [43], published in 2005, ranked viruses and worms as top security threats and hence detecting malicious code has become one of the prime research interests in the field of

information security.

In this chapter, we concentrate on new malicious code detection especially computer viruses. There are two approaches that are poles apart in virus detection. One approach is too generic which includes Activity monitors and Integrity management systems. *Activity monitors* alert to system activity that is commonly associated with viruses, but only rarely associated with the behavior of normal, legitimate programs. *Integrity management systems* warn against suspicious changes made to files.

The other approach, *Signature based virus detection*, is too specific and also popular. Almost all the commercial antivirus products rely on this approach. In this, a database of virus signatures (a virus-specific sequence of instructions) is maintained and a program is detected as a virus program based on the presence of these virus signatures. For example, the Chernobyl/CIH virus is detected by checking for the hexadecimal sequence [120]:

```
E800 0000 005B 8D4B 4251 5050
0F01 4C24 FE5B 83C3 1CFA 8B2B
```

The signature based virus detection technique is very effective in detecting known viruses (for which virus signature is available). But, at the same time it suffers from two major issues. One is, it cannot detect new and unknown viruses as their signatures are not present in the database. Also, sometimes new variants of known viruses can easily escape the detection by simple defenses like code obfuscation [21]. The other issue is, this technique is not scalable. As the number of known viruses increases, the size of the signature database increases and also the time of checking a file for virus signatures increases. Generation of virus signature is a cumbersome process and is prone to generating false positives on benign programs. For instance, a faulty virus signature issued by Symantec mistakenly removed essential operating system files, leaving thousands of PCs unable to boot [109].

Unlike signature based detection, generic virus detector uses features that are common to most of the viruses and that characterize only the virus programs. The assumption here is that viruses have certain typical characteristics in common and these characteristics are not present in benign programs. For instance, most of the virus writers use virus generating

toolkits [108], for example PS-MPC (Phalcon-Skism Mass-Produced Code Generator), to write and compile their code. It is believed that more than 15,000 variants of viruses were generated using this kit alone and there are more than 130 such kits available. The viruses generated using a toolkit have certain features that are specific to the toolkit, the compiler and also the programming environment [5].

Fred Cohen, in his seminal paper [22], proved that there is no algorithm that can detect the set of all possible computer viruses (returning "true" if and only if its input is an object infected with some computer virus). The proof is a simple diagonal argument like Turing's proof of the undecidability of the Halting Problem [112]. Cohen defines that an algorithm A detects a virus V if and only if for every program p, $A(p)$ terminates, and returns "true" if and only if p is infected with V. It then defines a program p, which reads: *if* $A(p)$ *then exits, else spread* for any candidate computer virus detection algorithm A. Clearly, if p is tested by the virus detection algorithm A, then it leads to a contradictory result, since if it returns "true" (i.e. it says that p is infected), then p just exits (and is therefore not infected). On the other hand, if A returns anything else (i.e. it says that p is not infected), then p spreads (and is therefore infected). So there is no algorithm which detects all viruses without error; any program that attempts to detect all viruses will either miss some infected files (a false negative), accuse some non-infected files of being infected (a false positive) or fail to return anything (a bug) [20].

It is clear from the foregoing discussion that a generic virus detector is though desirable is very difficult to accomplish in its true form. A practical approach is to develop a machine learning based detector where the detection process learns the generic characteristics from a set of known examples but avoids identifying any specific signatures. Recently, there have been several such attempts for detection of malicious code. The next section discusses such earlier methods.

The rest of the chapter is organized as follows. Section 4.2 discusses related work available in the literature on detection of malicious executables (using n-grams). Our observations and motivation for the present work is discussed in Section 4.2.8. In Section 4.3, we describe a method of discovery of variable length n-grams based on the concept of

episodes. We describe the concept of relevant episodes in Section 4.4. Section 4.5 discusses the experimental setup which is followed by the experimental results in Section 4.6. We conclude our work in Section 4.7.

4.2 Earlier Work

4.2.1 KSACTW ANN Scheme

Kephart et al. [57] propose the use of Neural Networks to detect boot sector malicious binaries. A boot sector is a small sequence of code (512 bytes for IBM-compatible PC's) that tells the computer how to "pick itself up by its bootstraps". A boot sector virus

3-byte features (trigrams) are extracted from a set of 150 boot sector viruses (training set). As this results in a high number of trigrams, selection is done in two phases. First, by removing trigrams which appear in benign boot sector code and trigrams with high frequency. Secondly, a subset of trigrams is selected such that each malign boot sector is represented by at least four trigrams from this set. These resulted in 50 trigrams which are used to construct binary feature vectors for the training set based on the presence (1) or absence (0) of these trigrams. But, as all the feature vectors are positive examples, artificial negative examples are added to the training set. For each feature an artificial negative example is generated in which that feature's input value is 1 and all other inputs are 0. Both positive and artificial negative feature vectors are used as input feature vectors to train the classifier, a single-layer feed-forward neural network. Thus, the classifier consists of about 50 input nodes and one output value. If the output is positive, it is detected as a virus, else a legitimate boot sector code. Empirically it is shown that this classifier can detect new instances of boot sector code with 15% false negative and 0.02% false positive rate.

4.2.2 AT ANN Scheme

A Win32 heuristic virus detector is proposed by Arnold et al. [4]. Pruned set of n-grams collected from both uninfected Win32 programs and Win32 viruses is used as features to build a model. For the uninfected Win32 programs, the compressed data part is excluded

while finding the n-grams. Similarly, n-grams are collected only from constant regions (regions which remain constant across all instances of the virus) of the Win32 viruses. Pruning is done to extract a smaller set of n-grams from the Win32 viruses, by selecting those n-grams that appear in the corpus of uninfected programs fewer times than a threshold. A greedy algorithm is then used to generate complete 1-cover n-grams from the n-grams of virus pool. A complete 1-cover consists of a set of features (n-grams) selected such that at least one feature is present in each virus. 8 such covers are generated without reusing the n-grams which are used to construct the input vector sets for a linear neural network classifier. Thus, each input vector set (for a Win32 virus or clean program) consists of 8 vectors one for each cover consisting of frequency of n-grams. Voting is used to combine the outputs from each network into a single result. If the sum of the outputs from all 8 networks is greater than a voting threshold τ, then output value is 1 (infected), else output value is 0 (uninfected).

4.2.3 SEZS Data Mining Scheme

Motivated by the success of data mining techniques in host based and network based intrusion detection system, Schultz et al. [102] propose several data mining techniques to detect different types of malicious executables. Different features are extracted statically from both benign and malicious binaries which represent different information contained within each binary. These features are described below.

DLL information: Three different features are extracted from the program header which convey the resource information used by the binaries. These are, the list of DLLs (Dynamic Link Libraries) used by the binary, the list of DLL function calls made by the binary and the number of different calls within each DLL.

Example 4.2.1. A feature vector such as

$$\neg\texttt{advapi32} \wedge \texttt{avicap32} \wedge\neg\texttt{wsock32}$$

informs that the binary is composed of two unused resources: `advapi32.DLL` and `wsock32.DLL` and uses the resource `avicap32.DLL`. Similarly, the vector

`advapi32.AdjustTokenPrivileges()` ∧ `wsock32.recv()` ∧ `wsock32.send()`

is composed of three function calls. One function is called in `advapi32.DLL`, which is `AdjustTokenPrivileges()`, and two functions are called in `wsock32.DLL`, which are `recv()` and `send()`. A feature vector such as

`advapi32=2` ∧ `avicap32=10` ∧ `winmm=8`

represents a conjunction of DLLs and a count of the number of functions called inside each DLL. □

The feature vector consists of 2259 boolean values representing whether or not a DLL is used (30 DLLs are considered), whether a DLL function is used (2229 distinct DLL functions are considered) and 30 integer values representing number of function calls used within each DLL. RIPPER [24], a rule based learner, is used to build a set of rules from these features to identify the benign and malicious classes.

GNU Strings: A binary file can be seen as a sequence of printable and non-printable characters. *GNU strings* program is used to extract consecutive printable characters, forming a string, from binaries. Some common strings found are `kernel, Microsoft, windows, win, null,heap,getversion,closehandle`. These strings serve as features to train a classifier.

Byte Sequence: *Hexdump* [78] tool is used to transform binary files into hexadecimal files containing byte sequence. Each byte sequence in a binary is used as a feature. The byte sequence representation of a sample binary file is shown in Table 28.

Table 28: Byte sequence representation of a sample binary file

6f72	7267	1f0e	0eba	b821	4c01	21cd	6854
7165	6975	7369	7020	6d61	7220	b400	cd09
2e73	0a0d	0024	0000	454e	3c05	026c	0009
000c	0000	0001	0060	021c	0238	0244	02f5

Naive Bayes classifier is used for classification using both the strings and Byte sequence features. It assumes the occurrence of features to be independent of each other and computes

the likelihood that a program is malicious or benign given a set of features, according to naive bayes rule.

In a companion paper [101] the authors develop a UNIX mail filter that detects malicious Windows executables based on the above work. Byte sequences are used as the set of features as machine codes are most informative to represent executables. The RIPPER algorithm of rule discovery and Naive Bayes classifiers are used for classification in this study.

4.2.4 ACKS CNG scheme

Abou-Assaleh et al. [5] observe that common n-grams (CNG) [**?**] extracted from byte sequences can be used as effective features to differentiate between malicious and benign code. During training, profiles for each of the benign and malicious classes are built. From the training data of each class, n-grams with their normalized frequencies are counted and L most frequent n-grams with their normalized frequencies are used to represent the class profile. For a test code, profile is built in the same way and 1-nearest neighbor is used for classification. Distance is computed as per Eq. 86 between the test profile and each of the class profiles and the class with the closest distance is selected.

$$\sum_{s \in Profiles} \left(\frac{f(s) - f_c(s)}{\frac{f(s)+f_c(s)}{2}} \right)^2 \tag{86}$$

where, s is any n-gram from one of the two profiles, $f(s)$ is the frequency of s in test profile and $f_c(s)$ is the frequency of s in a class profile. If an n-gram does not exist in a profile, then its frequency is treated as 0.

4.2.5 KM n-gram Scheme

Kolter et al. [60] independently realise that n-grams can possibly be used as a set of features. Executables from both classes (benign and malicious) are used for training a classifier. During training n-grams are extracted from the executables and L most relevant n-grams are selected by computing the information gain (IG) as shown in Eq. 87.

$$IG(j) = \sum_{v_j \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_j, C) \log \frac{P(v_j, C)}{P(v_j)P(C)} \tag{87}$$

Where, C is the class, v_j is the value of the jth attribute (n-gram) (1 represents present and 0 represents absent), $P(v_j, C)$ is the proportion that the jth attribute has the value v_j in the

class C_i, $P(v_j)$ is the proportion that the jth n-gram takes the value v_j in the training data, $P(C)$ is the proportion of the training data belonging to the class C [60]. Each executable in the training set is represented by a vector of size L indicating the presence (1) or absence (0) of the corresponding n-gram. Several classification techniques such as k-nn, TFIDF, Naive Bayes, SVM, Decision Trees, Boosted Classifiers which are implemented in WEKA [124] are used for experimentations. Empirically it is shown that boosted J48 algorithm performs better.

4.2.6 Cai Scheme

In a recent paper [16], Cai et al. compare seven different feature selection measures and four different classification algorithms in identifying malicious executables. In this approach, N-grams are extracted from the byte sequence of binary executables (both malicious and benign) and different feature selection methods, as mentioned below, are applied to select relevant set of n-grams. The feature selection methods are,

- Maximal Difference (MD): It computes the difference between the average frequency of an n-gram, s, in the benign executables ($f_B(s)$) and in the malicious executables ($f_M(s)$) as $f_B(s) - f_M(s)$ and the difference is sorted in ascending order. It then selects, $L/2$ top n-grams and $L/2$ bottom n-grams to have L n-grams as features.

- Maximal normalized difference (MND): It is same as the MD method except that it uses normalized difference between n-grams, $(f_B(s) - f_M(s))/\sigma(s)$, where $\sigma(s)$ is calculated as,

$$\sigma^2(s) = \frac{N_B\sigma^2(s|B) + N_M\sigma^2(s|M)}{N_B + N_M} \tag{88}$$

  where N_B and N_M are the number of benign and malicious executables respectively, $\sigma(s|B)$ is the standard deviation of the frequencies $f(s)$ over all benign executables and similarly $\sigma(s|M)$ for the malicious executables.

- Maximal Ratio (MR): It is same as MD except that instead of using differences it uses the ratio of frequencies, $f_B(s)/f_M(s)$, or equivalently $\log f_B(s) - \log f_M(s)$.

- Maximal Weighted Ratio (MWR): MWR is a modified maximal ratio that assigns weight based on the overall frequency of an n-gram as shown in Eq. 89

$$(f_B(s) + f_M(s))(\log f_B(s) - \log f_M(s)). \tag{89}$$

- Maximal Kullback-Leibler distance (MKLD): It uses a symmetrized variant of the original Kullback-Leibler distance [63] and computes MKLD as shown in Eq. 90 for each n-gram (s).

$$(f_B(s) - f_M(s))(\log f_B(s) - \log f_M(s)). \tag{90}$$

  As the above distance is always positive, so it selects L features with the largest values of MKLD.

- Bi-normal separation (BNS): BNS [38] is defined as $N^{-1}(tpr) - N^{-1}(fpr)$ where N^{-1} is the standard normal distribution's inverse cumulative probability function and *tpr* and *fpr* are true positive and false positive rates respectively. For each byte pattern (s) the largest value of $N^{-1}(tpr) - N^{-1}(fpr)$ called $D_{\max}(s)$ is selected by using Naive Bayes classifier and L byte patterns with higher $D_{\max}(s)$ are selected as features.

- Forward stepwise selection (FSS) and Backward stepwise selection (BSS): Both the FSS and BSS selects a set of features in an iterative fashion. In FSS each feature (1-gram) is used to form a one-feature predictive model. The features that gives the best performance is selected and the remaining features are paired with these selected features to evaluate all of the two-feature models. This process continues until all desired number of features is chosen or the performance stagnates. Similarly, the BSS does so as FSS but in the opposite direction. The BSS starts with a full set of features and at each step reduces the number of features by one, choosing the one with the least performance.

It is reported that single-byte patterns provided reliable features to separate malicious executables from benign.

4.2.7 Other approaches

Besides the above mentioned machine learning approaches, which tries to find common and distinguishing patterns for different classes of executables, static and dynamic analysis of executables to detect malices is also mentioned in literature.

4.2.7.1 Static Analysis

Static analysis examines the code of a program without executing it to determine its dynamic properties. The method proposed by Bergeron et al. [11] translates binary code into an internal intermediate form to perform flow-based analysis and is then checked against the security policies. Lo et al. [70] performs static analysis based on tell-tale signs which are program properties that discern malicious programs from benign programs easily with very high accuracy without the need to give a specification of the program. Static analysis is also attempted by Christodorescu et al. [21], where the problem of malicious detection is treated as an obfuscation-deobfuscation game between malicious code writers and researchers working on malicious code detection.

4.2.7.2 Dynamic Analysis

Dynamic analysis combines testing and debugging to detect malicious activities by running a program. It includes wrappers [7], sandboxing [2] etc. Behavior blocker is a technique which monitors program's activity and prevents the program if it initiates a possibly harmful action. The Bloodhound technology (used by Symantec) and ScriptTrap technique (by Trend Inc.) are examples of this technique. Egele et al. [36] propose a dynamic analysis approach to detect spyware by precisely tracking the flow of sensitive information as it is processed by the web browser and any loaded browser helper objects.

4.2.8 Observations and Motivation

It is interesting to note that many of these techniques use byte n-grams as basic features to detect malicious codes. Byte n-grams are overlapping continuous substrings, collected in a sliding-window fashion where the windows of fixed size of n slides one byte at a time.

N-grams have the ability to capture implicit features of the input that are difficult to detect explicitly. Byte n-grams can be viewed as features in the present context when an executable program is viewed as a sequence of bytes. N-grams have been successfully used for a long time in a wide variety of problems and domains, including information retrieval, language identification, automatic text categorization, computational immunology, authorship attribution etc. In many domains, techniques based on n-grams gave very good results. For instance, in natural language processing, n-grams can be used to distinguish between documents in different languages in multi-lingual collections and to gage topical similarity between documents in the same language. Some of the good features of n-grams are simplicity, robustness and efficiency. On the other hand, the problem which can appear in using n-grams is exponential explosion. It is clear that many of the algorithms with n-grams are computationally too expensive even for $n = 5$ or $n = 6$.

The main disadvantage of fixed-length n-grams is that they cannot capture meaningful n-gram sequences of different lengths [30]. Though variable length n-grams are previously used in intrusion detection [30, 54, 73] and text categorization [17, 40], no attempts have been made to use them in malicious code detection. In Section 2.5, we demonstrate that episodes as variable length n-grams can be used for Masquerade Detection. We extend this idea to extract variable length n-grams from byte sequences using the concept of episodes and select class-wise relevant n-grams to act as features for malicious code detection. The episode discovery algorithm given in Section 2.5.2 is suitable only for one sequence at a time. But in the present case, we need to find variable length n-grams for a class of executables (benign and malicious), so it is necessary to find episodes for a set of sequences. In the following section, we describe the method for finding episodes from a class of executables.

4.3 Episode discovery from a set of sequences

In the present context, we extend the algorithm described in Section 2.5.2 to determine episodes from a set of sequences. The simple extension would mean that we construct one trie for each sequence. But we propose to store the information of all sequences in a single trie structure. Thus we can capture the frequency and entropy of any element over all the

sequences together. The method of construction of trie for multiple sequences is illustrated in Example 4.3.1.

Example 4.3.1. Let us consider the following set of four sequences.

S_1 = `(bf 0e 3a bf d8 3a 3a bf)`

S_2 = `(3a bf bf 0e 3a 3a bf bf d8 3a)`

S_3 = `(bf d8 3a bf 0e 3a bf 0e 3a 0e 3a)`

S_4 = `(0e 3a bf d8 3a 3a bf 3a bf 0e 3a)`

The trie structure for S_1 can be obtained by using the trie construction algorithm outlined in Figure 2 with $d = 4$ and is shown in Figure 19. From the trie structure we note that `(bf)` appears 3 times in S_1, `(3a bf)` appears twice and `(3a 3a)` appears once. The structure captures the frequency of n-grams of different lengths (at most $d-1$). We embed S_2 on this structure to get the trie representing S_1 and S_2 (Figure 20). The trie with S_1, S_2 and S_3 is shown in Figure 21. The final trie structure after considering all the sequences is shown in Figure 22.

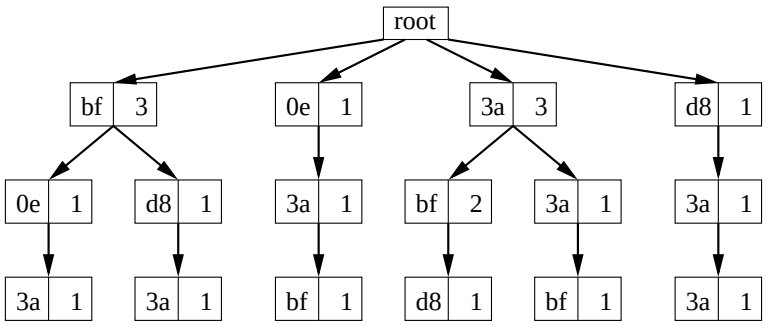


Figure 19: Trie for $S_1$with depth=4.

□

The algorithm for finding episodes from each of the sequences, using the above obtained combined trie structure (T_k), is shown in Figure 23. For each of the sequence S_t we take a sliding window of length $n(= d-1)$. Let $x_1, x_2, \ldots, x_n$ be the elements falling in the window at one instance. For each of the n possible break points in the window, we examine the frequency and entropy as follows.

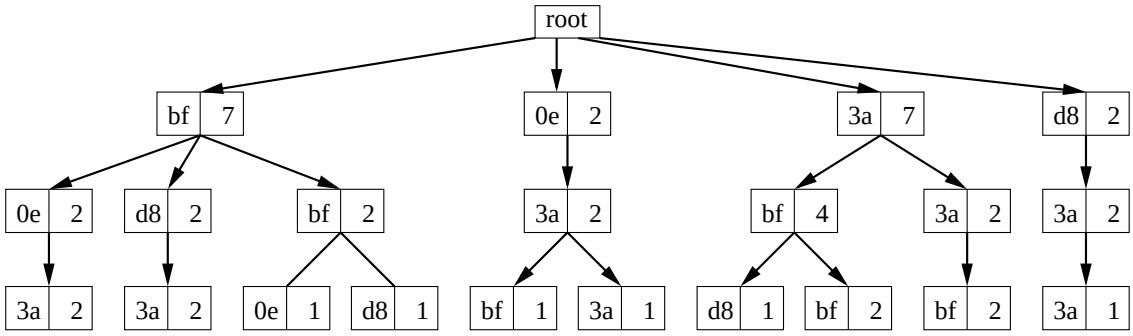


Figure 20: Trie after embedding S_2 into Figure 19.

The entropy at location i (between x_i, x_{i+1}) is the entropy of the node x_i at level i of the trie along path x_1, x_2, ..., x_i. For example, for S_1, with n=3 we get a window `(bf|0e|3a|)` with 3 positions for break points. The entropy at the first location is entropy of the node labeled (`bf`) at the first level of the trie. Similarly entropy at the second location is the entropy of the node labeled (`0e`) at level 2 with (`bf`) as the parent at level 1 in the trie. The location corresponding to the highest entropy in the window is identified and its score is incremented by 1.

The frequency at location i is calculated by the sum of the frequencies of subsequences $(x_1 \dots x_i)$ and $(x_{i+1} \dots x_n)$. For example, the frequency at the first location of the window `(bf|0e|3a|)` is $f($`bf`$) + f($`0e 3a`$)$, where $f($`bf`$)$ refers to frequency of the node labeled (`bf`) at the first level of the trie and $f($`0e 3a`$)$ refers to frequency of the node labeled (`3a`) at the second level of the trie whose parent node is (`0e`) at level 1. The score at the location with highest frequency is incremented by 1. In this case, our goal is to maximize the sum of the frequencies of the left and right subsequences of the probable break point.

After sliding the window across the sequence, we end up with scores for each location in the sequence. In a stream of $|S_t|$ 1-grams, there are $|S_t| - 1$ positions within the sequence. If a position is repeatedly voted for break point by different windows then it is likely to accrue a locally-maximum score. We choose the positions with local maximum of the score as break points of the episode.

Example 4.3.2. Continuing Example 4.3.1, we can find the list of episodes from each of the four sequences using the algorithm given in Figure 23. These are as follows.

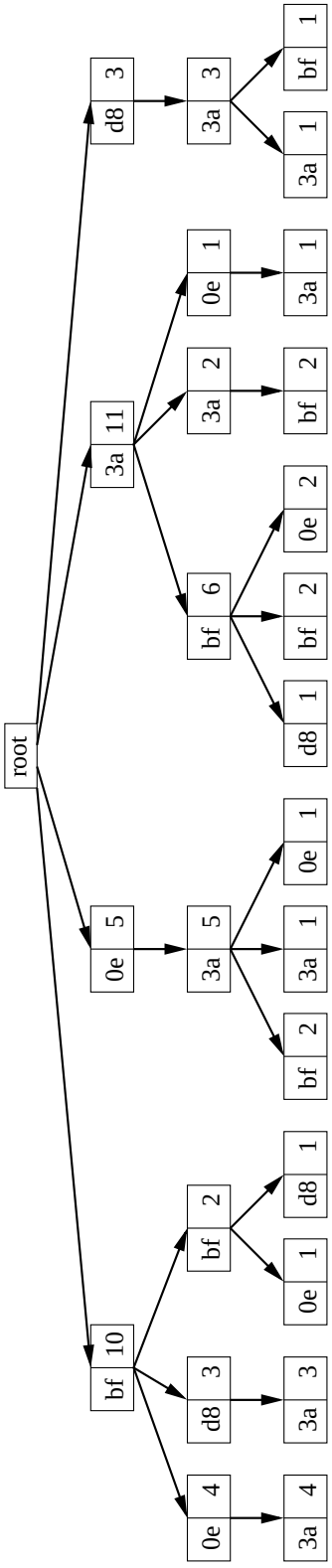


Figure 21: Trie after embedding S_3 into Figure 20.

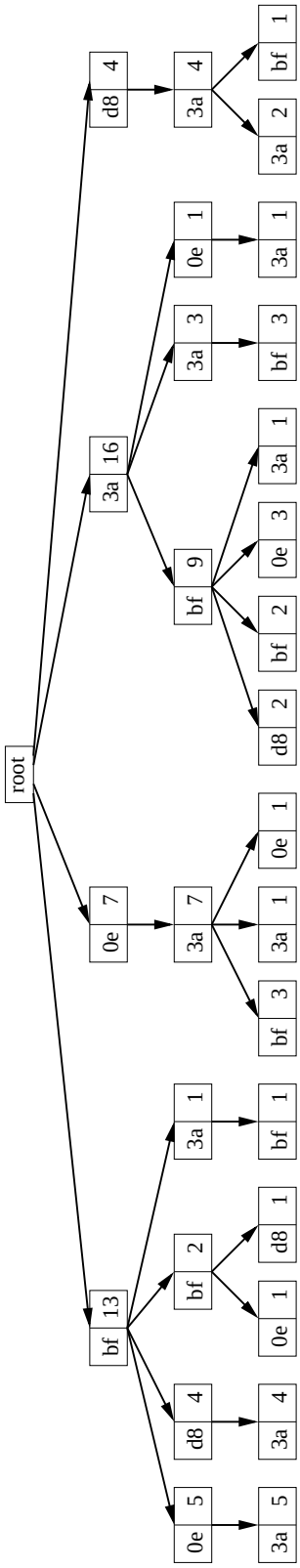


Figure 22: Trie after embedding S_3 into Figure 21.

Algorithm: To find episodes for a set of sequences S
Input: T_d (trie of depth d) and S

1. **for** each of the node $x \in T_d$ **do**
2. calculate entropy $e(x)$
3. **enddo**
4. **for** each level L_i of T_d **do**
5. find mean frequency ($\overline{f}$) and mean entropy ($\overline{e}$)
6. find standard deviations ($\sigma_{\overline{f}}$ and $\sigma_{\overline{e}}$) taking $\overline{f}$ and $\overline{e}$ respectively
7. $f(x) = \frac{f(x)-(\overline{f})}{\sigma_{\overline{f}}}$ and $e(x) = \frac{e(x)-(\overline{e})}{\sigma_{\overline{e}}}$ for $x \in L_i$
8. **enddo**
9. **for** each sequence $S_t(s_1, s_2, \ldots, s_{|S_t|}) \in S$ **do**
10. Episodes $E_t = \phi$
11. initialize $score(i) = 0$, for $1 \leq i \leq |S_t|$
12. **for** $i = 1$ to $(|S_t| - k + 1)$ **do**
13. take a window of length k starting at position i in S
14. **for** for each of the k possible boundary positions in the window **do**
15. find $Max_j\{f(s_i, \ldots, s_{i+j}) + f(s_{i+j+1}, \ldots, s_{i+k-1})\}$, $0 \leq j < k$
16. increment $score(i + j)$
17. find $Max_j\{e(s_i, \ldots, s_{i+j})\}$, $0 \leq j < k$
18. increment $score(i + j)$
19. **enddo**
20. **enddo**
21. $start = 1$
22. **for** $i = 2$ to $(|S_t| - 1)$ **do**
23. **if** $(score(i) > score(i - 1))$ and $(score(i) > score(i + 1))$
24. $end = i$
25. add $(s_{start}, \ldots, s_{end})$ to E_t
26. $start = end$
27. **endif**
28. **enddo**
29. add $(s_{start}, \ldots, s_{|S_t|})$ to E_t
30. **enddo**

Figure 23: Episode discovery algorithm for a set of sequences using a combined trie

E_1 = ((bf 0e 3a), (bf d8 3a), (3a bf))

E_2 = ((3a bf), (bf 0e 3a), (3a bf), (bf d8 3a))

E_3 = ((bf d8 3a), (bf 0e 3a), (bf 0e 3a), (0e 3a))

E_4 = ((0e 3a), (bf d8 3a), (3a bf), (3a bf 0e 3a)) □

Observing the above episodes, it is clear that episode discovery algorithm gives meaningful n-grams of variable length. If we consider fixed length n-grams, say for $n = 2$, then obviously we lose valuable information by missing n-grams for $n \neq 2$.

4.4 Relevant episodes

We observe that there can be large number of episodes for a single program and when we consider all the programs in the training set the set of distinct episodes becomes very large. We introduce here two novel concepts – *relevant episodes* for a class and a new feature selection measure, namely *class-wise episode frequency*. The main aim of our feature selection method is to identify a smaller set of features that are best representative of the virus and benign classes.

Let the set of virus programs be V and the set of benign programs be B.

Definition 4.4.1. The class-wise episode frequency of an episode with respect to a class V (or, B) is the number of times it occurs in the class V (or, B). □

While the term frequency is a kind of global measure, the class-wise episode frequency is a local measure with respect to a class. The main advantage of class-wise episode frequency is that we can analyze each class independently. This saves memory requirement as we handle only episodes of one class at a time.

For each executable program t in a class T of programs, let E_t be the set of all episodes. The set of all episodes for T, $E(T)$ is $\cup_{t \in T} E_t$. We assume that the elements of $E(T)$ are arranged in the non-increasing order of class-wise episode frequency.

Definition 4.4.2. We define $E^k(T)$ as the *relevant episodes* for T which is a subset of $E(T)$ containing only first k elements. □

Thus, the relevant episodes for classes V and B are $E^k(V)$ and $E^k(B)$, respectively. We get the set of relevant episodes for the whole training data as $E^k(V) \bigcup E^k(B)$.

With the set of relevant episodes for the data set, we build the vector space model which is a concept derived from information retrieval. An executable program is represented as a vector of $t_1,\ t_2, \ldots,\ t_M$, where t_i, $(1 \leq i \leq M)$ is a binary $(0-1)$ value denoting the occurrence of the i^{th} relevant episode. The value 1 represents the occurrence of an episode and its absence is represented by 0. Thus, each unique relevant episode corresponds to a dimension. Our training set consists of a set of labeled vectors – the vector representation of the set of programs together with the respective class label (virus or benign).

As we observed, the detection problem reduces essentially to supervised classification problem. Several algorithms exist [79] for supervised classification like support vector machine, decision tree, neural networks etc. We use the metaclassifier Ada Boost with J48 as base classifier available in WEKA [124]. The reason for choosing this particular classifier is that the authors of previous work [60] in this area claimed that they got the best results for this classifier.

4.5 Experimental Setup

Unlike the case of intrusion detection, there is no standard data set available for the detection of malicious executables. Most of the previous studies available in the literature [5, 60] use malicious executables collected from the VX Heavens [117] website. And the benign executables are collected from their respective laboratories.

For our studies, we collected 250 viruses from VX Heavens [117] and 250 benign executables from our lab, like files from the windows system directories, genuine windows utility files. For viruses, we use only the loader programs; we did not use the infected programs for our analysis. At present, we only concentrate on viruses.

Each executable in the dataset is converted to hexadecimal codes in an ASCII format. A sample of the hexadecimal code (1-byte sequence) of the *Dark Avenger* virus is shown in Table 29. To capture the sequence information for each class (V and B), we build a trie structure for each class using the training executables of the corresponding classes. Based

Table 29: Part of the hexadecimal code of Dark Avenger virus

```
4d 5a 1a 00 a0 00 60 02 00 07 50 2f ff f1 30 16 40 08 fc f4 ........
1e 80 d0 0b 41 9c d2 18 ad 00 44 18 87 5b 58 c3 50 56 57 8b ........
8a d0 5b 58 c3 80 fa 61 72 a8 0f a7 a7 70 58 0c 2e 0e b1 a8 ........
```

on the trie structure we find episodes from each of the executables as per the algorithm depicted in Figure 23. The top $M/2$ episodes in order of class-wise frequency are selected from each class. These are combined to get a set of relevant episodes of cardinality M with duplicates removed. This result in a vector space model of the dataset of size 500 rows $\times$ M columns.

For unbiased evaluation of our approach, we use stratified ten-fold cross-validation. In this cross-validation method, we partition the data into ten disjoint sets of equal size and select one partition as testing set and the remaining nine are combined to form the training set. This process is repeated 10 times.

As mentioned in the motivation part, we are interested to compare the performance of fixed length n-grams versus variable length n-grams as features for virus detection. We compare our method (we call it as EVLN, named from Episode based Variable Length N-gram method) with that of the method proposed by Kolter et al. [60] for two reasons. First, the method proposed by Kolter et al. is so far the best method in terms of performance for detection of generic viruses using n-grams. The other reason is that this method uses fixed length n-grams as features and information gain as the measure for feature selection. In order to have an unbiased comparison, we use the same dataset (250 benign and 250 virus executables collected by us) for the evaluation of both the methods.

For classification, we use the metaclassifier AdaBoost M1 with J48 as the base classifier. AdaBoost, short for Adaptive Boosting, is a meta-algorithm that can be used in conjunction with many other learning algorithms to improve their performance. AdaBoost is adaptive in the sense that subsequent classifiers built are tweaked in favor of those instances misclassified by previous classifiers. It is less susceptible to the overfitting problem than most learning algorithms [79]. We use the implementation of these classifiers available in WEKA [124] for

our study. WEKA [124] is a collection of machine learning algorithms implemented in Java and open sourced under the GPL.

4.6 Experimental Results and Discussions

For one run of the ten-fold cross validation, we discovered 216541 episodes from the benign class and 111533 episodes from the virus class. Some of the frequent episodes from both the benign and virus classes are shown in Table 30 and 31 respectively.

Table 30: A list of frequent episodes from the benign class

```
21 00
8d 45 f4
75 65
72 65 65 00
66 69
00 00 53 e8
ff 36
c8 10 00 01
89 7d fc 6a 01
5b 00 2d 00
57 57
44 00 69 00
40 50 52 4f 47 52 41 4d
```

Table 31: A list of frequent episodes from the virus class

```
c3 55 8b ec
57 9a
d1 e0
90 90 90
2e c7 06
07 1f
79 72 69
8b 5e 06
75 03
74 69 6f 6e
ff e8 6e ff a0
ff 86 54 ff 8a 86 51 ff 98
07 00
```

For each of the class V and B we extract relevant episodes taking the profile length M as 100. The result of the proposed method in terms of ROC curve is shown in Figure 24. It is evident from Figure 24 that the proposed method gives a 100% detection rate with a false positive rate of 11.2%.

For comparison purpose, we implement the fixed-length n-gram method proposed by

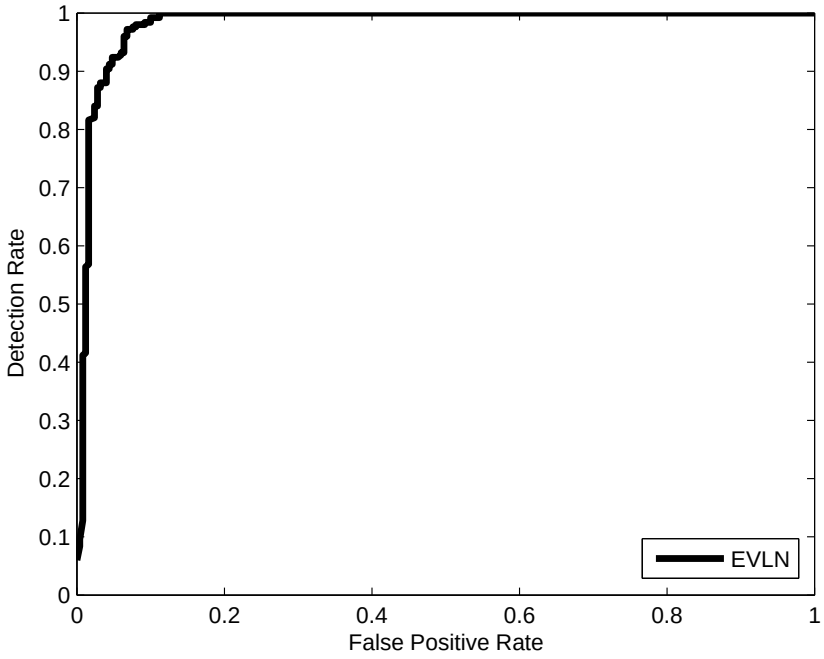


Figure 24: ROC curve of the proposed EVLN method for M=100

Kolter et al. [60] with n as 2, 3 and 4 and the profile length as 100. The ROC curves of these results are shown in Figures 25-28.

Figure 25 shows the ROC curves of the proposed method and the fixed-length n-gram approach (with $n = 2$) with profile length as 100. It is evident from the graph that the proposed EVLN method gives a 100% detection rate with a false positive rate of 11.2% whereas the fixed-length method attains it with a false positive rate of 21.6%. Even for fixed-length of n=3 and 4, EVLN method gives consistently better results in comparison to fixed-length n-gram approach as can be seen from Figures 26 and 27. The results along with area under the curve (AUC) are summarized in Table 32.

In intrusion detection research, it is usually recommended to keep the FPR as low as possible and a FPR of 1% is usually acceptable. Table 32, shows DR for FPR of 1.2%, the best case (DR and FPR that gives the highest F-measure value) and FPR for 100% DR. In a classification (or information retrieval) scenario, F-measure is used to find the overall results of the classification, which is defined in terms of *Precision* and *Recall* as,

$$\text{F-Measure} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{91}$$

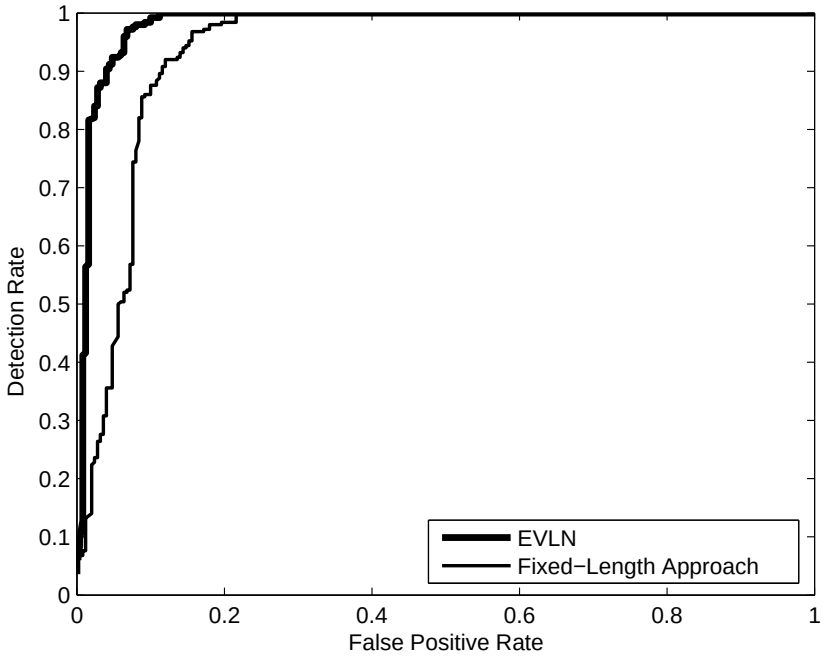


Figure 25: ROC curve of Fixed length n-gram vs EVLN method: Profile length = 100, n = 2 for fixed length n-grams.

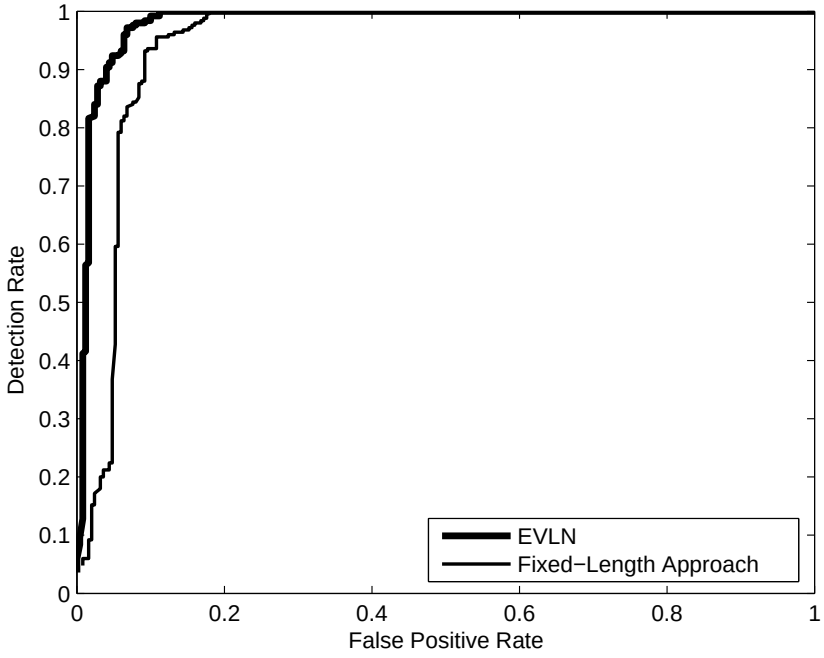


Figure 26: ROC curve of Fixed length n-gram vs EVLN method: Profile length = 100, n = 3 for fixed length n-grams.

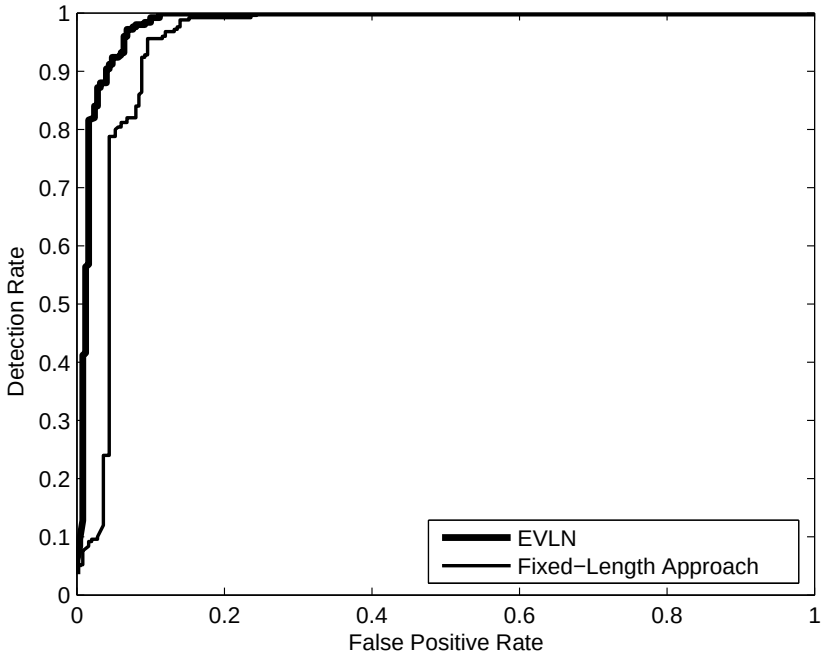


Figure 27: ROC curve of Fixed length n-gram vs EVLN method: Profile length = 100, n = 4 for fixed length n-grams.

Table 32: Summary of the results along with AUC values. The F-measure value with a star(*) is the highest F-measure for the corresponding method.

Methods	DR (%)	FPR(%)	F-Measure	AUC
EVLN method	100	11.2	0.94697	
	97.2	6.8	0.952941*	0.982232
	81.6	2.0	0.88889	
	56.4	1.2	0.715736	
Fixed length n-gram method ($n = 2$)	100	21.6	0.902527	
	96.8	15.6	0.911488*	0.93736
	22.4	2.0	0.360129	
	13.2	1.2	0.230769	
Fixed length n-gram method ($n = 3$)	100	18.0	0.917431	
	95.6	10.8	0.926357*	0.945368
	15.2	2.0	0.259386	
	6.0	1.2	0.11194	
Fixed length n-gram method ($n = 4$)	100	24.4	0.891266	
	95.6	9.6	0.931774*	0.94968
	9.6	2.0	0.172043	
	8.0	1.2	0.14652	

where Precision is defined as the number of viruses correctly classified divided by the total number of executables classified as viruses, and recall is defined as the number of viruses correctly classified divided by the total number of virus executables.

It can be seen that for all the cases, the EVLN method performs better than the fixed-length approach of Kolter et al. [60] consistently. The overall performance of the classifier in terms of area under the curve (AUC) as shown in Table 32 revels that EVLN method performs better than others.

We also carry out experiment by taking profile length as 500. The comparative ROC curve for the EVLN and Kolter method (with $n = 4$) is shown in Figure 28. For profile

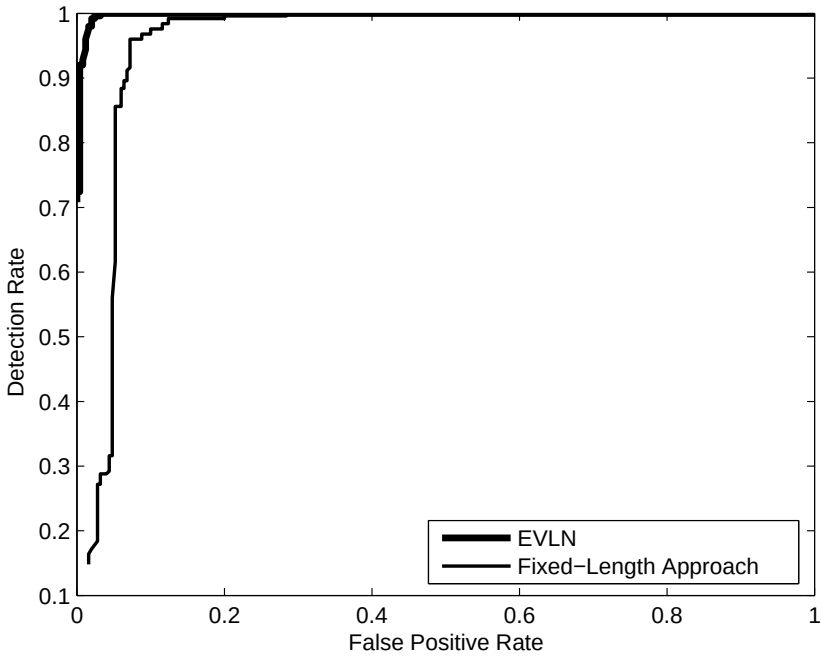


Figure 28: ROC curve of Fixed length n-gram vs EVLN method: Profile length = 500, n = 4 for fixed length n-grams.

length of 500, as given in Figure 28, the accuracy of our method is more pronounced than the fixed-length approach. The proposed method achieves a detection rate of 72.4% with zero false positive, and 100% detection with 3.2% false positive rate. For the same profile length fixed-length n-gram method, for n=4, shows the lowest detection rate of 2% with zero false positive rate and attains 100% detection rate with 28.4% false positive rate.

Taking area under ROC curve as performance criteria, from the visual inspection of the ROC curves it is clear that our proposed EVLN method outperforms the method proposed by Kolter et al. [60] in all cases. Based on the experimentation we infer the following. The variable-length n-grams approach is better than fixed length n-grams approach as the fixed-length n-grams do not capture the long meaningful sequences while keeping the size and number of n-grams manageable. Moreover it is also observed that possibly class-wise episode frequency is a better measure for feature selection than the information gain. This is also demonstrated in detail in the context of fixed length n-grams in a recent work by Reddy et al. [92].

When it comes to advanced computer viruses like polymorphic viruses, static analysis methods do not work. To tackle these viruses, static analysis methods should be combined with dynamic analysis methods for efficient detection. For example, polymorphic virus consists of three components – decryption routine, mutation engine and virus body. Since the mutation engine and the virus body are encrypted and decryption routine is different in each replication, it is not possible to directly apply any static analysis method (including our method) to detect this virus. Instead we can use a dynamic analysis technique (like sandboxing) that trick a polymorphic virus into decrypting and revealing itself [81]. On this decrypted virus we can use the static analysis method. In this work, we assume that a polymorphic virus must decrypt before it can execute normally.

4.7 Conclusion

The present study focuses to establish that proper feature extraction and selection technique can help in efficiently detecting virus programs. Our study shows that episodes as variable length n-grams is better than classic fixed length n-grams in efficiently discriminating virus executables from that of the benign ones. We also show that building a common trie structure for a class essentially captures the underlying frequency and entropy parameters for the whole class. Selecting frequent episodes in terms of class-wise relevance is shown to be a better feature selection technique. We demonstrate that a supervised classification by the proposed feature selection method gives better accuracy and less false positives in

comparison to earlier proposed methods.

In n-gram approach attaching semantic meaning for the relevant n-grams (episodes) is not yet explored by us. As a future task, we are further analyzing the variable length n-grams to find the semantic meaning attached to the n-grams, if any. Such knowledge can be used to develop a semantic aware method.

CHAPTER V

CONCLUSIONS

In this chapter we provide the summary of the thesis and discuss some of the future avenues of research that this thesis points to.

5.1 Summary

This thesis tries to answer some specific security problems observed in a host system using data mining techniques. In a nutshell, the thesis investigates the problem, stated earlier as,

- *How to extract correct features and model user behavior leading to an efficient masquerade detection.*
- *How to reduce the dimension of the input data without degrading the accuracy of the detection system.*
- *How to extract and select proper features for efficient classification of benign and malicious executables.*

A host based anomaly detection approach involves two important stages, First, profiling the normal behavior of the entity to be monitored and secondly, monitoring the entity for any deviations from these learned profile, which could be an intrusion. The entity to be monitored can be broadly categorised as either user(s) or program(s).

The main task in user profiling and monitoring is to address the problem of Masquerade attack, where one user impersonates as another. We provide an extensive study of methods available in the literature which addresses the problem of masquerade attack. We observe that most of the earlier methods work on users' command line data as the base for modeling behavior and detecting abnormality. While studying the users' command line data, we observe that the usage by a user consists of a sequence of commands rather than isolated commands. We call such a sequence of commands an *episode*.

Based on the voting-expert paradigm [**?**], we propose a novel and efficient way of finding episodes from users' command line data. We represent the command line data as a trie structure which facilitates in computing the entropy and frequency of n-grams of various lengths. Based on the entropy and frequency information, we break the data into episodes which may represent tasks or actions. Our proposed episode based masquerade detection technique finds episodes from user's data and classify each episode as normal or abnormal using Naive Bayes classifier. Based on overall strength of abnormal episodes in a command block, the block is detected as normal – belongs to same user or masquerade – generated by a different user. We test our method on the SEA dataset which is considered as a benchmark in masquerade detection research. The experimental results show that our method performs better than many earlier methods.

Based on our observation that a user may use multiple tasks simultaneously, thus, leading to temporal interleaving of episodes, we use constraint programming for user modeling and masquerade detection. We model a legitimate user as an Interval Algebra (IA) network by capturing the binary relationships between episodes in the user's command line data. We extract episodes from the training data of all users and select a few frequent ones to represent the nodes in the network. The network is built such that each node represents an episode of commands and binary relationship between each pair of episodes is encoded as the disjunction of the Allen's Interval relations. Given a test block of a user, we find the interleaving of episodes in the new block and build a network to represent the current behavior. We apply path consistency algorithm to check whether the new network is consistent with the profiled normal network, in which case it is classified as normal. Though the performance of the method on SEA dataset is not encouraging but it definitely gives a novel way of profiling and detecting abnormalities in user behavior.

We also propose a deferral model to detect masqueraders which delays the detection by few blocks. We use Naive Bayes technique to classify a block of commands as doubtful or normal. If a block is classified as doubtful, we increase the threshold of the legitimacy test condition and wait for subsequent blocks. This is because once we know that a block of commands is a questionable block, the following block would have a higher likelihood

of being doubtful. If the doubtful status continues for 2 to 3 consecutive blocks then our proposed method tags it as masquerade. One may point out that by deferring the process, it delays the detection of masqueraders. But in real life applications, the block size can be decided to be reasonably small depending on the context. Empirically we show that by reducing the block size from 100 to 50 or even 25, the results do not degrade much. In fact we get best result for block size of 50 in SEA dataset. We provide rigorous experimental results on all the standard datasets.

On program profiling and abnormal process detection scenario, we observe that the existing methods represent system calls data in either of the two formats – term frequency and decision table representations. This results in a high dimensional representation of the system calls data which may have a low dimensional embedding which cannot be captured by any linearity assumption. We use Locally Linear Embedding (LLE), a non-linear dimensionality reduction technique to map these high dimensional data into low dimensional space. Another way to look at this reduction is to maintain the information in the data while reducing the noise from it which may lead to better results. Given a new trace of system calls, we use incremental LLE to map it into a low dimensional embedding and use k-NN classifier to classify it as normal or abnormal. We provide two different algorithms for this whole process, based on the two different ways of representation of system calls data. The experimental results show that the performance of the algorithms in terms of accuracy are much better than earlier ones.

For detection of malicious executables, we observe that a classification technique is as good as the features extracted and the feature selection method employed. We extend our episode concept to collect relevant features from a class of executables. We extract byte n-grams from each executable and represent it as a trie structure. Trie structures of all the executables from a class (malicious or benign) are embedded together to represent trie of the whole class. Based on this trie structure we extract episodes from each executable and selects relevant episodes based on class-wise document frequency. This way we could capture relevant n-grams of variable lengths to profile a class. We use these relevant sets of episodes as features for classification of executables. We perform experiments on our

own collected dataset of benign and malicious executables and compare with fixed length n-gram approach. The experimental results prove our assumption.

In view of the above mentioned points, we believe that the thesis has addressed the problems stated in the beginning i.e. computing novel ways of extracting features, improving the accuracy of results for HIDS and efficient classification of malicious executables. If we combine the results of various methods proposed under this thesis, we can state that this thesis presents a study of detecting abnormal sequences of events of interest in the field of computer security.

We next discuss some of the possible future extensions of the work summarized above.

5.2 Future Work

- We extract episodes as frequent and meaningful patterns to capture and model user behavior. Identifying semantic meaning of such episodes may be done for profiling users based on tasks or actions.

- Most of the masquerade detection techniques use command line data of users. It would be interesting to study its applicability with other users' generated data such as users' mouse movement and click pattern, keystroke pattern, interaction with graphical interfaces etc.

- A feedback approach can be used to update the classifier, which may improve the results.

- We use LLE to reduce the dimension of the vector (term frequency and decision table representation) formed from system calls data. There are other non-linear dimensionality reduction techniques which can be studied to reduce the dimension.

- Though the reduction in dimension improves the accuracy of detection, but its trade-off with complexity can be analyzed.

- It will be interesting to study the properties of data for which LLE yields better results.

- We extract episodes as features and select relevant episodes, according to class wise episode frequency measure, to represent benign and virus class of executables. It would be interesting to apply this process to different malicious classes such as viruses, worms, trojan horses, back doors and spyware and identify relevant episodes for each class. This may help to correlate the basic property of each class with respect to the relevant episodes.

APPENDIX A

MANIFOLD LEARNING APPROACH FOR CLUSTERING CATEGORICAL DATA

A.1 Introduction

Dimensionality reduction is aimed at mapping high dimensional data points to a lower dimension preserving certain geometrical properties. When the data points are defined in a high-dimensional space, it is computationally intractable to apply data analysis or pattern recognition algorithms that require repeated computation of similarities or distances in the original data source. It is desirable, in such cases, that the similarity is preserved (at least in a neighborhood) while reducing the dimension so that several machine learning tasks like data visualization, clustering or classification can still be carried out at lower dimension. The goal of dimensionality reduction techniques is to map D-dimensional objects into d-dimensional objects, where $d \ll D$, minimizing some objective function such as *stress function* [85]. The research on dimensionality reduction is mostly concentrated on numerical data and there has not been much investigation on categorical data sets. The main reason is that it becomes easy to map a point in R^D to R^d retaining certain properties such as distance or angle in the Euclidean space. But it is hard to retain any such property of D-dimensional categorical data set to some other (may even be Euclidean) d-dimensional data set. Except for some few cases [85], [15], there is not much work in dimensionality reduction of categorical data. On the other hand, categorical data sets are invariably of large dimensions and dimensionality reduction is more relevant to such data sets.

Manifold Learning is a machine learning technique that learns the low dimensional manifold structure from the data. It is a non-linear dimensionality reduction technique based on the principle that every neighborhood of a point in high dimension can be embedded

into a linear manifold in a lower dimension. This technique maintains the local neighborhood geometry while embedding. The local geometry is traditionally based on Euclidean geometry in the sense that the distances in higher and lower dimensions are taken as Euclidean distance. Manifold learning technique cannot be trivially extended to categorical data as there is no obvious way to interpret Euclidean distance for categorical data. On the other hand, the basic definition of manifolds does not insist that the source data points should be in Euclidean space [111][80]. It identifies a nonlinear mapping of a neighborhood of the original space to a neighborhood in Euclidean space. A question comes naturally to mind that whether the manifold learning can be applied to categorical data set. The mathematical concept of manifold does not rule out this possibility.

We address the problem of learning the low dimensional manifold from categorical data set. We propose a method of dimensionality reduction for high dimensional categorical data. The proposed dimensionality reduction technique is an adaptation of Locally Linear Embedding (LLE) for categorical data. LLE as proposed originally by Saul and Roweis [96] depends on Euclidean distance among the neighbors. We show here that this assumption is critical for deriving the steps of LLE. On the other hand, for categorical data it is not possible to define Euclidean distance among the data items. We show here that by suitably modifying the LLE formulation with similarity metric for categorical data it is possible to attain dimensionality reduction. In order to justify the proposed technique, we demonstrate its application in clustering high dimensional categorical data. We show here that the categorical data can be clustered in a reduced dimension as efficiently as any of the well-known categorical clustering techniques. The proposed manifold learning technique for categorical data maps a D-dimensional data to a d-dimensional Euclidean space by locally embedding linear manifolds. It can be seen that this process maps a categorical data set to a numerical (Euclidean) data set preserving the local geometry. Thus, we use well-known traditional data clustering techniques instead of specialized categorical data clustering techniques. In that sense, we are also proposing a novel method of clustering of high dimensional categorical data by Manifold Learning.

A.2 Alternative Interpretation of Reconstruction Weights

We present here a combinatorial interpretation of the reconstruction weights. In original proposal of LLE, the weights are considered to be representing the contribution of neighbors to the linear reconstruction of the data point and in order to obtain the optimal set of weights, we minimize the error function (Equation 76). In the context of categorical data set, we view the weights as the contribution of influences of neighboring points. We formulate the problem of computation of weights as a system of linear equation that need to be satisfied for the set of neighbors to be consistent.

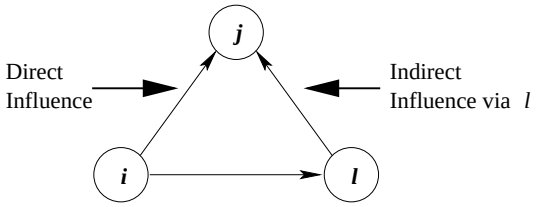


Figure 29: Direct and indirect influence of i on j

We view the set of data items as a network of influence where a node corresponds to data item and a link between nodes represents the influence of one data item on the other. Let us assume that every data point X_ℓ is influenced by its neighboring points. The link strength is proportional to similarity measure of the data items. We have two types of influences of one node on the other - direct and indirect. The direct influence of one node on another acts along the direct link between the nodes and it is a function of the similarity measure between the nodes. The indirect influence of a node on another via a set of nodes forming a path between the nodes is a function of similarity measures of all links in the path. For the present study, we restrict to paths of length two only. Thus, the influence of a data point X_i on X_j is obtained in two ways - directly from X_i and indirectly via other neighboring point X_ℓ as shown in Figure 29. We assume that the influence of one data point on the other is a function of similarity measure between data points. It may be noted that the usual concept of distance measure does not work well here. When the points are overlapping, the influence should be the highest and when they are far apart it should be decreasing. Thus, for a given data X_ℓ and set of its neighboring data items, the

direct influence of the neighbor X_i on X_j is sim_{ij}^2 and the indirect influence of X_i on X_j via X_ℓ is $sim_{i\ell}^2 + sim_{j\ell}^2$. There can be many ways of defining influence as a function of the similarity measures and we have chosen the square of the similarity measure in order to be consistent with the LLE formulation of the continuous data. There can be some decision problems in this scenario. We are interested in ensuring that X_ℓ plays a sort of balancing role of its neighbors. For this we have to determine the contributing factor of each of the neighbors of X_ℓ such that the resulting aggregated influence at every point is invariant. In other words, for every neighboring point X_i the difference of the total direct influences of all the neighboring points on X_i and the total indirect influences via X_ℓ on X_i should be invariant for all data points X_i. Mathematically, we assume that X_ℓ assigns weights $W_{\ell i}$ to each of its neighbors such that the weighted aggregate of direct influences on X_i of all neighbors of X_ℓ can be expressed as follows.

$$\sum_{j=1}^{K} W_{\ell j} sim_{ij}^2 \tag{92}$$

Similarly the indirect influence on X_i of all neighbors via X_ℓ can be expressed as follows.

$$\sum_{j=1}^{K} W_{\ell j} (sim_{i\ell}^2 + sim_{j\ell}^2) \tag{93}$$

If we assume that X_ℓ assigns normalized weights to all its neighbors such that the difference between the direct influence of each other and indirect influence via X_ℓ is a constant. Then,

$$\sum_{j=1}^{K} W_{\ell j} sim_{ij}^2 - \sum_{j=1}^{K} W_{\ell j} (sim_{i\ell}^2 + sim_{j\ell}^2) = const, \quad \forall i \tag{94}$$

This becomes

$$\sum_{j=1}^{K} W_{\ell j} (sim_{i\ell}^2 + sim_{j\ell}^2 - sim_{ij}^2) = \lambda, \quad \forall i \tag{95}$$

where λ is a constant.

Thus, the problem reduces to solving for $W_{\ell j}$ so that the above system of equations are satisfied. Without loss of generality, we can assume that the contributing factors of the neighbors are normalized and hence the sum of weights equals to 1. This also helps in deciding the value of the right hand side constant. It is also interesting to see that the computation of optimal reconstruction weights of LLE fits into this formalism. This

is essentially the system of equation $Q_\ell \mathbf{W}_\ell = \lambda e$, where e is the vector of 1's and the ij^{th} entry of Q_ℓ is $({sim_{\ell i}}^2 + {sim_{\ell j}}^2 - {sim_{ij}}^2)$.

Based on this interpretation, we can use the same LLE framework for categorical data and solve for $\mathbf{W}$'s according to Equation 79. We compute the Q matrix for categorical data sets using Equation (78) with sim_{ij} as similarity between X_i and X_j. The proposed method is formally stated below.

Step 1: For every pair X_i and X_j, compute similarity sim_{ij} as follows.

$$sim_{ij} = \frac{X_i \otimes X_j}{2D - (X_i \otimes X_j)}.$$

where $(X_i \otimes X_j) = \sum_{m=1}^{D} \delta(x_{im}, x_{jm})$ with $\delta(x_{im}, x_{jm}) = 1$ if $x_{im} = x_{jm}$ and 0 otherwise. x_{im} and x_{jm} are the m^{th} dimension values of X_i and X_j, respectively. Thus, $X_i \otimes X_j$ refers to the number of corresponding attributes having same value. For example, if we have two data objects as $X_1 = (\alpha,\ 0,\ a,\ sunny,\ 1)$ and $X_2 = (\beta,\ 0,\ a,\ rainy,\ 0)$, then $(X_1 \otimes X_2) = 2$, so $sim_{12} = \frac{2}{(10-2)} = \frac{1}{4}$.

Based on this similarity, K best neighbors of each data point are computed.

Step 2: For every x_ℓ, Q_ℓ matrix is determined using its neighbors as

$$Q_\ell(i,j) = {sim_{\ell i}}^2 + {sim_{\ell j}}^2 - {sim_{ij}}^2.$$

Step 3: Compute the weight matrix as

$$\mathbf{W}_\ell = \frac{\mathbf{Q}_\ell^- e}{e^T \mathbf{Q}_\ell^- e}$$

where $\mathbf{Q}_\ell^-$ is the generalized inverse of Q_ℓ.

Step 4: Construct the matrix $M = (I - \mathbf{W})^T \times (I - \mathbf{W})$. Find the lowest $d+1$ eigenvectors of M corresponding to the $d+1$ smallest eigenvalues. Ignoring the smallest eigenvalue, the remaining d eigenvectors give the embedding Y.

To have an intuitive feeling of this process, let us consider the following simple example. We generate a synthetic dataset, shown in Table 33, as per the procedure described in [19]. This dataset has 10 records and 10 categorical attributes. One can see that the data is specially generated to have 3 normal clusters.

Table 33: Sample data set

Rec. No.	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
1	3	3	1	0	0	0	0	0	0	0
2	3	1	3	0	0	0	0	0	0	0
3	1	4	1	0	0	0	0	0	0	0
4	0	0	0	4	2	1	0	0	0	0
5	0	0	0	4	3	2	0	0	0	0
6	0	0	0	2	3	1	0	0	0	0
7	0	0	0	0	0	0	4	2	2	3
8	0	0	0	0	0	0	2	3	4	3
9	0	0	0	0	0	0	1	4	3	2
10	0	0	0	0	0	0	4	2	1	1

The similarity of X_1 with X_i, $1 \leq i \leq 10$, sim_{1i} are 1, $\frac{8}{12}$, $\frac{8}{12}$, $\frac{4}{16}$, $\frac{4}{16}$, $\frac{4}{16}$, $\frac{3}{17}$, $\frac{3}{17}$, $\frac{3}{17}$, $\frac{3}{17}$. The five best neighbours (excluding self) are records 2, 3, 4, 5 and 6. We compute the matrix Q_1 as shown in Table 34.

Table 34: Q_1 matrix for record 1.

K-NN Rec. No.	2	3	4	5	6
2	-0.11111	0.66942	0.44444	0.44444	0.44444
3	0.66942	0.33884	0.66942	0.66942	0.66942
4	0.44444	0.66942	-0.77778	-0.22222	-0.22222
5	0.44444	0.66942	-0.22222	-0.77778	-0.22222
6	0.44444	0.66942	-0.22222	-0.22222	-0.77778

Similarly, for each of the record X_ℓ, corresponding Q_ℓ is computed. Following steps 3-4 corresponding vectors in 2 dimensional Euclidean space are computed. By applying the proposed dimensionality reduction technique for d=2 and K=5, the embedding in R^2 is shown in Table 35.

Figure 30 shows the points in 2D Euclidean space. One can see that the similar points are clustered together.

It is interesting to note that we can use this process for visualization of categorical data in a simple geometrical space. Moreover, we are able to map categorical data to a *numerical* dataset. The advantage of embedding a categorical data set on a numerical space can never be exaggerated. For instance, we can cluster the categorical data by any of the traditional clustering algorithms by simply embedding on Euclidean space. For example, once a categorical data set is locally embedded to a linear manifold on a Euclidean space of

Table 35: Reduced dimension of the sample data shown in Table 33 with d=2 and K=5

Rec. No.	RD_1	RD_2
1	0.11518	-0.53897
2	0.14846	-0.30961
3	0.097775	-0.51975
4	0.36873	0.30444
5	0.36873	0.30444
6	0.36873	0.30444
7	-0.42417	0.18405
8	-0.33727	0.076192
9	-0.30703	0.039062
10	-0.39913	0.1557

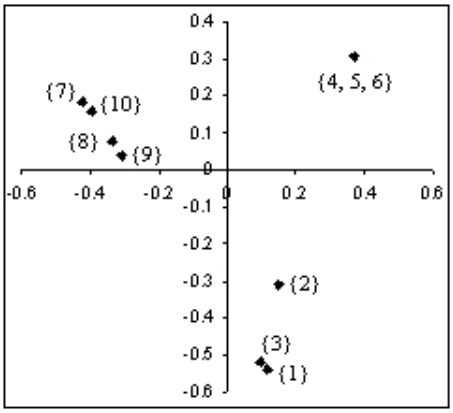


Figure 30: 2-D plot of the dimensionally reduced sample data shown in Table 35

lower dimension, we can make use of simple but powerful and well-known k-means clustering to cluster categorical data. Hence, one does not need any special clustering algorithm for categorical data. We emphasize here that in addition to dimensionality reduction this particular feature of embedding categorical data on numerical space is a major contribution of the present work. We demonstrate the power of this method on categorical clustering of high dimensional data in our experimental study.

Clustering techniques for categorical data are very different from those for numerical data in terms of the definition of similarity measure. Most numerical clustering techniques use distance functions, for example, Euclidean distance, to define the similarity measure, while there is no inherent distance meaning between categorical values. Initial attempts for clustering of categorical data were to adopt numerical clustering through the data preprocessing stage, where numerical features are extracted/constructed from the categorical data

[49]. This approach is not popular as meaningful numerical features or conceptual similarity are usually difficult to extract. It has been widely recognized that clustering directly on the raw categorical data is important for many applications. Therefore, there are increasing interests in clustering categorical data recently like, K-Modes [51], ROCK [47], CACTUS [42], STIRR [44], Coolcat [8], LIMBO [3], CLICK [86], QROCK [35], Subspace clustering [41]. We show that manifold learning for categorical data, as proposed here, can help to use simple and conventional clustering algorithms without compromising the clustering quality.

A.3 Experimental Study

We use certain real categorical data set available in UCI Machine Learning Repository [113] for experimentation. We use our dimensionality reduction technique by locally embedding a linear manifold to these data sets. The result of LLE is a dataset of reduced dimension in Euclidean space. We make use of k-means clustering algorithm for this embedded, dimensionally reduced data set. We compare our clustering result with some of the major well-known categorical clustering algorithms such as ROCK, CLICK etc.

We observe that our results are sometimes better and sometimes equally efficient as these algorithms. We give below the experimental results in terms of purity function and confusion matrix for each data set.

Soybean small data: The Soybean small data set has 47 instances each of which is described by 35 attributes, all being categorical. Each instance is labeled as one of the four diseases: Diaporthe Stem Canker (D), Charcoal Rot (C), Rhizoctonia Root Rot (R), and Phytophthora Rot (P). Except for Phytophthora Rot, which has 17 instances, all other diseases have 10 instances each. Table 36 summarizes the result of the proposed method, with d=2 and K=5, on soybean small data and finding clusters using k-means clustering with k=4.

Each C_i represents a cluster and D, C, R, P are the names of the original classes in the Soybean data. The misclassification table depicts that each cluster contains instances belonging to a particular class only, thus achieving 100% purity as against 93.62% achieved

Table 36: The misclassification matrix for the Soybean small data

	C_1	C_2	C_3	C_4
D	10	0	0	0
C	0	10	0	0
R	0	0	10	0
P	0	0	0	17

by subspace clustering method [41] and 100% by the best-K method [19]. The most important point is that even by reducing the dimension from 35 to 2, we can achieve full purity for this data set.

Congressional votes data: This data set contains the United States Congressional Voting Records in 1984. Each record corresponds to one congressman's votes on 16 issues (e.g., education spending, crime, etc.). All attributes are boolean valued with 1 or 0 (for 'Yes' and 'No' respectively). A classification label of 'Republican' or 'Democrat' is provided with each data record. It has 435 records (168 republicans and 267 democrats). A few of the attributes contains missing values represented by "?", which we treat as another domain value for the attribute.

We project the congressional votes data to R^5 by the proposed algorithm and apply k-means clustering algorithm. Table 37 gives the misclassification matrix for the dataset. We experiment with K=$\{10, 20\}$ but there is not much significant difference in the output of the clustering algorithm. We get a purity value of 88.74% by our method with K=20, d=5 and k=2, whereas ROCK yields purity of 79.31% [47] and subspace clustering gives purity of 91.95% [41]. For this data set, CLICK generates 12 clusters with a purity value of 95.5% [86].

Table 37: The misclassification matrix for the congressional votes data

	C_1	C_2
Democrat	40	227
Republican	158	10

(a) for K=10

	C_1	C_2
Democrat	38	229
Republican	157	11

(b) for K=20

	C_1	C_2	C_3	C_4
Democrat	172	37	27	31
Republican	4	0	143	21

(c) for K=10 with 4 clusters

Zoo data: Zoo data contains 101 animal instances each with 17 categorical attributes describing features of animals such as hair, fins, tail, etc., most of them being boolean. The records are grouped into seven classes according to animal type (e.g., mammal, bird, insect, etc.). By applying the proposed method with $d = 5$, $K = 10$ we get a purity value of 89.11% for k=7 and 95.05% for k=15. The results are summarized in Table 38.

Table 38: The misclassification matrix for the zoo data

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
A1	0	0	0	0	0	41	0
A2	6	0	0	0	0	0	14
A3	0	4	0	0	1	0	0
A4	0	0	0	0	13	0	0
A5	0	3	0	1	0	0	0
A6	0	0	8	0	0	0	0
A7	0	0	6	4	0	0	0

(a) for k=7

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}
A1	0	0	0	0	0	0	0	0	3	0	3	15	20	0	0
A2	4	0	0	0	0	0	12	0	0	4	0	0	0	0	0
A3	0	0	0	0	1	1	0	3	0	0	0	0	0	0	0
A4	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0
A5	0	2	0	1	0	0	0	1	0	0	0	0	0	0	0
A6	0	0	6	0	0	0	0	0	0	0	0	0	0	2	0
A7	0	0	0	1	0	0	0	0	0	0	0	0	0	2	7

(b) for k=15

Mushroom data: The mushroom data set contains 8124 records. Each record contains information describing 22 physical properties (e.g., color, odor, size, shape) of a single mushroom and also a label as either poisonous (3916 records) or edible (4208 records). All the 22 attributes are categorical, but 2480 records have missing values on the 11^{th} attribute, which we treat as another domain value for the attribute.

By reducing the dimensionality of the mushroom data to 5, with 10 neighbors and by applying k-means clustering we get purity value of 95.92% and 97.96% with 20 and 25 clusters respectively, as against the purity value of 99.6% obtained by ROCK with 21 clusters [47] and 99.3% by CLICK using 533 clusters [86]. The confusion matrix of our result is shown in Table 39.

Table 40 gives the purity values obtained by the proposed method for different datasets

Table 39: Confusion matrix of mushroom data

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
P	8.67	1.18	0.66	0.59	1.22	1.69	0.02	7.77	0	4.25
E	0.06	0	4.00	0	3.20	5.15	0	0	5.54	0
	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
P	0.05	4.51	4.68	2.36	3.55	4.07	0.40	3.77	2.36	0
E	2.80	0	0	0	0	0	22.18	0	0	5.27

(a) with 20 clusters

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
P	7.13	0	0	2.36	3.37	0.59	3.55	2.36	0.59	3.56
E	0	5.39	0.01	0	0	0.47	0	0	0	0
	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
P	1.18	0.20	0	3.14	3.18	7.26	2.93	0	0	1.95
E	24.81	0	0.44	0	0	0	0	0.09	5.54	0
	C_{21}	C_{22}	C_{23}	C_{24}	C_{25}					
P	1.18	0.39	0	6.88	0					
E	0	0.89	5.40	0	5.16					

(b) with 25 clusters

with different values of parametes (K, d and number of clusters). Table 41 shows a comparision of purity values obtained by different categorical clustering techniques with that of proposed method.

Table 40: Purity value obtained by the proposed method for different data sets with different values of parameters

Datasets	K, d	No. of clusters	Purity
Soybean small	5,2	4	100.00%
Congressional votes	10,5	2	88.51%
	20,5	2	88.74%
	10,5	4	88.00%
Zoo	10,5	7	89.11%
		15	95.05%
Mushroom	10,5	20	95.92%
		25	97.96%

As can be seen from the results obtained by using different methods on several datasets, none of the existing categorical clustering algorithm performs consistently better than our method. Though CLICK gives better purity value its granularity is very high and hence, it yields very large number of clusters.

Table 41: Purity value obtained by different categorical clustering techniques.

Dataset	Method	No. of clusters	Purity
Soybean small	Best K method	–	100.00%
	Subspace clustering	4	93.62%
	Our method	4	100.00%
Congressional votes	Best K method	–	83.00%
	CLICK	12	95.50%
	ROCK	2	79.31%
	Subspace clustering	2	91.95%
	Our method	2	88.74%
Zoo	Best K method	–	93.10%
	Our method	15	95.05%
Mushroom	CLICK	533	99.30%
	ROCK	21	99.60%
	Our method	25	97.96%

A.4 Conclusions

In the present work we have proposed a method of dimensionality reduction for the categorical data using manifold learning. To the best of our knowledge there has not been any attempt earlier in this direction. We provide a new interpretation of reconstruction weights which helps in applying locally linear embedding to categorical data. In this process we also propose a technique of converting categorical data to numerical data that preserves local similarity. We demonstrate experimentally that our dimensionality reduction approach is robust. We show that traditional k-means clustering of dimensionally reduced dataset can yield as efficient result as many of the well-known categorical clustering algorithms. The present work is an initial study and we plan to extend it further in different domain.

Publications from the Thesis

- Subrat Kumar Dash, Krupa Sagar Reddy and Arun K Pujari (2009). Adaptive Naive Bayes method for Masquerade Detection. Submitted to Journal of Security and Communication Networks (revised version submitted).

- Subrat Kumar Dash, D. Krishna Sandeep Reddy, Arun K Pujari (2009). New Malicious Code Detection Using Variable Length n-Grams. In Algorithms, Architectures and Information Systems Security (Indian Statistical Institute Platinum Jubilee Series), B.B. Bhattacharya, S. Sur-Kolay, S.C. Nandy, A. Bagchi (Eds.), World Scientific, New Jersey, pp.307-323.

- Subrat Kumar Dash, Sanjay Rawat, Arun K. Pujari (2007). Use of Dimensionality Reduction for Intrusion Detection. In P.D. McDaniel, S.K. Gupta (Eds.): Information Systems Security, Third International Conference, ICISS 2007 proceedings, LNCS 4812, Springer-Verlag, pp.306-320.

- Subrat Kumar Dash, Rasmikanta Pati, Chakravarthy Bhagvati, Arun K Pujari, Kuldip K. Paliwal (2007). Manifold Learning Approach for Clustering Categorical Data. In Proceedings of the National Conference on Computer Vision AI and Robotics (NCCVAIR07), Chennai, India, 3-5 Oct, 2007.

- D. Krishna Sandeep Reddy, Subrat Kumar Dash, Arun K. Pujari (2006). New Malicious Code Detection Using Variable Length n-grams. In A. Bagchi, V. Atluri (Eds.): Information Systems Security, Second International Conference, ICISS 2006 proceedings, LNCS 4332, Springer-Verlag, pp.276-288.

- Subrat Kumar Dash, Sanjay Rawat, Arun K. Pujari (2006). LLE on System Calls for Host Based Intrusion Detection. In IEEE Proc. of the International Conference on Computational Intelligence and Security (CIS'06), Guangzhou, China, Vol.1, pp.609-612.

- Subrat Kumar Dash, Krupa Sagar Reddy, Arun K. Pujari (2005). Episode Based Masquerade Detection. In S. Jajodia and C. Mazumdar (Eds.): Information Systems Security, ICISS 2005 proceedings, LNCS 3803, Springer-Verlag, pp.251-262.

- Subrat Kumar Dash, Sanjay Rawat, G. Vijaya Kumari, Arun K. Pujari (2005). Masquerade Detection Using IA Network. In Proc. of the CP 2005 Workshop on Applications of Constraint Satisfaction and Programming to Computer Security (CPSec), Barcelona, Spain, pp.18-30.

REFERENCES

[1] Allen, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM 26*, 11 (1983), 832–843.

[2] Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., and Keromytis, A. D. Detecting targeted attacks using shadow honeypots. In *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 129–144.

[3] Andritsos, P., Tsaparas, P., Miller, R. J., and Sevcik, K. C. LIMBO: Scalable clustering of categorical data. In *9th International Conference on Extending DataBase Technology (EDBT). Hellenic Database Symposium, 2003* (March 2004).

[4] Arnold, W., and Tesauro, G. Automatically generated win32 heuristic virus detection. In *Proceedings of the Virus Bulletin Conference* (September 2000).

[5] Assaleh, T. A., Cercone, N., Keselj, V., and Sweidan, R. Detection of new malicious code using n-grams signatures. In *Proceedings of the Second Annual Conference on Privacy, Security and Trust* (2004), pp. 193–196.

[6] Bace, R., and Mell, P. NIST special publication on intrusion detection system. Tech. Rep. SP800-31, NIST, Gaithersburg, MD, 2001.

[7] Balzer, R. M., and Goldman, N. M. Mediating connectors. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Workshop* (1999), pp. 73–77.

[8] Barbara, D., Li, Y., and Couto, J. COOLCAT: An entropy-based algorithm for categorical clustering. In *Proc. of the eleventh Int. Conf on Information and Knowledge Management* (2002), pp. 582–589.

[9] Beauquier, J., and Hu, Y. Intrusion detection based on distance combination. In *CESSE07* (Venice, Italy, November 2007), World Acacemy of Sciences, WAS.

[10] Belkin, M., and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput. 15*, 6 (2003), 1373–1396.

[11] Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M. M., Lavoie, Y., and Tawbi, N. Static detection of malicious code in executable programs. *Int. J. of Req. Eng.* (2001).

[12] Bertacchini, M., and Benitez, C. E. NCD based masquerade detection using enriched command lines. In *CIBSI* (Mar del Plata, Argentina, November 2007).

[13] Bertacchini, M., and Fierens, P. I. Preliminary results on masquerader detection using compression based similarity metrics. *Electronic Journal of SADIO 7*, 1 (1 2007), 31–42.

[14] Boschetti, F. Dimensionality reduction and visualization of geoscientific images via locally linear embedding. *Computers and Geosciences 31*, 6 (2005), 689–697.

[15] Buntine, W., and Jakulin, A. Applying discrete pca in data analysis. In *Proceedings of the 20th conference on Uncertainty in Artificial Intelligence* (Banff, Canada, 2004), pp. 59–66.

[16] Cai, D. M., Gokhale, M., and Theiler, J. Comparison of feature selection and classification algorithms in identifying malicious executables. *Computational Statistics & Data Analysis 51*, 6 (2007), 3156–3172.

[17] Cavnar, W. B., and Trenkle, J. M. N-gram based text categorization. In *Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR-94)* (1994), pp. 161–175.

[18] Chao, S., and Lihui, C. Feature dimension reduction for microarray data analysis using locally linear embedding. In *Proceeding of 3rd Asia-Pacific bioinformatics conference (APBC '05)* (2005), pp. 211–217.

[19] Chen, K., and Liu, L. The "best k" for entropy-based categorical data clustering. In *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management* (Santa Barbara, CA, June 2005), pp. 253–262.

[20] Chess, D. M., and White, S. R. An undetectable computer virus. In *Proceedings of Virus Bulletin Conference* (2000).

[21] Christodorescu, M., and Jha, S. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium* (2003), pp. 169–186.

[22] Cohen, F. Computational aspects of computer viruses. *Computers & Security 8*, 4 (1989), 297–298.

[23] Cohen, P. R., Heeringa, B., and Adams, N. M. An unsupervised algorithm for segmenting categorical timeseries into episodes. In *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery* (London, UK, 2002), Springer-Verlag, pp. 49–62.

[24] Cohen, W. W. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning* (1995), Morgan Kaufmann, pp. 115–123.

[25] Coull, S., Branch, J., Szymanski, B., and Breimer, E. Intrusion detection: A bioinformatics approach. In *19th Annual Computer Security Applications Conference* (Las Vegas, NV, December 2003), pp. 24–33.

[26] Coull, S. E., and Szymanski, B. K. Sequence alignment for masquerade detection. *Computational Statistics & Data Analysis 52*, 8 (April 2008), 4116–4131.

[27] Cox, M. F., and Cox, M. A. A. *Multidimensional Scaling*. Chapman & Hall, London, UK, 2001.

[28] DARPA dataset. MIT Lincoln Laboratory. `http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html`.

[29] Davison, B. D., and Hirsh, H. Predicting sequences of user actions. In *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis* (Madison, WI, July 1998), AAAI Press, pp. 5–12.

[30] Debar, H., Dacier, M., Nassehi, M., and Wespi, A. Fixed vs. variable-length patterns for detecting suspicious process behavior. In *Proceedings of the 5th European Symposium on Research in Computer Security (ESORICS'98)* (1998), vol. 1485 of *LNCS*, Springer Verlag, pp. 2–15.

[31] Dechter, R. *Constraint Processing*, 1 ed. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, 2003.

[32] Denning, D. E. *Internet besieged: countering cyberspace scofflaws.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998, ch. Cyberspace attacks and countermeasures, pp. 29–55.

[33] Donoho, D. L., and Grimes, C. Hessian eigenmaps: Locally linear embedding techniques for highdimensional data. In *Proceedings of the National Academy of Sciences* (2003), vol. 100, pp. 5591–5596.

[34] DuMouchel, W. Computer intrusion detection based on bayes factors for comparing command transition probabilities. Technical Report 91, National Institute of Statistical Sciences, 1999.

[35] Dutta, M., Mahanta, A. K., and Pujari, A. K. QROCK: A quick version of the ROCK algorithm for clustering of categorical data. *Pattern Recognition Letters 26*, 15 (2005), 2364–2373.

[36] Egele, M., Kruegel, C., Kirda, E., Yin, H., and Song, D. Dynamic spyware analysis. In *ATC'07: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference* (2007), USENIX Association, pp. 1–14.

[37] Fodor, I. K. A survey of dimension reduction techniques. Technical Report, UCRL-ID- 148494, LLNL, 2002.

[38] Forman, G. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res. 3* (2003), 1289–1305.

[39] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (Los Alamitos, CA, 1996), pp. 120–128.

[40] Furnkranz, J. A study using n-gram features for text categorization. Tech. Rep. OEFAI-TR-9830, Austrian Research Institute for Artificial Intelligence, 1998.

[41] Gan, G., and Wu, J. Subspace clustering for high dimensional categorical data. *ACM SIGKDD Explorations Newsletter 6*, 2 (2004).

[42] Ganti, V., Gehrke, J., and Ramakrishnan, R. CACTUS—Clustering categorical data using summaries. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (1999), ACM, pp. 73–83.

[43] Gartner, Inc. Gartner survey ranks viruses and worms as top IT security threats. http://www.gartner.com/press_releases/asset_129199_11.html, June 15, 2005.

[44] Gibson, D., Kleinberg, J., and Raghavan, P. Clustering categorical data: an approach based on dynamical systems. In *Proc. of the 24th VLDB Conference* (New York, USA, 1998).

[45] Greenberg, S. Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Canada, 1988.

[46] Grzymala-Busse, J. W. A new version of the rule induction system LERS. *Fundam. Inf. 31*, 1 (1997), 27–39.

[47] Guha, S., Rastogi, R., and Shim, K. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems 25*, 5 (2000), 345–366.

[48] Hadid, A., Kouropteva, O., and Pietikainen, M. Unsupervised learning using locally linear embedding: Experiments with face pose analysis. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR)* (2002), vol. 1, pp. 111–114.

[49] Han, J., and Kamber, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, March 2006.

[50] Hofmeyr, S. A., Forrest, S., and Somayaji, A. Intrusion detection using sequences of system calls. *Journal of Computer Security 6* (1998), 151–180.

[51] Huang, Z. A fast clustering algorithm to cluster very large categorical datasets in data mining. In *Proc. ACM SIGMOD Workshop on Data Mining and Knowledge Discovery* (May 1997), pp. 146–151.

[52] Jain, V., and Saul, L. Exploratory analysis and visualization of speech and music by locally linear embedding. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004 (ICASSP'04)* (2004), vol. 3, pp. 1111111–9999999.

[53] Jha, S., Kruger, L., Kurtz, T., Lee, Y., and Smith, A. A filtering approach to anomaly and masquerade detection. Tech. rep., University of Wisconsin, Madison, 2006.

[54] Jiang, G., Chen, H., Ungureanu, C., and Yoshihira, K. Multi-resolution abnormal trace detection using varied-length n-grams and automata. In *Proceedings of the Second International Conference on Automatic Computing (ICAC'05)* (2005), IEEE Computer Society, pp. 111–122.

[55] Jolliffe, I. T. *Principal Component Analysis*, 2nd ed. Springer Series in Statistics. Springer, NY, 2002.

[56] Ju, W., and Vardi, Y. A hybrid high-order markov chain model for computer intrusion detection. Technical Report 92, National Institute of Statistical Sciences, 1999.

[57] Kephart, J. O., Sorkin, G. B., Arnold, W. C., Chess, D. M., Tesauro, G. J., and White, S. R. Biologically inspired defenses against computer viruses. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'95)* (1995), Morgan Kaufmann, pp. 985–996.

[58] Killourhy, K. S., and Maxion, R. A. Learning from a flaw in a Naive-Bayes masquerade detector. Tech. rep., Dependable Systems Laboratory, Carnegie Mellon University, year.

[59] Kim, H.-S., and Cha, S.-D. Empirical evaluation of SVM-based masquerade detection using UNIX commands. *Computers & Security 24*, 2 (March 2005), 160–168.

[60] Kolter, J. Z., and Maloof, M. A. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), ACM, pp. 470–478.

[61] Kosoresow, A. P., and Hofmeyr, S. A. Intrusion detection via system call traces. *IEEE Software 14*, 5 (1997), 35–42.

[62] Kouropteva, O., Okun, O., and Pietikinen, M. Incremental locally linear embedding. *Pattern Recognition 38*, 10 (Oct 2005), 1764–1767.

[63] Kullback, S., and Leibler, R. A. On information and sufficiency. *The Annals of Mathematical Statistics 22*, 1 (1951), 79–86.

[64] Lane, T., and Brodley, C. E. Approaches to online learning and concept drift for user identification in computer security. In *The Fourth International Conference on Knowledge Discovery and Data Mining* (New York, August 1998), pp. 259–263.

[65] Lee, D. D., and Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature 401*, 6755 (1999), 788–791.

[66] Lee, W., and Stolfo, S. J. Data mining approaches for intrusion detection. In *In Proceedings of the 7th USENIX Security Symposium* (1998).

[67] Lee, W., Stolfo, S. J., and Chan, P. K. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management* (1997), AAAI Press, pp. 50–56.

[68] Li, M., and Vitanyi, P. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 1997.

[69] Liao, Y., and Vemuri, V. R. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security 21*, 5 (Oct 2002), 439–448.

[70] Lo, R. W., Levitt, K. N., and Olsson, R. A. MCF: A malicious code filter. *Computers & Security 14*, 6 (1995), 541–566.

[71] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. Text classification using string kernels. *Journal of Machine Learning Research 2* (2002), 419–444.

[72] Loeb, V. Spy case prompts computer search. The Washington Post, 05 March 2001. page A01.

[73] Marceau, C. Characterizing the behavior of a program using multiple-length n-grams. In *Proceedings of the workshop on New security paradigms* (2000), ACM, pp. 101–110.

[74] Max-Perera, C., Posadas, R., Nolazco, J. A., Monroy, R., Soberanes, A., and Trejo, L. An improved non-negative matrix factorization method for masquerade detection. In *Proceedings of the 1st Mexican International Conference on Informatics Security, MCIS 2006* (2006).

[75] Maxion, R. A., and Townsend, T. N. Masquerade detection using truncated command lines. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks* (2002), IEEE Computer Society, pp. 219–228.

[76] McGraw, G., and Morrisett, G. Attacking malicious code: a report to the infosec research council. *IEEE Software 17*, 5 (September/October 2000), 33–41.

[77] McHugh, J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur. 3*, 4 (2000), 262–294.

[78] Miller, P. Hexdump. http://www.pcug.org.au/millerp/hexdump.html, 2000.

[79] Mitchell, T. M. *Machine Learning*. McGraw-Hill, New York, NY, 1997.

[80] Munkres, J. R. *Topology: A First Course*. Prentice-Hall, New Jersey, 1975.

[81] Nachenberg, C. Understanding and managing polymorphic viruses. *The Symantec Enterprise Papers 30* (1996), 1–13.

[82] Needleman, S., and Wunch, C. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology 48* (1970), 443–453.

[83] Oka, M., Oyama, Y., Abe, H., and Kato, K. Anomaly detection using layered networks based on eigen co-occurrence matrix. In *Proceedings of Seventh International Symposium on Recent Advances in Intrusion Detection (RAID'2004)* (2004), vol. 3224 of *LNCS*, Springer, pp. 223–237.

[84] Oka, M., Oyama, Y., and Kato, K. Eigen co-occurrence matrix method for masquerade detection. In *Proceedings of the 7th JSSST SIGSYS Workshop on Systems for Programming and Applications (SPA)* (2004).

[85] Oliveira, S., and Zaiane, O. Privacy-preserving clustering by object similarity-based representation and dimensionality reduction transformation. In *Workshop on Privacy and Security Aspects of Data Mining (PSDM'04)* (Brighton, November 2004), pp. 21–30.

[86] Peters, M., and Zaki, M. CLICK: Clustering categorical data using K-partite maximal cliques. Technical Report CS TR 04-11, RPI, 2004.

[87] Power, R. *Current and Future Danger: A CSI Primer on Computer Crime and Information Warfare*, 3rd ed. Computer Security Institute, 1999.

[88] Raftery, A. E. A model for high-order Markov chains. *Journal of the Royal Statistical Society. Series B (Methodological) 47* (1985), 528–539.

[89] Rawat, S., Gulati, V. P., and Pujari, A. K. A fast host-based intrusion detection system using rough set theory. *T. Rough Sets* (2005), 144–161.

[90] Rawat, S., Gulati, V. P., Pujari, A. K., and Vemuri, V. R. Intrusion detection using text processing techniques with a binary-weighted cosine metric. *Information Assurance and Security 1*, 1 (2006), 43–50.

[91] Rawat, S., Pujari, A. K., and Gulati, V. P. On the use of singular value decomposition for a fast intrusion detection system. *Electr. Notes Theor. Comput. Sci. 142* (2006), 215–228.

[92] Reddy, D. K. S., and Pujari, A. K. N-gram analysis for new computer virus detection. *Journal in Computer Virology 2*, 3 (2006), 231–239.

[93] Reddy, K. S. Masquerade detection by mining UNIX command line data. Master's thesis, Department of Computer & Information Sciences, University of Hyderabad, 2005.

[94] Ridder, D. D., and Duin, R. P. Locally linear embedding for classification. Technical Report PH-2002-01, Pattern Recognition Group, Dept. of Imaging Science & Technology, Delft University of Technology, Delft, The Netherlands, 2002.

[95] Rigoutsos, I., and Floratos, A. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics 14* (1998), 55–67.

[96] Roweis, S. T., and Saul, L. K. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science 290*, 5500 (2000), 2323–2326.

[97] Saul, L. K., and Roweis, S. T. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *J. Mach. Learn. Res. 4* (2003), 119–155.

[98] Schlkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation 13*, 7 (2001), 1443–1471.

[99] Schonlau, M., DuMouchel, W., Ju, W.-H., Karr, A. F., Theus, M., and Vardi, Y. Computer intrusion: Detecting masquerades. *Statistical Science 16*, 1 (2001), 58–74.

[100] Schonlau, M., and Theus, M. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters 76* (2000), 33–38.

[101] Schultz, M. G., Eskin, E., Zadok, E., Bhattacharyya, M., and Stolfo, S. J. MEF: Malicious Email Filter - A UNIX mail filter that detects malicious windows executables. In *Proceedings of USENIX Annual Technical Conference - FREENIX Track* (2001), pp. 245–252.

[102] Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy* (2001), IEEE Computer Society, pp. 38–49.

[103] Seo, J., and Cha, S. Masquerade detection based on SVM and sequence-based user commands profile. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security* (March 2007), ACM New York, NY, USA, pp. 398–400.

[104] Sharma, A., and Paliwal, K. K. Detecting masquerades using a combination of nave bayes and weighted RBF approach. *Journal in Computer Virology 3*, 3 (2007), 237–245.

[105] Sharma, A., Pujari, A. K., and Paliwal, K. K. Intrusion detection using text processing techniques with a kernel based similarity measure. *Computers & Security 26*, 7-8 (2007), 488–495.

[106] Smaha, S. E. Haystack: An intrusion detection system. In *The Fourth IEEE Aerospace Computer Security Applications Conference* (Orlando, Florida, December 1988).

[107] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology 147* (1981), 195–197.

[108] Szor, P. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[109] Tan, A. Flawed symantec update cripples chinese PCs. http://news.cnet.com/Flawed-Symantec-update-cripples-Chinese-PCs/2100-1002_3-6186271.html, May 2007. (Last accessed: 13 June 2009).

[110] Tenenbaum, J. B., de Silva, V., and Langford, J. C. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science 290*, 5500 (2000), 2319–2323.

[111] Thorpe, J. A. *Elementary Topics in Differential Geometry*. Springer Verlag, New York, 1979.

[112] Turing, A. M. On computable numbers: With an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society* (1936), vol. 42 of *2*, Mathematical Society, pp. 230–265.

[113] UCI. Machine Learning Repository. http://www.ics.uci.edu/mlearn/MLRepository.html.

[114] UNM dataset. `http://www.cs.unm.edu/~immsec/systemcalls.htm`.

[115] Violino, B. The security facade: Are organizations doing enough to protect themselves? This year's IW/Ernst & Young survey will shock you. *Information Week* (October 21 1996), 36–48.

[116] Vishwanathan, S., and Smola, A. Fast kernels for string and tree matching. *Advances in Neural Information Processing Systems 15* (2003), 569–576.

[117] VX Heavens. `http://vx.netlux.org`.

[118] Wagner, R. A., and Fischer, M. J. The string-to-string correction problem. *Journal of the ACM (JACM) 21*, 1 (January 1974), 168–173.

[119] Wang, K., and Stolfo, S. J. One-class training for masquerade detection. In *Proceedings of the 3rd IEEE International Conference on Data Mining Workshop on Data Mining for Security Applications (DMSEC)* (Nov 2003).

[120] Wang, R. Flash in the pan? *Virus Bulletin* (July 1998).

[121] Wang, W., Guan, X., and Zhang, X. Profiling program and user behaviors for anomaly intrusion detection based on non-negative matrix factorization. In *43rd IEEE Conference on Decision and Control* (December 2004), vol. 1, pp. 99–104.

[122] Warrender, C., Forrest, S., and Pearlmutter, B. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy* (1999), IEEE Computer Society, pp. 133–145.

[123] Wespi, A., Dacier, M., and Debar, H. Intrusion detection using variable-length audit trail patterns. In *Proceedings of RAID 2000* (London, UK, 2000), LNCS#1907, Springer-Verlag, pp. 110–129.

[124] Witten, I. H., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, CA, 2000.

[125] Witten, I. H., Moffat, A., and Bell, T. C. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[126] Yang, M., Zhang, H., and Cai, H. J. Masquerade detection using string kernels. In *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom 2007)* (Sep 2007), pp. 3681–3684.