PROBABILISTIC Vs CLUSTERING ANALYSIS OF MODIFIED UNIX COMMAND LINES FOR MASQUERADE DETECTION

BY

KARTHIC GUNASEKARAN

Bachelor of Engineering

Madras University

Chennai, India

2002

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfilment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 2005

PROBABILISTIC Vs CLUSTERING ANALYSIS OF

MODIFIED UNIX COMMAND LINES FOR

MASQUERADE DETECTION

Thesis Approved:

Dr. JOHNSON P. THOMAS
Thesis Advisor

Dr. G.E.HEDRICK

Dr. DEBAO CHEN

Dr. GORDON EMSLIE
Dean of the Graduate College

ACKNOWLEDGMENT

Many people have ambitions in life but only few are fortunate to fulfil them. In this regard I am greatly thankful to the almighty for guiding me in the right direction and rewarding my efforts. At this moment I must mention some important people who have always been a cornerstone for the successful completion of my master's thesis. Firstly, I would like to thank my thesis advisor Dr. Johnson P. Thomas for his valuable time, guidance, ideas and comments on my thesis. His timely advice on formatting my draft was really helpful in making this document complete. Secondly, my heartfelt thanks to my committee members, Dr. G.E.Hedrick and Dr. Debao Chen, for providing their comments and suggestions. I also appreciate Dr.G.E.Hedrick's patience and efforts for correcting my thesis draft and providing more experienced suggestions for building this document to perfection. Dr. Debao Chen's timely suggestions on some of my document's formats were also immensely helpful. I would also like to thank Dr. Istvan Jonyer for his presence during my final thesis presentation.

I am grateful to all these people for their advices and friendly support in all the stages of my thesis development. I am surely blessed with the right people's support and friendship at the right time. I also thank Oklahoma State University for having provided right technical facilities for presentation of my thesis work. I can never forget the Edmond Low Library that showed the path to my successful thesis completion.

I would never miss my parents at any stage of my life. Dad and Mom thanks for providing your moral and financial support to complete my Masters degree in Computer Science, in the USA. Last but not the least; I would like to thank all of my friends for having encouraged me during the entire Masters Degree program.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

1.1 Masquerade Detection

A computer system masquerader is an intruder who takes over a genuine user session and misuses it. Masqueraders occupy a privileged user's seat and exploit their privileges to access secure programs and data. Session take over by a masquerader can be due to careless behavior of the legitimate user, who mindlessly skips either to lock the system or to logout temporarily before going on a break. Such a careless act by the proper user creates an opportunity for hackers to misuse secure computer systems. Masqueraders typically look for secure files, software and privileges that they cannot use. A well-known instance of masquerader activity is the case of Robert. P. Hanssen, the FBI mole who allegedly used agency computers to ferret out information later sold to his conspirators [11]. Hanssen was an insider who abused his privileges to make money.

Masquerade detection software is therefore essential for secure systems as in defense and banking which contain highly confidential information that must be shielded against hackers, especially masqueraders. Masquerade detection is a challenging problem, which people have been trying to solve since 1988. Numerous attempts targeted at building masquerade detection systems have resulted in failure. User profiles were constructed from system-log data, which contained information such as time of login, physical location of login, duration of user session, cumulative CPU time, commands issued, programs executed, names of files accessed and so forth [9] [3]. This log information was used as training data to build user profiles. If a deviation was noted

while testing new data for a user with his already existing user profile, a masquerade attempt was reported.

1.2 Simulating a Masquerade Detection System

One of the major challenges facing Masquerade detection systems is real-time detection. A real-time Intrusion Detection System (IDS) to detect masquerades would analyze a user's log file with various fields like commands, time of login, location of login, to name a few. Simulating or implementing real-time IDS is a huge task and requires a detailed study of many components. Even if such a huge system is simulated or implemented, there would still be false alarms if the data were noisy. Hence, researchers started focusing on the components of IDS to perform analysis and reduce false alarms as much as possible.

Schonlau et al. [14] analyzed user log files that contained truncated UNIX commands for 70 users. They proposed four statistical and two machine-learning techniques to analyze users command log files. The best result they reported had 70% hit rate and 6.7% false alarm rate. Maxion and Townsend [12] later proposed a Naïve Bayes text classification analysis on Schonlau's data and recorded a hit rate of 61% for a false alarm rate of 1.3%. Finally, Yung [18] proposed a Self-Consistent Naïve Bayes technique that lowered the missing alarm rate by 40% for a false alarm rate of 1.3%. Yung's model classified the test sessions offline, in other words, the test sessions were classified only after the final test session arrives. He justified that offline classification has very low missing alarms, which was the main purpose of Masquerade detection system. It is better

to classify a session as masquerade session with a delay rather than classifying it as proper session. However, the disadvantage in this scheme is that by the time the masquerader is detected it may be too late as the damage may already be inflicted. For an effective detection system, the detection must be achieved as quickly as possible, ideally in real time.

1.2.1 Bayesian Networks and Clustering schemes for masquerade detection

Though offline evaluation of sessions produces more accurate results, the need for online classification cannot be substituted with offline techniques. This thesis first presents a Naïve Bayes classifier with deferred or n-test sessions lag classification technique. Our deferred Naïve Bayes classifier is also an offline classifier like the one, which Yung proposed, but it has a wait on varying number of test sessions before classifying them. This technique will be called a Deferred Naïve Bayes Classifier in the rest of this thesis.

Online classification immediately classifies a test session and an offline classifier produces a more accurate classification. To use the advantages of both online and offline classification techniques we propose a novel Bayesian networks approach to toggle between a Naïve Bayes Classifier and a Deferred Naïve Bayes Classifier. The Bayesian networks computes trust values on the simple Naïve Bayes classifier after classifying each test session. While the sessions are classified online using Naïve Bayes classifier, the Deferred Naïve Bayes classifier also works in parallel for each test sessions. If the trust on the Naïve Bayes classifier is less than a threshold value, then the Bayesian

networks signals the Deferred Naïve Bayes classifier to classify all the test sessions offline after it updates its classifier model with the latest test session. The latest session, which was not classified by the Naïve Bayes classifier, will now be classified by the Deferred Naïve Bayes classifier. After this classification the current classifier model of the Naïve Bayes classifier is replaced by the classifier model of Deferred Naïve Bayes classifier. The proposed cross technique update approach has several advantages. Firstly, the classification is online resulting in much faster detection. Secondly, our approach increases the accuracy of Naïve Bayes classifier on the classification of future sessions. Furthermore, the trust on this new Naïve Bayes Classifier is reset to the maximum and the classification of future test sessions proceeds online.

Chapter 4 introduces a clustering approach to train and test commands sessions. This approach was initially used in data mining field to cluster real time data and compute their likelihood patterns based on the formed clusters [5]. We propose an online clustering approach, which is a variant of the data mining approach presented in [5]. As far as we are aware, this clustering approach is novel to user command level masquerade detection scenario. Clustering introduces several advantages. These include a possible data reduction during the testing phase, simplicity of implementation and the ability to adapt to noisy data, which is the main reason for its success in data mining and pattern matching areas.

Our Clustering approach is further classified into three techniques based on the sequence rating mechanism in the testing phase. These are called Last n, Weighted Means and Decayed Weights Clustering analysis techniques. These three techniques are not new to clustering scenario but are indeed novel to masquerade detection. We intend

to test the feasibility of our clustering approach in real time to detect user command level masquerades and also compare the results of this approach and Bayesian networks scheme in this thesis.

Schonlau et al's user command log files [7] were used in all our experiments. The main objective of these analyses was to reduce the false and missing alarm rates and try to achieve better results than previously proposed techniques. The three clustering techniques and the Bayesian networks scheme were compared based on their results. A ROC curve for these techniques was drawn and a detailed discussion based on this curve is presented in section 5.3 for comparing the two schemes. The rest of the thesis is outlined below.

In chapter 2 we present a review of previous work in masquerade detection; chapter 3 explains both the Deferred Naïve Bayes and Bayesian networks Classifier schemes for masquerade detection. In chapter 4 we present our clustering approach to masquerade detection. Chapter 5 explains the simulators that we built to detect masquerades and compares the results of Bayesian networks and Clustering approach by the means of ROC (Relative Operating Characteristic) curve. Chapter 6 concludes our research with potential future work.

CHAPTER II

LITERATURE REVIEW

2.1 Early Research

Several attempts to tackle the problem of masquerade detection have been proposed. The first system was IDES (Intrusion detection Expert Systems), which dated to the middle of 1990's. The problems solved by IDES were misuse and anomaly detection. To detect misuse an expert system was built to identify a set of vulnerabilities that could be taken advantage of by the user. Such a system was very effective and was able to alarm lot of misuses, but one major drawback is that it is not sensitive to novel attacks. New vulnerabilities would have greater damaging impact and can open up many ways of attacking systems. In order to detect such novel attacks another statistical or pattern recognition system was built along with the expert system to identify deviant behavior from the normal ones [2]. Such statistical classifiers or pattern recognizers combined with the expert systems formed a complete IDES.

2.1.1 Audit Trails for threat identification

Auditing is a methodical examination of a subject's behavior while reviewing their current moves with already existing trails or acceptable standards. Anderson was the first to study the feasibility of automated audit trail analysis. He categorized the threats that could be addressed by audit trail analysis as the following [16] [1]:

- External penetrators (who are not authorized to use the computer)

- Internal penetrators (who are authorized use of the computer but are not authorized for data, program or resource accessed), including the following:

  -Masqueraders (who operate under another user's id and password).

  -Clandestine users (who evade auditing access controls).

- Misfeasors (authorized users of the computer and resources who misuse their privileges).

Masqueraders can be detected by observing departures from established patterns of use for individual users.

2.1.2 Measures for detecting Intrusions

There are two types of measures for detecting intrusions in a computer system. These are: a) Continuous measures and b) Discrete measures. Continuous measures take into account the CPU Time, amount of memory used, protection violations and I/O Time. Discrete measures are concerned with finite set quantities such as time of login and location of login. We outline some of these discrete and continuous measures.

- Continuous Measures: A continuous measure is a function of some aspect of a user session. This aspect has its effect on the system's resource usage. Some of the continuous measures are as follows.
  - Connect time – This is the length of the user session on the target system. Each session is referred to as the job and has a unique job number.
  - CPU Time – It is the amount of processing time consumed in each of the user session on the target system.

- IO Activity – This is the amount of I/O activity in a given user session.
- Protection violations – It is the number of directory and file access violations in a user session on the target system.

- Discrete Measures: A discrete measure is a function of user session whose range of value is a finite unordered set that is used to characterize some aspect of user behavior. Some of the discrete measures are:
  - Time of Login – Time of login is the number of sessions for each user for a time period of the day.
  - Location of login – It is the number of sessions for each user initiated from a particular physically identifiable location.

2.2 Hypothesis

An audit record contains information related to an action performed by the user. Each measure of the impact of an action on the system is logged as a field of the audit record. We view each entry in the audit record as a variable reflecting a part of the user behavior. There are several levels of monitoring defined by a number of researchers. One of the foremost levels that gained more attention is the **command level.**

A **command level** takes into account the various commands issued by the user during a session. We maintain a sequential relationship between the commands issued by a specific user in a given session. Command usage depends upon the role of the users. So a user with a specific role in an organization has privileges to some commands and

restriction to others. The concept here is that after some elapsed period of experience every user gets set into a groove of using a set of commands in almost the same order. In other words it becomes a habit for the user to use a set of commands every time he logs into a system. Such habitual use of commands give us the necessary pattern that we are looking for to establish a user's character.

For example in a session user "A" opens directory1 only once to check its contents. The reason may be that it contains some important files that are required by his agency. If a masquerader opens this directory many times and also tries to access the files in it, then it can be marked an intrusion.

Moreover, based on his work style the user may start off with checking his mail and reading through some newspapers after logging in; this is a habit for the user and it should occur every time when he logs into the system. If an intruder has the system in his hands then he overlooks these petty patterns that are learnt by the system and he tries to accomplish his task by checking for the file locations. This is an easy case of intruder detection scenario.

Some user command types that will be considered in this thesis are:

- Multi-Threaded process commands
- Mailer usage commands
- File/Directory listing commands

2.3 Recent Research

Schonlau et al. conducted their first experiment [14] based on UNIX commands that were keyed in by 70 users. Actually, a log of these commands was taken by using the

UNIX acct auditing mechanism. For every user a set of 15000 commands was recorded over a maximum period of one month. Out of these 70 users, 50 users were randomly selected and considered as legitimate users and the remaining 20 users were used as masqueraders. For the legitimate users, the first 5000 commands of each user was used as training data for that user and the remaining 10000 commands were used as testing data. These 10000 commands were interspersed with a masquerading user's command sessions. A session is actually a set of 100 commands. Hence with 10000 testing commands we now have 100 sessions to test. An example to illustrate the crust of Scholau technique is given below.

User1 has 15000 commands on the whole in his log file. So from a total of 150 sessions the first 50 sessions are used for training and the remaining 100 sessions are interspersed with sessions of say user2, user60 and user21. These sessions are now the masquerading sessions. The assumptions here are that, if we have a masquerading session then the probability of the next session being followed by the masquerading session from the same masquerader is 0.8 and the probability that another masquerader's session will immediately follow one masquerading session is 0.1. This means that if one masquerading session is to follow another masquerading session then the probability that the session comes from the same masquerader is very high. The chances of one masquerader to be followed by another masquerader will be such that there are at least two proper sessions in between the sessions of the two different masqueraders.

The review paper by Schonlau et al. describes an experiment in which six masquerade detection schemes are compared on the same user data spliced with masqueraders. The actual target that was achieved was from the range of 39.4% to 69.3%

for hit rates and 1.4% to 6.7% for false-alarm rate. The results achieved may seem poor but considering the difficulty of the problem the results are good. The problem to be addressed here is the false alarm rate. This rate of even 1% in a real world scenario can be very costly and might lead to a lot of system resource and time wastage. Figure 1 shows the results of Schonlau et al. graphically. The six schemes used by them are explained elaborately in the following paragraphs.

Bayes 1-Step Markov. This detector is based on single-step transitions from one command to the next and is due to DuMouchel [14]. The detector actually looks for transition of commands and it compares the trained user histories for such transitions. The comparison used here is actually a probabilistic comparison. It was a best performer for correct detections, but failed to accomplish the desired false alarm rates.

Hybrid Multi-Step Markov. This method is based on Markov chains, and is due to Ju and Vardi [14] [7]. Based on the proportion of the testing data the model toggles itself from a Markov model to a simple independence model, depending on the proportion of commands in the test data that were not observed in the training data. It had the highest performance rate when compared to other techniques.

IPAM. This detector is based on single-step command transition probabilities, estimated from the training data. IPAM stands for Incremental Probabilistic Action Modeling and was developed by Davison and Hirsh [14]. They actually developed this for predicting sequences of user actions. The results of this method ranks among the lowest-performing groups.

Uniqueness. This method was due to Schonlau and Theus [14] and used the ideas of command frequency detection. If a command frequently occurs for most of the

sessions of the user then the next session that has the same frequency of commands will be classified as a proper session. If an unusual command is seen in a session then the session will be classified as a masquerading session. Uniqueness was a poor performer to detect proper session but performed better to reduce false sessions. The false alarm rate attained by this scheme was close to 1%.

Compression. This method uses the logic that the new data from a given user compresses at about the same ratio as the old data from that same user. If this is the case then the masquerading user's data compression ratio should certainly vary significantly. Thus one can distinguish between a masquerader and a legitimate user. This idea belongs to Karr and Schonlau [15] and the results of this work are shown in Figure 1. Compression showed the worst results of all the schemes used for testing.

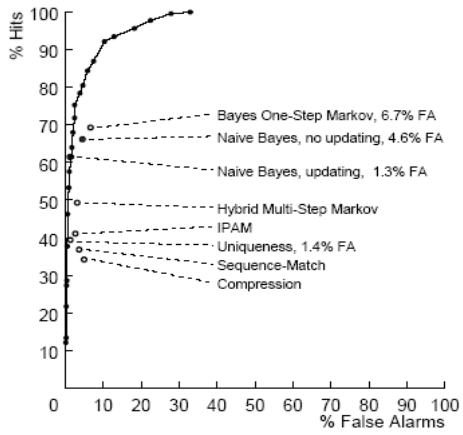

Figure 1: Relative operating characteristic (ROC) curve for the Naïve Bayes Classifier (with updating) on SEA data.

The Schonlau et al technique is called the SEA configuration or the SEA technique. The graph in Figure 1 is the ROC curve for the Naïve Bayes Classifer and it also includes the best-outcome results achieved by SEA techniques. In other words the

graph shows the relative best outcomes of different schemes proposed by Schonlau et al with the Naïve Bayes scheme. The Y-axis of the graph measures the % hit rate and the X-axis measures the % False Alarms. From the graph it is evident that the Bayes-One Step Markov had better hit rate to false alarm rate ratio. When one talks about masquerade detection the false alarm rates should also be considered. In fact false alarm rates can cause 6 times more loss than the hit rate measure. Considering these the Hybrid Multi-Step Markov scheme is the best performer amongst the Schonlau et al schemes. It has a hit rate close to 50% with a false alarm rate of 2%.

2.4 The Naïve Bayes Approach for Classification

The naïve Bayes approach for anomaly detection considers the masquerade detection problem as a text classification problem. This classification algorithm was initially used to classify text documents based on specific word counts as either sports, theatrical, health or politics article. It was typically called “bag of words” approach as it profiles documents simply based on word frequencies.

Maxion and Townsend [12] were the first to formulate the masquerade detection problem as a classification problem. The classification paradigm is shown in Figure 2.

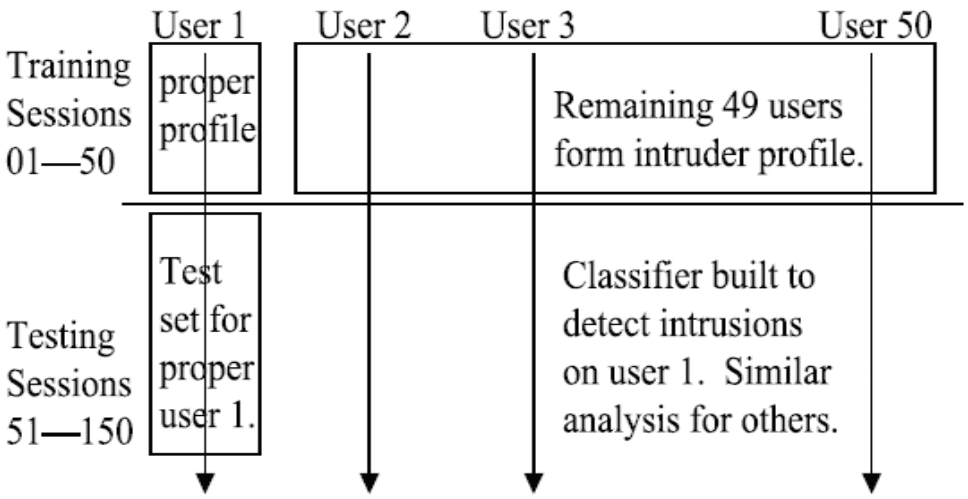

Figure 2: Classification Paradigm.

Figure 2 shows that Maxion and Townsend further probed into the data prepared by Schonlau et al. to consider the possibility of masquerading effect of the remaining 49 users to the user whose profile is being monitored. The classification paradigm works appropriately for the dataset prepared by Schonlau et al. It is more likely to work now since 2450 artificial masquerading sessions are at hand to be studied during the training phase itself. These masquerading sessions give a better knowledge of how one user will be victimized by the remaining 49 users who are insiders. The classification paradigm thus explored more possibilities in the SEA configured data. In a more general setting the dataset of a malicious user may not be like the typically well behaved user. Here an improvisation of the use of dataset was tried out to achieve better results. Apparently, pooling the 49 other users into one single intruder class provided a more stable classification problem.

2.4.1 Bag-of-Words Model

A text document can be easily classified based on the frequency of some words used in the document. This simple model was proven to be more successful than complicated models in the classification of text. Some complex models involved sequence information for text-classification but the bag-of-words model has proven to be more efficient and simpler than most of the algorithms.

After looking into the success of this model as a text classifier, Maxion and Townsend realized that this model could be used to categorize a set of commands as being used by a particular user. This classification comes after training the model to first identify the probability of use of each command by a user and then using this as classifying information to classify the future sessions as either proper or masquerading session [12]. A mathematical explanation of this model is given in the following paragraphs.

The main assumption of this model was that the user generates a sequence of commands, one at a time, with a fixed probability that is independent of the commands preceding it. The probability for each command *c* for a given user *u* is based on the frequency with which the command was seen in the training data, and is given by:

$$P_{c,u} = \frac{\text{Training Count}_{c,u} + \alpha}{\text{Training Data Length} + (\alpha \times A)}$$

Where $\text{Training Count}_{c,u}$ is the count of command c from all the sessions in the training phase produced by user u, Training Data Length = 5000, which is the count of all

the commands seen in the training phase, α is the pseudocount and A is the number of distinct commands (i.e. the alphabet) in the data. The pseudocount can be any real number, and was set to 0.01 in the study. The pseudocount's effect is to make sure that there aren't any commands with a zero count. The 'α' in the denominator balances the effect of α in the numerator. The probability that a test sequence of commands "x x y y x" was generated by user1, *u*1, is:

$$P_{u1,x} * P_{u1,x} * P_{u1,y} * P_{u1,y} * P_{u1,x}$$

Or $(P_{u1,x})^3 * (P_{u1,y})^2$ where Pu1, x is the probability that user1 typed the command x. Probability of the command x being not typed by user1 can also be computed and if the ratio of the probability of command being typed by user1 is greater than the probability of not being typed by user1, then command x can be associated to be typed by user1.

The data used in this work is the same used by Schonlau et al. [13]. As mentioned before these commands were captured from UNIX acct auditing mechanism. Here only two fields were captured i.e. the command name and the user. This limitation was imposed for privacy reasons. Some commands that were captured are: sed, eqn, troff, dpost, echo, sh, cat, netstat, tbl, sed, eqn, sh and so on.

2.4.2 1v49 Experimental setup

Maxion and Townsend's experimental setup was called the 1v49 experiment [12]. A major drawback of Schonlau et al. experiment was that not all users were tested as masqueraders; rather a different set of 20 users were considered as masqueraders. This kind of analysis doesn't use the dataset available to its maximum. When an algorithm

fails to identify a masquerader block, when checked for only one masquerader, we cannot really make out whether the problem really is with the algorithm or with the inappropriate use of the available data set. To address these shortcomings the experiment conducted by Maxion and Townsend considered the rest 49 users log files as masquerade data. This configuration was consistent in the number and origin of the masquerade events encountered by each detector. The result of such use of the dataset considered 2450 sessions of masquerade attacks for every user rather than 0-24 sessions of attacks considered by Schonlau et al. Such effective use of the available dataset brought better results than Schonlau's techniques and this is clearly shown in Figure 1.

2.4.3 Update problems in naïve Bayes Classification Method

In naïve Bayes classification with update model as proposed by Maxion and Townsend [12] a Single-Step update was made i.e. whenever an update of the user profile is made the decision that the session is either a proper session or a masquerading session was made in just one step. In other words a binary decision was made at each step and this may sometimes cause a complete collapse of the entire classification.

Two scenarios that might cause this mishap are, firstly when a masquerade session is classified as a true session without proper evidence. This might cause the future user sessions to be wrongly classified and there are chances for getting a lot of false-negatives. A false-negative session is one in which a masquerade session is falsely classified as a true session. This means that an intruder takes over the system. Secondly, an intruder session might look like a true session and it needs further profile updates to

classify it. Under these circumstances if the classifier puts this session into a proper session and after a few more classifications, if this session is not still seen, then we start doubting the correctness of this session classification.

All these problems can only be solved if the decision taken to classify the sessions is made backtrackable. If we had a mechanism that would classify a session later it would become easier for us to classify the sessions that we doubted, later, based on the new updated profile which now contains more stronger information for classification. This can be called a more "Consistent-Update" mechanism. If the algorithm decides when to classify these sessions then this mechanism now becomes a more "Self-Consistent Update" mechanism.

Figure 3 below shows the back tracking problems involved in Maxion and Townsend mechanism and also explains a self-consistent update mechanism as a solution for backtracking.

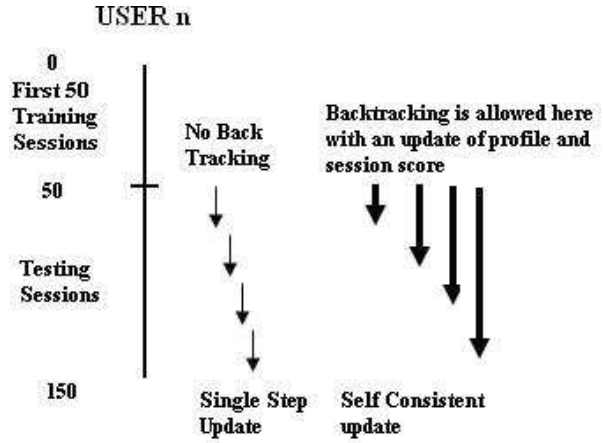

Figure 3: Single-step and self-consistent update mechanisms.

CHAPTER III

BAYESIAN NETWORKS APPROACH

3.1 Bayesian Networks

A Bayesian Network is a "Probabilistic Belief Network" [17] represented in the form of a DAG (Direct Acyclic Graph) [6]. The properties of the Bayesian Network are summarized below.

- Each node in the graph represents a variable or propositions.
- The relationship between the variables is represented by the links between the nodes. For Example: If a node "A" is connected to a node "B" with a downward arrow, then node "A" is the parent of node "B" and it directly influences the node "B".
- Every node in the network has a conditional probability table to keep the specific influence ratio that all parents pass to this node.
- There are no cycles in the graph.

A simple diagram of a Bayesian network is shown in Figure 4.

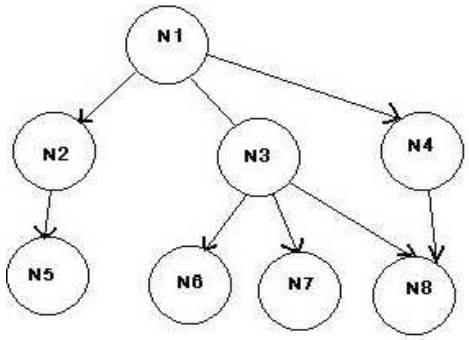

Figure 4: A Simple Bayesian Network.

We can compute the joint probability of all the nodes in the above network as given below.

P (N1, N2, N3, N4, N5, N6, N7, N8) =

P(N5|N2) * P(N6|N3) * P(N7|N3) * P(N8|N3, N4) * P(N2|N1) * P(N3|N1) * P(N4|N1).

For each node in the network we can specify a conditional probability using a table of its ancestors and its possible value. This is called the Conditional Probability Table (CPT). Each row in the CPT represents a possible case for the nodes considered and the corresponding value involved. An example of CPT is shown in Table 1. If both the parent nodes N3 and N4 are true then the probability of N8 to be true is 0.8. Similarly the remaining three conditions of the nodes N3 and N4 result in the probabilities of 0.7, 0.35 and 0.1 for N8 to be true.

N3	**N4**	**P(N8=T)**
T	**T**	0.90
T	**F**	0.70
F	**T**	0.35
F	**F**	0.1

Table 1: Conditional Probability for N8

3.2 Classification of the user sessions using trained Naïve Bayesian Networks of commands

We used the user's truncated UNIX commands log files created by Schonlau et al [13] for training and testing. This dataset contains 15000 commands for 50 users. The first 5000 commands of each user are true commands, in other words these were 100% pure commands that were typed by the genuine user. The remaining 10000 commands contain interspersed masquerade sessions in them. A session consists of 100 commands. These commands for each user were built over a period of time using the UNIX accounting command that takes a log of user typed commands namely "acct". After building 15000 commands for each of the 50 users Schonlau et al. carefully injected command sessions of another 20 users after 5000 commands of each user. So, starting from command number 5001 we have masquerade commands interspersed with genuine user's commands.

Schonlau et al. used the first 5000 commands to train their models and then using these trained models the sessions after 5000 commands were tested for masquerade sessions without any idea of what a masquerade session would look like. Maxion and Townsend [12] went a step forward to create masquerade sessions using the 5000 commands of the other users during the training session itself. This gave an idea of what a masquerade session would look like before the testing phase. Kwong H. Yung [18] then changed the method of updating the user sessions by using a self consistent update mechanism rather than single-step update mechanism. He also proposed a feedback mechanism to update sessions correctly.

3.2.1 Naïve Bayesian Network

The analysis that we will perform will consists of two phases

- Training Phase
- Testing Phase.

In the training phase a user log file is chosen. The first 5000 commands of this log file will be scanned by our classifier and forms a model for the genuine user. We also create an artificial masquerader model for the user under investigation from another log file of 5000 commands that was formed by picking up sessions of 100 commands from the remaining user log files. This artificial masquerade file makes the remaining users masqueraders and tests the sessions of the investigating user with a prior knowledge of intrusions based on the formed masquerade model. After the training phase is complete we have a two-layered Naïve Bayesian network of commands and models. Such a network is shown in Figure 5.

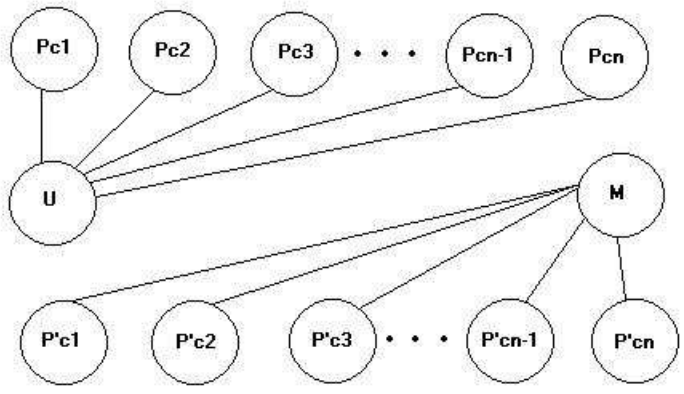

Figure 5: A Naïve Bayesian Networks of command proportions and Classes formed after the training phase for a user

In figure 5, U= Proper User Class that consists of probabilities of commands from the first 5000 commands in the user log file and M = Masquerader Class that consists of probabilities of commands from 5000 commands in the artificially created masquerader file.

The first layer of Bayesian network consists of probabilities of all possible commands seen in training and the second layer represents the class node that is derived from the command probability layer. For each user such Bayesian networks of commands and classes can be imagined. Since this is a simple classification problem based on the command count the above Bayesian networks setup forms our Naïve Bayes classifier. So the figure above is actually a naïve Bayesian network of possible commands proportions in one layer and the class to which such proportions of commands is assigned in the other layer. The command probability layer forms our model parameters and contributes to the mixture proportion of the class in the next layer. After the training phase the Bayesian network formed represents an initial configuration of user and artificial masquerade classes.

3.2.2 Deferral mode for classifying test sessions

Basically there are two types of classification in our test sessions.

- Offline Classification
- Online Classification

In Offline classification all test sessions can be classified with some lag. In other words we wait for the arrival of say “t” test sessions and only after the arrival of

the t^{th} test session we would classify all 1 to t sessions. Offline classification allows backtrack updating of test sessions. This backtracking mechanism allows for a probabilistic classification of each test sessions rather than binary 0 or 1 classification provided by the Naïve Bayes classifier. Each test session is assigned a probabilistic score which is recomputed a number of times as and when new test sessions arrive. Since each test session is scored in a sequential manner until t test sessions arrive, offline classification produces better results. The fact that the Schonlau data is contaminated with successive masquerade sessions in 80% of the cases and that each test session is scored in an overlapping manner due to the wait on t^{th} test session, the offline mechanism produces lower false and missing alarms.

Online classification makes a binary decision on each test session in a greedy fashion. In other words each session is classified at once as either proper or masquerade sessions. This could be harmful in cases where we have a false alarm and an update was not made, as this could cause more false alarms. The test session commands are added to the initial command proportions computed during the training phase as shown in figure 5. Hence, new probabilities are computed for each command during the testing phase. The larger the ratio of the probability of commands of User class to that of Masquerader class the greater the evidence of assigning the test sequence to the user.

Our Deferred Naïve Bayes Classifier is an Offline classifier. As discussed in the previous paragraphs we assign a probabilistic score to each of the test sessions and keep computing the multiple scores for each of the previous test sessions until some point where we need to classify all our test sessions. This means that we defer our decision to classify our test sessions until the arrival of say “t” test sessions. Once “t”

sessions arrive we make our classification of the waiting sessions that were “t” sessions behind. Such a deferral mode will surely produce optimal results than the simple Naïve Bayes classifier of Maxion and Townsend [12], since it also uses the information of “t” test sessions that follow to classify the waiting session.

3.2.3 Bayesian Network Detector for masquerade detection

In our analysis we propose to test for a wait on “t” sessions to classify test sessions. The trust node in our Bayesian Networks decides “t”. The Bayesian Networks detector that we propose makes use of the advantages of both online and offline techniques. In other words the Bayesian Network toggles between a Naïve Bayes Classifier and a Deferred Naïve Bayes Classifier based on a trust criterion. Such a Bayesian Network is shown in Figure 6. The figure shows two types of detectors working in parallel on the test sessions. Initially a trust level of 1 is set on the Naïve Bayes Classifier. As new test sessions arrive the trust level is computed based on the ratio of correct and incorrect classifications. An incorrect classification on complete real-time IDS would consist of only false alarms. In other words when there is a false alarm the genuine user reports it as a bug to a bug database. This information can be used to compute the new trust of our trust node. In our analysis we already know the classifications of all the test sessions. This is because we interspersed some masquerade test sessions into the genuine user sessions. So we computed incorrect sessions based on this test session classification table. In other words our incorrect classifications accounted for both false and missing alarms.

A threshold of 0.7 was set on the trust node. This threshold limit is adjustable and it represents the maximum erroneous classifications that our detector would accept. Once the trust on the Naïve Bayes classifier is reduced to 0.7, our Bayesian Networks Detector fires the Deferred Naïve Bayes Detector to classify the test sessions it has seen so far. This includes the new test session, which was not classified by Naïve Bayes Classifier. After this classification the trust node is reset to its initial setting and the Naïve Bayes Classifier model is updated with the Deferred Naïve Bayes Model. This tunes the Naïve Bayes Classifier and we could expect better results than before since the substituted model is from an offline classifier.

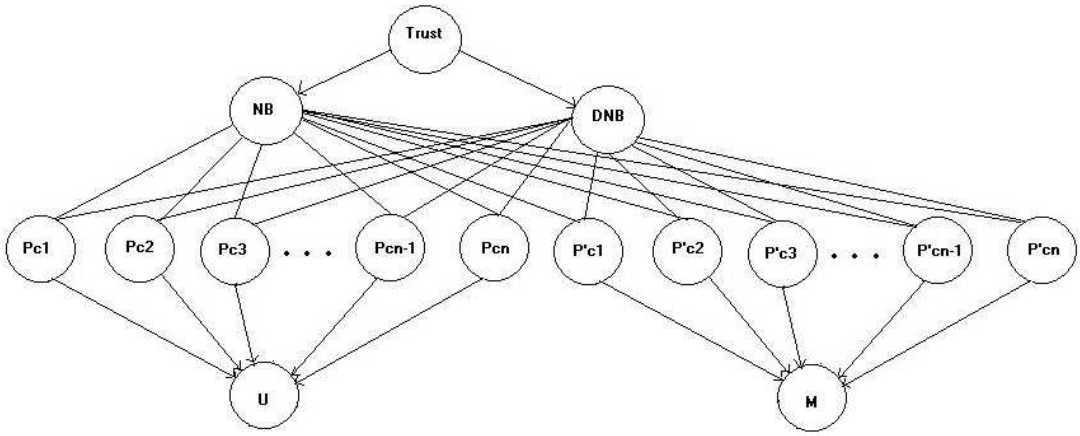

Figure 6: Bayesian Networks Detector with Naïve Bayes and Deferred Naïve Bayes Classifiers.

In Figure 6, NB = Naïve Bayes Classifier and DNB = Deferred Naïve Bayes Classifer. The trust node (Trust) toggles between the Naïve Bayes and Deferred Naïve Bayes based on the threshold set. When the Deferred Naïve Bayes Classifier announces its results, we could even crosscheck if some of our previous sessions were classified incorrectly by the Naïve Bayes Classifier.

3.2.4 Offline classification of Deferred Naïve Bayes Classifier

A Deferred Naïve Bayes Classifier performs an offline classification of a session. Whenever a new test session arrives, we compute the model parameters pc and p'c for each command by including the test session commands with the already existing model parameters. A detailed explanation of these parameters is presented in section 3.2.5. When computing the model parameters we use the expectation formula to compute the probability that the session is a masquerade or proper session. This is given as a score for the current session. In the Naïve Bayes classifier we make an instant binary classification decision.

Since we provide a probabilistic score we can keep scoring the current test session until a point where we could say that the session has a good evidence to be classified as a proper or masquerade session. The fact that Schonlau's data has continuous proper and masquerade sessions, help in this offline classification of test sessions. Thus a test session's decision can be deferred until we encounter few more test sessions so that the probabilistic score that is assigned to the test sessions becomes more trustable so that we could now classify the session without much doubt. A mathematical explanation of this discussion is provided in session 3.2.5.

3.2.5 Explanation of Naive Bayes and Deferred Naïve Bayes Classifiers

The Naïve Bayes classifier was explained in section 2.4.1 where we provided a brief introduction to the Bag of words model. For each session a certain probability of

every command occurs based on its count in that session. This probability of a command now becomes our point of focus. To be precise, the probability that a test sequence of commands "x x y y x" was generated by user1, *u*1, is:

$$P_{u1,x} * P_{u1,x} * P_{u1,y} * P_{u1,y} * P_{u1,x}$$

Or $(P_{u1,x})^3 * (P_{u1,y})^2$, where Pu1, x is the probability that user1 typed the command x. Based on the command probabilities we now build a session node that has a conditional probability of all commands that form its parent node.

Similar probability estimation will be made for session2 also and in this manner all the sessions in training will have their respective parent command nodes contributing their probability of occurrence for each of the session nodes. This is clear from Figure 7. The first layer shown in figure 7 is a layer of command probabilities and the second layer has two different nodes. The S node represents the genuine user sessions and the M node represents the artificial masquerade user sessions during the training phase. The arrows from layer 1 to layer 2 indicate that each session receives some contribution of command probability from layer one.

One important thing to be noted here is if there is no contribution by a command in a session then its probability contribution is assigned a value "α" equal to 0.01 (see section 2.4.1). "α" is called the psuedocount and is added to both the numerator and denominator of the Bayes formula shown in section 2.4.1.In this way we form a complete set of training session nodes for both the proper sessions and the artificial masquerade sessions.

Figure 7 represents the Bayesian network consisting of 50 genuine sessions of a user and 50 artificial masquerade sessions picked from the remaining 49 users. That is if

we are monitoring user1's commands then the remaining 49 user's (user2 to user50) sessions will form artificial masquerade sessions for user1. In this way we can build our Bayesian network for each of the users under surveillance. For each of the session nodes the respective conditional probability is calculated based on the commands in that session. These conditional probabilities can be later used for classifying future test sessions and updating them into the network. An example CPT is shown in table 2.

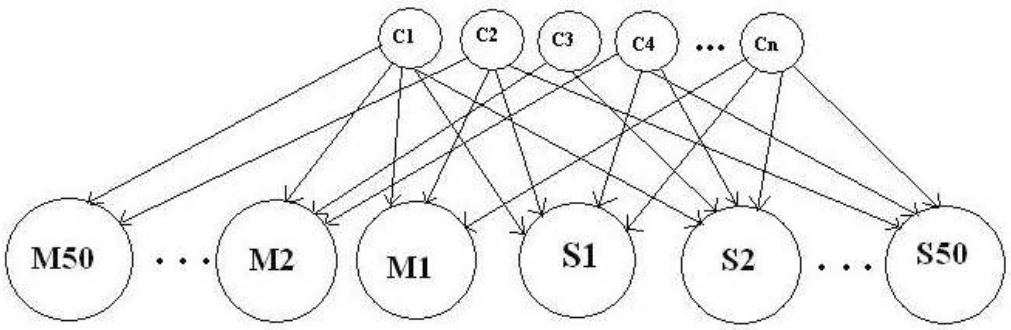

Figure 7: Bayesian Networks for the set of artificial masquerade and proper sessions after training phase.

$P(C_1)$	$P(C_2)$	…	$P(C_n)$	$P(S_i)$
0.0002	0.0003	…	0.005	0.000034
0.009	0.0008	…	0.0007	0.000056
. . .	. . .	. . .	. . .	. . .
0.0023	0.0000034	…	0.000789	0.0000089

Table 2: CPT for various sessions based on the command count Probabilities.

A Similar table can be constructed for masquerade sessions also. Since this is a command counting problem considering only session information in the training phase to classify sessions will not be of great help. Instead of computing the conditional probabilities of each session we could compute the conditional probability of all commands in the training phase and form masquerade and proper classes. The stronger

the evidence, the chances of a correct classification increase. Moreover, it is very difficult in reality to expect accurate classifications using a session's conditional probability. Thus in practice we don't compute the conditional probability values as explained above in Table 2. Since we know the identity of all the training sessions we consider the joint conditional probability of all the training sessions and we compute two classes as shown in Figure 5.

As explained before we classify our new test sessions using a Bayesian trust network that toggles between a Naïve Bayes and Deferred Naïve Bayes Classifiers. So, before testing begins the classes that were formed in training are used as initial classifiers. We update this initial classifier during our testing phase and use this updated classifier at each step to classify future sessions. Actually we build two different classifiers for our Bayesian Networks approach, one for Naïve Bayes and another for Deferred Naïve Bayes. Once the trust node toggles to Deferred Naïve Bayes we first classify the new test session with the Deferred Naïve Bayes model, replace our Naïve Bayes classifier model with the Deferred Naïve Bayes classifier model and then proceed with this model on the Naïve Bayes classifier. Once again our Deferred Naïve Bayes model starts its offline classification all over again and waits for its chance to classify the test sessions.

3.3 A Mathematical analysis of Naïve Bayes and Deferred Naïve Bayes Classifiers

The following discussion illustrates the mathematical model of the classifiers used in the masquerade detection problem.

We first explain how we compute the model parameters for naïve Bayes classifier and then present how expectation scores were computed for Deferred Naïve Bayes classifier. This is followed by a brief explanation of the Bayesian Network toggling approach.

1. Conditional Probability of a session of U_k (User k) based on the probabilities of command frequencies is,

   $P(Sk) = \prod_{i=1}^{n} P(Ci)$, Where P (C_i) is the probability of occurrence of command 'i' in the session k of 100 commands and P (S_k) is the conditional probability of the session k based on the command frequencies.

   i.e. P (S_k) = $P(C_1)^{Nuk1} P(C_2)^{Nuk2} \ldots P(C_j)^{Nukj}$, for all C_i with N_{Uki} (Number of times user k used command j) > 0 i.e. if there is no contribution of a command in a session then that command is assigned probabilities based on our adjusting parameter α as explained in section 2.4.1.

2. Let Cs denote the sequence of commands in session number "s". By Bayes inversion formula the posterior probability P (U|Cs) of user U given the sequence s is,

   P (U|Cs) = P (Cs|U) P (U)/P (Cs) i.e. α P(Cs|U) P (U).

   Where, P (u) is the prior probability for user u, and P (C_s|U) is the probability that the sequence C_s was generated by user u. In practice, C_s is assigned to the user u_0 = argmax{P (C_s|u) P (u) |u = 1, 2…, U}, among U different users. In other words, the session Cs is assigned to the user u_0 who most likely generated that

session. This rule is the optimal Bayes rule under the usual uniform loss function for misclassification. In the naive-Bayes model, each command is assumed to be chosen independently of the other commands. User u has probability P_{UC} of choosing command C. Because each command is chosen independently, the probability, P (Cs|U), that user u produced the sequence $C_s = (c_{s1}, c_{s2} \ldots, c_{sk} \ldots, c_{szs})$ of commands in session number s, is simply

$$P(c_s|u) = \prod_{k=1}^{z_s} P(c_{sk}|u) = \prod_{k=1}^{z_s} p_{uc_{sk}} = \prod_{c=1}^{C} p_{uc}^{n_{sc}}$$

, Where

$n_{sc} = \sum_{k=1}^{z_s} 1_{\{c_{sk}=c\}}$, is the total count of command c in session "s" [4].

3. From now on let pc and p'c be the probability of command c in a proper and masquerading session respectively. We compute the log likelihood L_s of a test session s up to an additive constant as shown in equation 1

$$L_s = (1 - l_s)\,(\log\,(1 - \varepsilon\,) + \sum_{c=1}^{C} nsc \log pc\,) + l_s\,(\log\,\varepsilon\, + \sum_{c=1}^{C} nsc \log p'c\,) \qquad (1)$$

ε and 1-ε are the prior probabilities that a session is proper and masquerading. In our entire analysis these values where fixed to 0.5.

Assuming that all the test sessions are generated independently of each other, the cumulative log-likelihood L^t after t test sessions is up to an additive constant

$$L^t_{+} = \sum_{s=1}^{t} Ls \quad = \omega^t_{+} \log\,(1 - \varepsilon\,) + \sum_{c=1}^{C} \mathrm{nt} + c \log \mathrm{pc} \; + \; \omega'^t_{+} \log\,\varepsilon\, +$$

$$\sum_{c=1}^{C} n't + c \log p'c \qquad (2)$$

where,

$$\omega^{t}_{+} = \sum_{s=1}^{t} (1 - 1s) , \qquad (3)$$

$$\omega'^{t}_{+} = \sum_{s=1}^{t} ls \ , \qquad (4)$$

$$n^{t}_{+c} = \sum_{s=1}^{t} (1 - 1s) nsc , \qquad (5)$$

$$n'^{t}_{+c} = \sum_{s=1}^{t} (ls)(nsc) \qquad (6)$$

Here ω^{t}_{+} and ω'^{t}_{+} are the cumulative numbers of proper and masquerading sessions respectively; n^{t}_{+c} and n'^{t}_{+c} are the cumulative counts of command c amongst proper and masquerading sessions, respectively, in the t total observed test sessions.

4. Rare classes and rare commands may not be properly reflected in the training set. Therefore to avoid zero estimates, smoothing is applied to the maximum-likelihood estimators. This smoothing can also be motivated by shrinkage estimation under Dirichlet priors on the parameters ε, p_c and p'_c

   Here the parameters ε, p_c and p'_c are drawn from known prior distributions with the fixed parameters, and simple standard Bayesian analysis is applied. Based on this discussion we compute the new cumulative posterior likelihood $L^{\sim t}$ as in equation 7.

$$L^{\sim t} = (\beta\ -1+\omega^{t}_{+})\log(1-\varepsilon) + \sum_{c=1}^{C}(\alpha c -1+(n0+c)+(nt+c))\log pc$$

$$+(\beta'\ 1+\omega^{'t}_{+})\log\ \varepsilon\ +\ \sum_{c=1}^{C}(\alpha' c -1+(n'0+c)+(n't+c))\log p'c \qquad (7)$$

Here, n^{0}_{+c} and $n^{'0}_{+c}$ are the total counts of command c in the proper and masquerading sessions of the training sessions.

5. Using equation 7 we compute our model parameters ε, p_c and p'_c as shown in the equations below,

$$\hat{\epsilon}^{t} = \frac{\beta' - 1 + w'^{t}_{+}}{\beta - 1 + \beta' - 1 + t}, \qquad (8)$$

$$\hat{p}^{t}_{c} = \frac{\alpha_c - 1 + n^{0}_{+c} + n^{t}_{+c}}{\sum_{v=1}^{C}(\alpha_v - 1 + n^{0}_{+v} + n^{t}_{+v})}, \qquad (9)$$

$$\hat{p}'^{t}_{c} = \frac{\alpha'_c - 1 + n'^{0}_{+c} + n'^{t}_{+c}}{\sum_{v=1}^{C}(\alpha'_v - 1 + n'^{0}_{+v} + n'^{t}_{+v})}. \qquad (10)$$

After the training phase we compute initial classifier model θ^{0} by neglecting the test session command counts in the equations 8, 9 and 10. As the test sessions arrive we include the test session command counts and compute the new model parameters, which reflect the influence of the test sessions.

6. If we use Naïve Bayes classifier then we need to immediately classify the newly arrived test session. This is done by finding out the ratio of the newly obtained model with the initial model. If the ratio of p_c to pc0 is greater than p'c to p'c0 then we classify the test session as proper session, else as masquerade session.

7. In the case of Deferred Naïve Bayes classifier we use probability values to estimate the test sessions proximity of being a masquerade or proper session. Equation 11 below shows how a session can be assigned a probability value. Let us call this a score. Hence our test sessions are scored as and when they arrive. Since we use a backtracking method to score our test sessions we keep updating the scores as new test sessions arrive. In other words multiple scores are provided for each of our already seen test sessions before we see the last or the t^{th} test session. At this stage when the t^{th} test session arrives our Deferred Naïve Bayes classifier classifies the test sessions as proper or masquerade if the probability scores are below and above 0.5 respectively.

   The equation used to score each of our test session is as below,

   $$P(ls=1|Cs;\theta^t) = (P(ls=1;\theta^t)\ P(Cs|ls=1;\theta^t))/P(Cs;\theta^t) \quad (11)$$

   where,

   $$P(Cs;\theta^t) = P(ls=0;\theta^t)\ P(Cs|ls=0;\theta^t) + P(ls=1;\theta^t)\ P(Cs|ls=1;\theta^t).$$

8. Our Bayesian network first places a trust of 100% on the Naïve Bayes Classifier.

So we consider the model parameters computed by this classifier initially to classify our test sessions online. Each time when we classify a session incorrectly the trust value on the node is adjusted based on the ratio of correct to incorrect classifications. Once the trust level reaches 70% we toggle to our Deferred Naïve Bayes Detector, which was running in parallel, and classify all the sessions it has seen at this point. Thus the Deferred Naïve Bayes Classifier also classifies our new test session. At this point we update our initial Naïve Bayes model with the Deferred Naïve Bayes model and reset our trust node to 100% on the Naïve Bayes Classifier. Thus a more reliable model is now used to classify our future sessions using Naïve Bayes Classifier.

9. During our classification of test sessions we also build a table of hits, false alarms, and missing alarms. This table was used to calculate the trust on the trust node and was also helpful to draw ROC curves during our final analysis.

CHAPTER IV

CLUSTERING OF COMMAND SESSIONS

4.1 Clustering Technique

Intrusion detectors that use audit data such as command sequences handle voluminous information in order to classify test sessions. More data as backend implies more storage area and hence more information to peruse before making a decision. Clustering of command sequences also has the same problem of dealing with huge audit data during the training phase but once dense clusters are formed the centroids of clusters become our point of focus. Thus there is a possible data reduction during the testing phase as only cluster centroids can be used while matching a test session.

To form clusters of command sequences we first divide a command session into a number of overlapping sequences. Once the sequences are formed they are clustered based on a sequence match criteria. Finally we have a centriod that represents the cluster characteristic. The amount of data to be handled while forming a cluster of similar records with a *centroid* that holds the mean characteristic of the cluster is huge [8]. A centroid of a cluster is like a black hole that attracts the members of the cluster toward it. Mathematically, a centroid is the mean value of a set of values that are closely related. This close relation of the members of the cluster is the most important factor of all clustering algorithms. After clustering, the centroids of the clusters become the point of interest and thus the rest of the information is deemed outdated. So while testing we can compare our test sequences to the cluster centroids. This might be considered as an

advantage of clustering technique but this also depends on the density of our clusters. The denser clusters will have less number of centroids to compare with our test sessions.

Clustering approach was initially used in data mining field to cluster real time data and compute their likelihood patterns based on the formed clusters [5]. We propose an online clustering approach, which is a variant of the data mining approach presented in [5]. Our Clustering approach is further classified into three techniques based on the sequence rating mechanism in the testing phase. We propose this clustering analysis to study its feasibility as an Intrusion Detection System in real time. Also we compare the results of clustering and Bayesian Networks analysis by plotting ROC curve in section 5.3.2.2.

With this general picture of clustering of command sequences we now explain the components of our clustering mechanism and discuss our clustering algorithm in the following sections.

4.2 Clustering scenario of Masquerade detection using command lines.

Any solution proposed to a problem views the problem in its context. A clustering solution to our problem of detecting masquerades can be viewed as consisting of the following components.

1. Audit Data Creator
2. User Profile Creator
    a. Sequence Grouper
    b. Sequence Analyzer

c. Frequency Counter

d. Cluster Tuner

3. Validity Checker

a. Sequence Scorer

b. Alarm Raiser

4. User Profile Updater

The above blocks work internally communicating with each other for two phases namely Training and Testing phase. A block diagram with the communications between various components is shown in Figure 8. A complete description of the components is also presented after the block diagram.

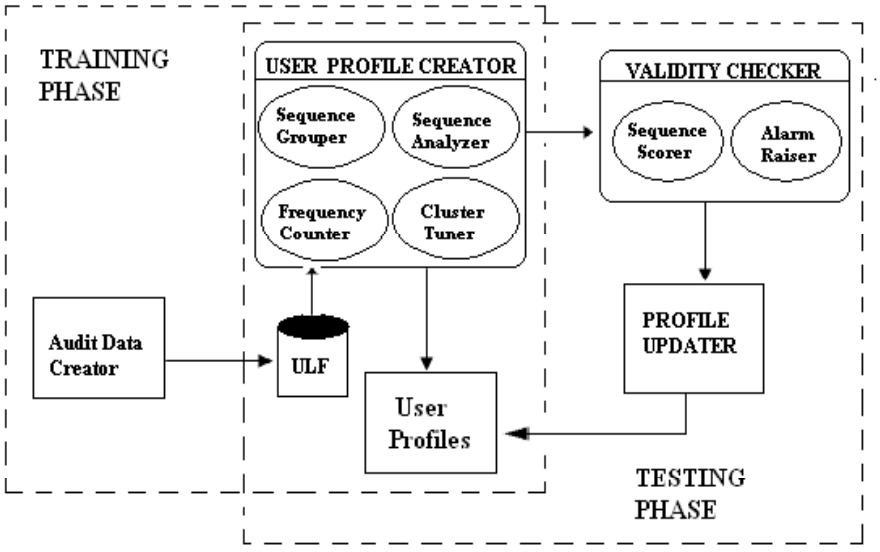

Figure 8: Masquerade Detector based on clustering of user command sequences.

Audit Data Creator – A Log File creator that stores the log of user commands during a login logout session to a database called User Log Files (ULF).

User Profile Creator – Based on the clustering scenario a user profile creator can be defined as user command sequences grouper. If we present sequences of commands to a UPC (User Profile Creator), the output will be clusters of command sequences. A profile of user "u" is denoted by "P_u" which represents the set of all clusters of user u's command sequences that were formed using the clustering algorithm i.e.

$P_u = \{C_1, C_2, C_3 \ldots C_n\}$, where $C_1 \ldots C_n$ are clusters of user u's command sequences.

Sequence Grouper – A sequence grouper is a component that clusters the command sequences presented to it. Generally it contains the algorithmic implementation of clustering.

Sequence Analyzer – This component examines the command sequences and finds the similarity between two sequences based on the underlying principle. Generally the LCS (Longest Common Subsequence) principle is used to match similarities between two sequences. Other techniques used for similarity matching are MCP (Match Count Polynomial Bound) that matches each slot of the command sequences considered for similarity checking, MCE (Match count Exponential Bound) that has an exponential influence for each slot that matched and MCAP (Match Count Adjacency Reward Polynomial Bound) that matches adjacent slots of command sequences.

Frequency Counter – This section counts the distinct commands occurring in each sequence. It basically finds out the frequency of occurrence of each command in a

sequence. In order to find the similarity between two command sequences s_1 and s_2 we compute both the LCS and the command counts, we then match the sequences based on both LCS and the command counts.

Cluster Tuner – The job of cluster tuner is to either split a cluster or join two clusters. This is explained in the algorithms given below.

Validity Checker – During the testing phase this component checks the validity of the command sequences that are to be clustered. Validity is computed based on scoring given to each new sequence.

Sequence Scorer – Each of the new sequence is scored based on one of the three techniques namely, Previous_n, Weighted and Decayed Weight techniques [10].

Alarm Raiser – This component raises an alarm if an anomalous sequence is found.

Profile Updater – If a normal command sequence is found then this sequence is updated to the user profile i.e. it is pushed to a particular cluster. The Profile updater's job is to carry out this updating of normal command sequences.

4.3 Clustering of command Sequences – Training phase

The job of the sequence grouper is,

- To cluster the command sequences based on the similarities.

- To compute the new centroids of the clusters.

A sequence "i" of length "j" denoted by S_{ij} is a set of j commands as shown in equation 1.

$$S_{ij} = \{C_1, C_2 \ldots , Cj\}, \text{ where } j = 10, 11 \ldots 100 \quad (12)$$

During the training phase a sequence is selected from the session log file and the value of j is set to integral values in the range [10,100]. In the testing phase the sequence grouper waits for online j commands and forms a sequence.

Two sessions S_1 and S_2 are considered to be similar if,

$$\text{Match } (S_1, S_2) \geq T_r, \text{ Where } T_r \text{ is the threshold set for similarity.} \quad (13)$$

A Match of two sessions S_i and S_k is defined by the following equation,

$$\text{Match } (S_i, S_k) = \text{LCS } (S_i, S_k) + \text{MCAP } (S_i , S_k) \quad (14)$$

In equation 14 LCS is the Longest Common Subsequence between the two sequences S_i, and S_k, and MCAP is the Match Count Adjacency reward Polynomial Bound.

The Sequence Grouper Algorithm for forming clusters is given below.

Sequence Grouper (T_r, S_u, p_u, S_u^c):

/* T_r is intra-cluster similarity threshold
S_u is a set of a user u's sequences to be clustered
p_u is the user u's profile, i.e., set of clusters
S_u^c is the set of user u's cluster centers
C_x is the cluster for center x*/

1. Initialize $p_u = \phi$, $S_u^c = \phi$, $C_x = 0$.
2. Randomly select a cluster center from S_u and update both S_u and S_u^c
   i.e. $S_u^c = \{S_l\}$ and $S_u = S_u - \{S_l\}$, where S_l = sequence l considered as centroid.

3. Initialize the Cluster set to ϕ i.e. $C_u = \phi$.
4. Include the center to the Cluster C_x i.e. $C_{x=}\{S_l\}$
4. While $S_u \neq \phi$ do
5. For j = 1 to sizeof(S_u)
   If (Match $((S_u^c)_{x,} S_{uj}) \geq T_r$ then (Where $(S_u^c)_x$ the clustercentroid that produces the best match)
   $C_x = C_x \ \mathbf{Y} \{S_{uj}\}$
   Else $S_u^c = S_u^c \ \mathbf{Y} \ \{S_{uj}\}$
6 $C_x = C_x \ \mathbf{Y} \ (S_u^c)_x$
7. $S_u = S_u - ((S_u \ \mathbf{I} \ C_x) \ \mathbf{Y} \ (S_u^c - (S_u^c)_x))$
8. Recalculate the cluster centre for the cluster C_x say $RCCC_x$
9. $(S_u^c)_{x=} RCCC_x$
10. $C_u = C_u \ \mathbf{Y} \ C_x$ (We assume that this is the latest updated C_x)
11. $p_u = p_u \ \mathbf{Y} \ C_u$
12. If a new sequence arrives $S_u = S_u \ \mathbf{Y} \ S_{New}$

4.3.1 Cluster Tuning

Another important component of the sequence grouper is the cluster tuner. The main job of the tuner is to join two similar clusters or split the discrepancies in cluster into two. This actually means that we are now fine-tuning our clusters so as to reduce the inherent noise in the cluster. The two algorithms are given below,

//T' be the inter cluster similarity threshold

Join (T', p_u, S_u^c)

1. For each pair of clusters c_i and c_j in the profile $p_{u,}$ $i \neq j$
2. If Match(c_i, c_j) >= T'
   $c_i = c_i \ \mathbf{Y} \ c_j$ // We can now unite the two clusters
3. Recalculate the centroid for $c_{i.}$
4. $p_u = p_u - c_j$
5. $S_u^c = S_u^c - S_{cj}$

Split (T, ST', p_u, S_u^c)

1. For each Cluster c_i in the profile p_u

2. If (T >= ST') SequenceGrouper(Tr+1, c_i , p_u, S_u^c)
3. $p_u = p_u - c_j$
4. $S_u^c = S_u^c - S_{cj}$

4.3.2 Recalculating Cluster Center

To recalculate cluster centers we use a lazy technique that computes cumulative scores considering each sequence in the cluster as the new cluster center. Finally the sequence with the maximum cumulative score is elected as the new centroid.

To match two sequences we again use the same matching criteria as shown in equation 14.

Thus the new cluster centre is based on the following scoring mechanism,

New Cluster Center = Max (Cumulative Scores of each sequence "i" in the cluster), where,

Cumulative Scores of each sequence "i" in the cluster = $\sum_{i=1}^{t} \sum_{j=1}^{t} Sim(Si, Sj)$, and i<>j.

4.4 Influence of Threshold value in cluster formation

Threshold values are the percentage of acceptance values that are set based on the problem situation and observer's approximation. Any approximation that has a threshold value toward its upper bound will tend to form stringent clusters but most of the situations don't want this rigid cluster formation, as the tendency to strike false alarms is higher. In other words strict threshold values reject similar values that were supposed to fall into a cluster if the rules weren't rigid. It is therefore advisable to iterate the

algorithm with a loosened threshold value until we get a close match for all the values to be clustered. Here again we assume that all the values that were intend to be pushed into a cluster are proper values. We now determine a mathematical explanation for threshold fixation.

Let T_r be the threshold value to be set for clustering our command sequences, ωi, be the density of the i^{th} cluster that was formed based on T_r. T_r must be chosen such that equation 6 is met,

$$(\sum_{i=1}^{n} \omega) \to \infty \tag{15}$$

i.e. our threshold value should be such that we form more dense clusters. To achieve such cluster formation is not easy. In most cases we try to approximate the threshold values after some iteration.

We now discuss the technique of choosing the threshold value for our situation. Firstly, a command sequence length "*l*" is decided. Thus each sequence of commands will be of length "*l*". This length can be chosen at random, let us say *l*=10 in our case. Now, threshold value is actually an integer value that approximates two sequences based on the matching function Match (S1, S2) (For Ex: sequences S1 and S2 are considered). If threshold T_r = 10, then we are looking for sequences such that their LCS + MCAP of their commands is >= 10, so as to cluster them. If we are not able to cluster some of the command sequences then we drop our threshold by a scale of 1. This is done until we reach a limit such that T_r is at least 6. That is we have an approximation of about 60%. After this approximation if we are still not able to cluster some of the commands we keep them as it is. All of the above discussion holds good for training phase.

Generally, setting threshold value to the upper bound has its own cost of raising false alarms. If we set the cost function to be as below,

$$\text{Cost} = \text{misses} + \text{false alarms} \tag{16}$$

We can actually have a threshold close to the upper bound. The reason is that we give same weight for both misses and false alarms. If the cost function was like in equation 8 then its better to choose a threshold value somewhere close to the lower bound.

$$\text{Cost} = \text{misses} + 6(\text{false alarms}) \tag{17}$$

In equation 17 it is very costly to get a false alarm so we need to balance this out with our reduced threshold value. Thus choosing threshold value is critical to clustering problem.

4.5 Testing Phase

The testing phase of the clustering scenario is an online testing scenario. Whenever sequences arrive online we rate the sequences based on a Scoring mechanism. After scoring a sequence, the score value is checked with an accept threshold value to either cluster this sequence or reject it. If the scoring value is significantly below the accept threshold then the sequence is considered anomalous and immediately an alarm will be triggered. If the scoring value is >= threshold value then we consider this sequence to be produced by a normal user and thus we update this sequence to the already existing user profile. The accept threshold is an empirical value that is computed based on observation of some test sequences.

The scoring mechanisms that will be used in this project are LAST n, WEIGHTED and DECAYED WEIGHTS. These are explained below.

LAST n – Here first n sequences in the testing phase are rated cumulatively and the later sequences are judged based on these n sequences. The rating mechanism can be mathematically represented as,

$$R_j = 1/(j+1) \sum_{t=0}^{t=j} Sim(s't, pu) \text{ , if } j<n$$

Or

$$R_j = 1/n \sum_{t=j-n+1}^{j} Sim(s't, pu) \text{ , if } j>=n \quad (18)$$

WEIGHTED – Here the weighted mean of the last sequence's rating is incorporated into the current sequences Similarity measure. The mathematical representation is given as,

$$SC_j = \alpha * \text{Match}(s_j, p_u) + (1- \alpha) * SC_{j-1} \quad (19)$$

Here we can fix α to be a finite nonzero quantity. Generally it is fixed to 0.33. The match function in equation 5 actually computes the match of the new sequence with the cluster center for each cluster. In the equation SC_{j-1} actually refers to the last accepted sequence before the sequence j. Thus we also weigh this new sequence j with respect to the last accepted sequence j-1. This kind of cumulative scoring helps maintain the characteristic of the new sequences with already matched sequences.

DECAYED WEIGHTS – This is a variation of Weighted means mechanism. Here α is varied based on the current sequence. Mathematically it can be represented as,

$$SC_{j} = \alpha_{j} * \text{Match}(s_j, p_u) + (1- \alpha_{j}) * SC_{j-1} \quad (20)$$

Where $\alpha_j = \alpha_{j-1} / \alpha_{j-1} + 1 - \log(z/(y+j))$, and $\alpha_0 = 1$. Thus α_j is a decaying weight as long as $1 - \log(z/(y+j)) > 0$, also y = 6750 and z = 7500 in our analysis.

4.5.1 Predicting acceptable and unacceptable Command sequences

In order to classify the sessions that occur online as either proper or masquerade sessions we use the above scoring mechanisms. We empirically choose an accept threshold score called Seq_Accept. If the score calculated for a test sequence is >= this Seq_Accept value then we consider the new session as a proper session. We now call the Sequence Grouper algorithm to cluster this new session with the already existing clusters.

Profile updater does this clustering of the test sequences. The profile scorer carries out calculating the scores. The alarm raiser's job is to raise an alarm if an anomalous session is found. Once the newly scored sessions are updated to the user profiles we now have a more reliable user profile database to refer to in order to classify future sessions.

4.6 Steps Involved In Training and Testing Phase

The definition of keywords to be used in both the phases is given below.

- A Command Sequence S_i is a finite set of 10 commands and is represented as

  $S_i = \{C_1, C_2, C_3, \ldots C_{10}\}$.

- A Command Session Se_i is a finite set of 10 command sequences i.e.,

  $Se_i = \{\{S_1\}, \{S_2\}, \{S_3\}, \ldots \{S_{10}\}\}$.

- A Cluster C_i of command sequences is defined on a Centroid i represented as S_c^i is a set of sequences that match closely to the centroid i and is represented as

  $C_i = \{S_{j,} S_a , S_{n...,} S_p \}$.

- A profile P_u is a set of clusters that are obtained during training session i.e.,

  $P_u = \{C_1, C_2, C_3 \ldots C_n\}$, where 1, 2...n represent the cluster centers.

4.6.1 Training Phase

1. Randomly select a cluster center and try to cluster other sequences in the session to this centroid using the match criteria Match $(S_{i,} S_j) >= T_r$.
2. All sequences that are not clustered are considered as a set of unclassified sequences and a new cluster center is considered randomly for this set. Step 1 is then followed for this set also, this will continue until there are no more command sequences to cluster.
3. In Step1 when a cluster is formed the profile of that user for whom training is done is updated with the cluster center. Therefore, finally when training is done the profile set will consist of all the cluster centroids.
4. Once we form clusters, our next step is to tune the clusters by either joining or splitting them. To join two clusters we match the two centroids of the clusters. If we find a match we then merge the two clusters and recalculate their centroids. To split a cluster we pick centroids in the profile and calculate the threshold value based on the matching function. If this value T is much greater than the assigned

sequence threshold value for the centroid, we split the cluster using the Split algorithm.

4.6.2 Testing Phase

1. In this phase first online command sequences are formed as the commands arrive.
2. Each sequence is scored based on the weighted scoring phenomenon.
    a. The first sequence is scored independently
    b. The sequences that follow are scored with some weight given to the previous accepted sequence.
3. After, a sequence is found to be acceptable; it is updated to the user profile by using the Sequence Grouper Algorithm.
4. If a session is found to be anomalous then an alarm is triggered.

CHAPTER V

BAYESIAN NETWORKS VS CLUSTERING ANALYSIS

5.1 A Comparative Study

Probabilistic and Clustering analysis simulators were created in C#. The User interfaces of these two simulators are shown in Figure 9 and Figure 10. The output Bayesian Networks simulator is a text file named UserX_Classifier.txt. This file contains the classification of test sessions into Proper and Masquerade sessions based on the probabilistic model $\theta = \{\varepsilon 0, pc, p'c\}$ for each command. Clustering analysis on the other hand produces five output text files for each analysis; these files are named as UserX_Classifier.txt, UserX_Sequences.txt, UserX_TestSequences.txt, UserX_ClustersWithScores.txt, and UserX_Testing_ClustersWithScores.txt. The Classier.txt file has the classification of each test session as Masquerade and Proper session based on the sequence rating mechanism chosen. Each session also has sequence's rating along with its classification information. The Sequences.txt and TestSequence.txt files contain the sequences that were formed during training and testing phases respectively. Finally, the ClustersWithScores.txt and Testing_ClustersWithScores.txt files contain the clusters that were formed during training and testing phases. A brief overview of these two detectors is given in the following section.

5.2. An Overview of Probabilistic and Clustering Intrusion Detectors

5.2.1 Bayesian Networks Intrusion Detector

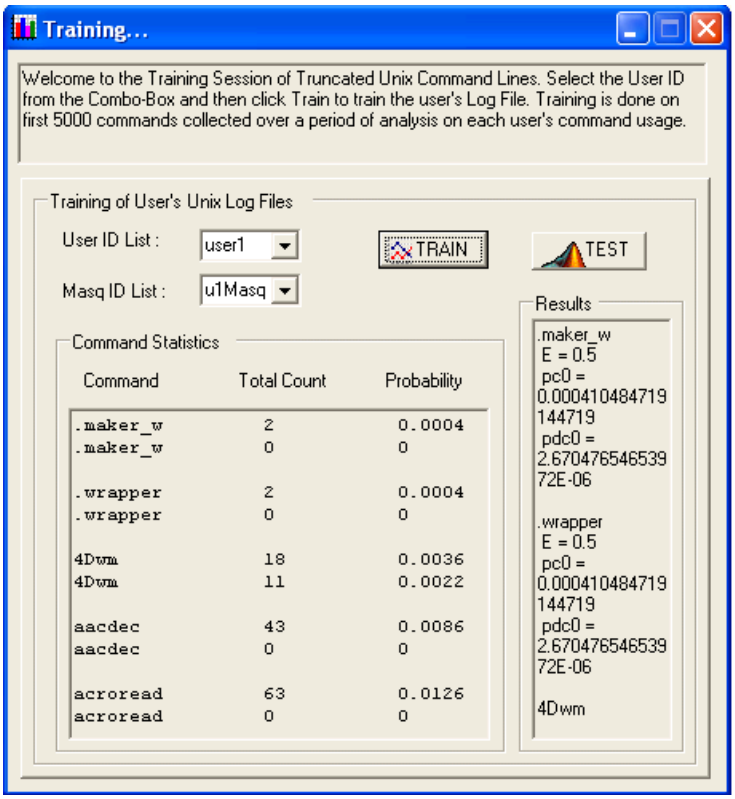

Figure 9: A snapshot of Bayesian Networks Intrusion Detector after training phase.

In figure 9, the Command Statistics group box shows the histogram of commands and the Probabilistic Model for each command is shown in the Results group box. In order to compute a proper and masquerade model for each command used by UserX, the UserX's log file and a Maquerade log file should be selected from the UserID List and Masq ID List respectively. Clicking on the Train button now would generate the initial probabilistic model for each commands based on the first 5000 commands in the chosen user's log file. The Command Statistics group box and the Results group box display the initial model parameters θ_0 of each commands used so far by the user.

To test the command sessions simply click on the Test button. A message box containing the online classification of the session under investigation is displayed. This continues for all the test sessions. Internally two algorithms are executed in parallel. As explained in section 3.2.3 the trust node toggles between the Naïve Bayes and Deferred Naïve Bayes Algorithms. Finally the results of classification are backed up in UserX_Classifie.txt file.

5.2.2 Clustering based Intrusion Detector

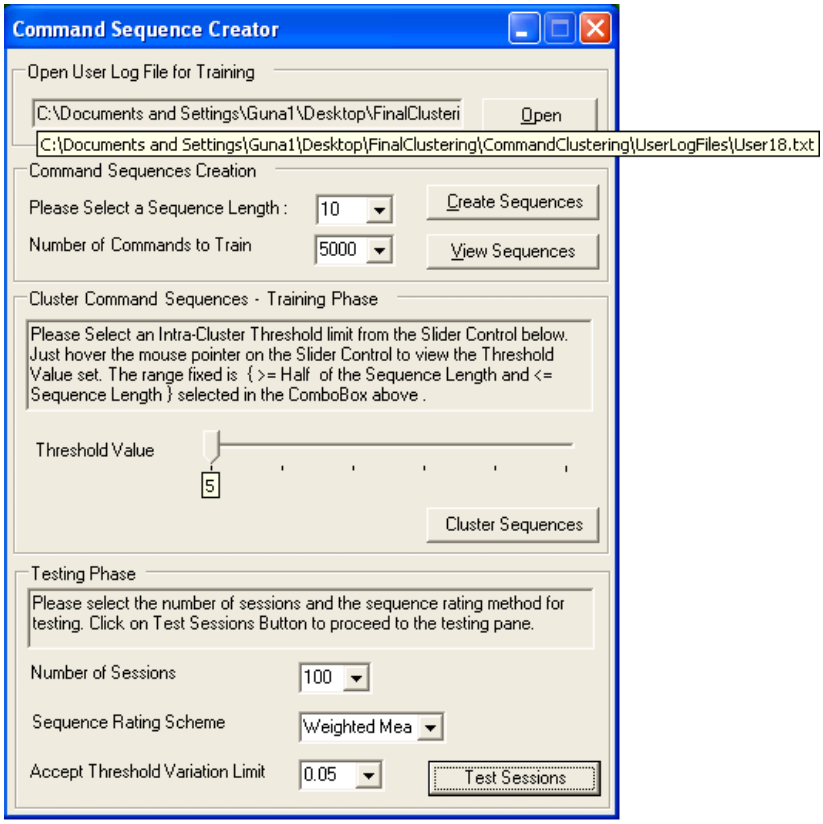

Figure 10: A snapshot of Clustering Analyzer of command sequences when analyzing User18.txt log file.

5.2.2.1 Steps to perform clustering analysis

1. Open a User log file for training using the Open button
2. Select the Sequence Length and the Number of Commands to train from the combo boxes in Command Sequences creation group box.
3. Click on Create Sequences to create command sequences. The output file UserX_Sequences.txt will be created at this step.
4. To view the created sequences click on View Sequences button. This would pop up a dialog box showing the command sequences that were created.
5. Set a threshold value using the slider control and click on the Cluster Sequences button. This results in the creation of UserX_ClustersWithScores.txt file that contains the clusters with a center sequence and the clustered sequences appended with their similarity Score with the center sequence.
6. We now enter the testing phase. To test the command sessions select the number of sessions to test, the sequence rating mechanism and an Accept threshold value; click on the Test Sessions button to perform profile based testing.
7. The result of Testing is three text files namely UserX_TestSequences.txt that contains all the sequences that were tested, UserX_Testing_ClustersWithScores.txt that contains the updated clusters that were formed during the testing phase and UserX_Classifier.txt that contains classified test sessions based on the Accept threshold and the Sequence rating mechanism used on the test sessions.
8. Results in UserX_Classifier.txt can be used to plot ROC Curves for clustering analysis.

5.3 Results and Discussions

5.3.1 Probabilistic analysis of truncated UNIX commands

5.3.1.1 A modified data scenario

Probabilistic analysis can also be carried out on command sessions with modified UNIX commands. Truncated UNIX commands from the Schonlau's Masquerade data files can be appended with arguments to enrich the data. A data pre-processing step will be used to enrich some of the commands. For example, the truncated command 'vi' can be appended with the file name of the file that was viewed with the editor i.e. 'vi' can be enriched to 'vi <File Name>'. The file name in the angle bracket now distinguishes this 'vi' command from other 'vi' commands in the user dataset.

In the masquerade detection scenario we assume that every user considered has a pattern of command usage based on the current project. Moreover, the user profiles are revamped based on the active project that he/she works on. The enrichment of some of the commands would certainly work in such situations. In case if the intruder targets on opening some files that the privileged user cares no more due to project change, this could be marked as an intrusion. Such patterns of file accesses can be added into the user log file's test commands region and considered as a masquerading data pattern. To be more practical the training data can also added with some enriched commands to account for the enrichment in the test data.

In our probabilistic analysis we only considered truncated UNIX commands from the Schonlau's User log files. We limited our analysis for four users and followed the same for the clustering analysis also. It is beyond the scope of this thesis to include enriched commands and may be considered as future work.

5.3.1.2 Analysis

A limited analysis was carried out due to time constraints. The following discussion explains how the proper and masquerade data model was built for four users. The Schonlau data for 50 users was used as a user data pool and just 4 users were picked from it to carry out our analysis. For these four users a proper and masquerade data model was built and the sessions were tested based on the built models.

We Considered analysis of User1, User9, User10 and User18 log files for our masquerade detection problem. For detecting masquerading test sessions of User1 the probabilistic analysis was carried out as follows: the first 5000 commands from the log file of User1 was trained to build a proper user model for User1. To build a masquerade model for User1, 5000 commands in sets of 100 commands, making 50 sessions were chosen from the rest of the users and trained. The result of training was the model parameter θ (ε_0, p_{c0}, p'_{c0}) for each command, which was computed from log likelihood estimation formulas.

During the testing phase new model parameters were computed using the likelihood estimation formulas and finally the difference between these parameters and the previous step's parameters were computed and compared to a threshold. If this threshold limit was not crossed then the test session was reported as a masquerade session, else the new session was updated by re-calculating the model parameter. The Bayesian Networks toggles between a Naïve Bayes and Deferred Naïve Bayes Algorithm. Based on the trust mechanism explained in section 3.2.3 the best classifier is chosen at proper time and the online classification of test sessions is continued using the

Naïve Bayes classifier after the its current model is replaced by the Deferred Naïve Bayes classifier's offline model.

The results of our above analysis using Bayesian Network method are shown below. In this discussion only the sets of classifier graphs for four users that point out the proper and masquerade sessions are attached. After discussing the Clustering analysis the hit rate to false alarm rate graphs are discussed as ROC curves.

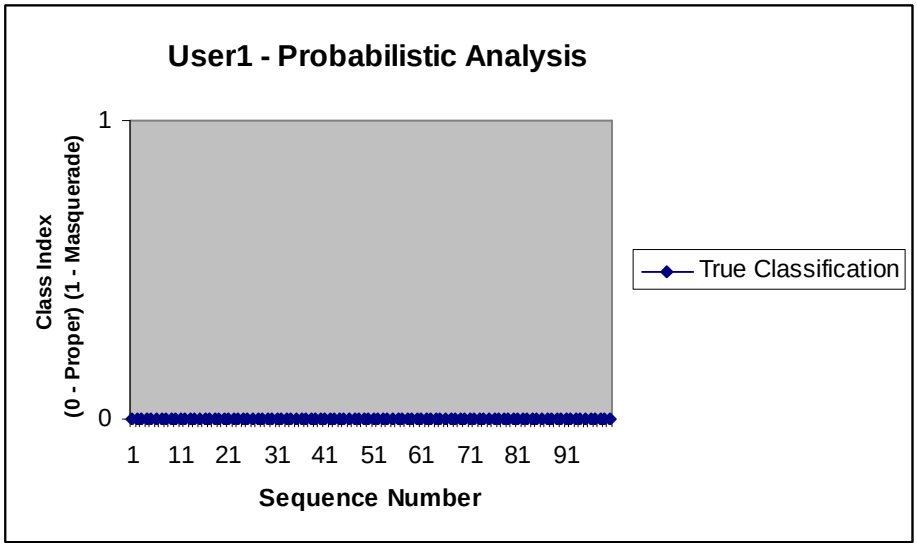

Figure11.1: True classification of 100 Test Sessions for User1

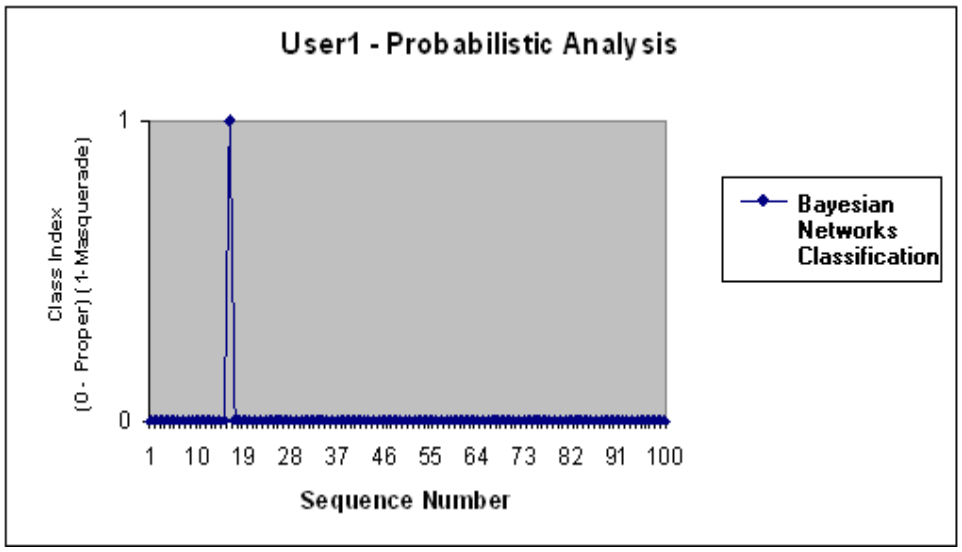

Figure 11.2: Bayesian Network Classification of 100 Test Sessions for User1

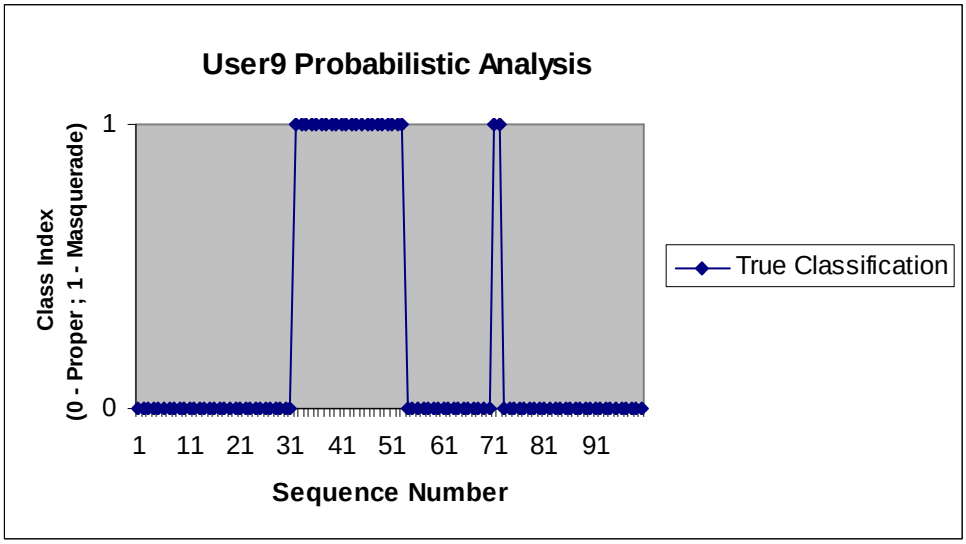

Figure 12.1: True classification of 100 Test Sessions for User9

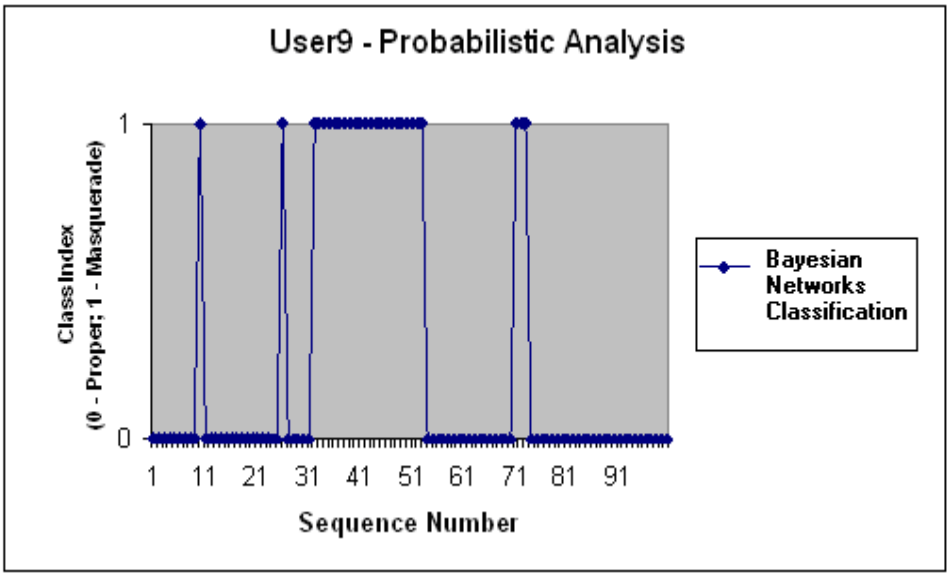

Figure 12.2: Bayesian Network Classification of 100 Test Sessions for User9

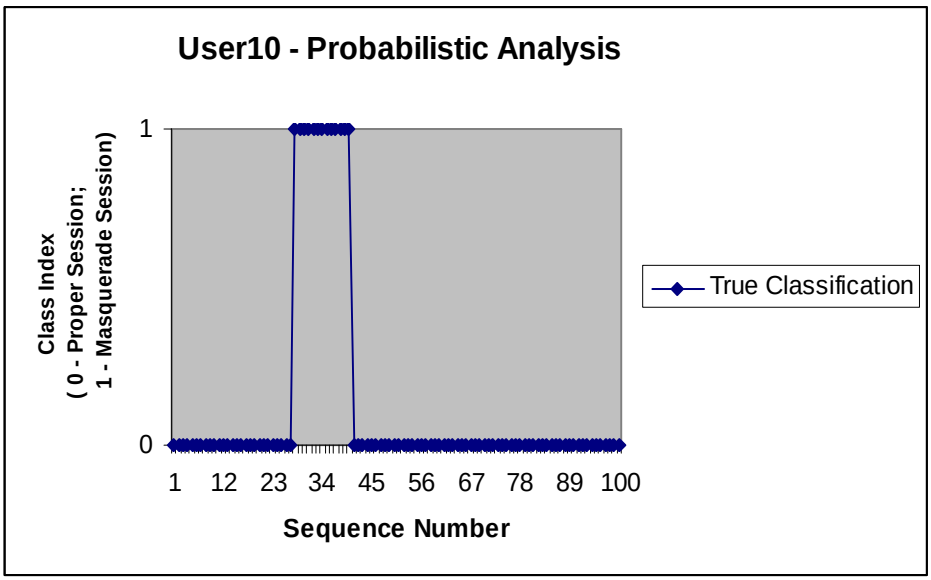

Figure 13.1: True classification of 100 Test Sessions for User10

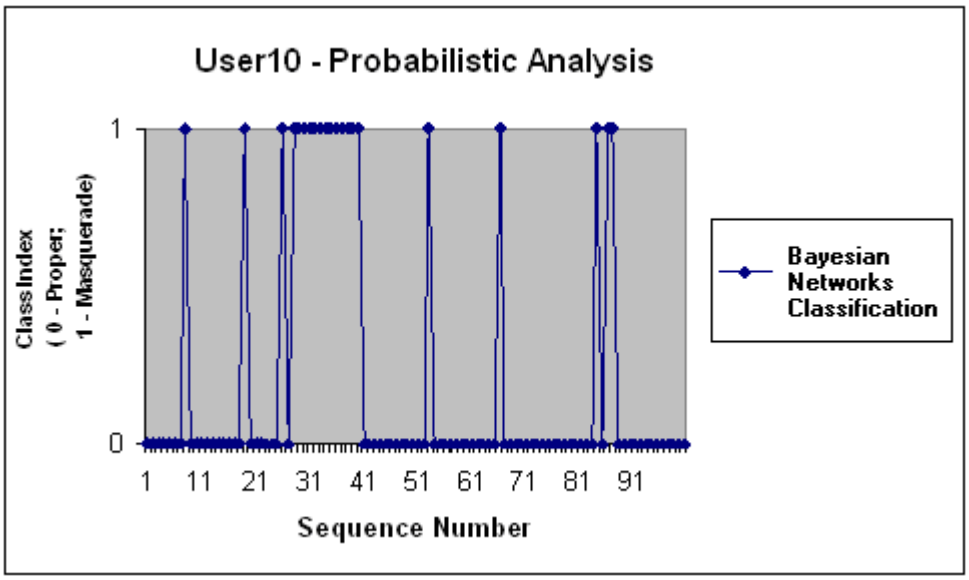

Figure 13.2: Bayesian Network Classification of 100 Test Sessions for User10

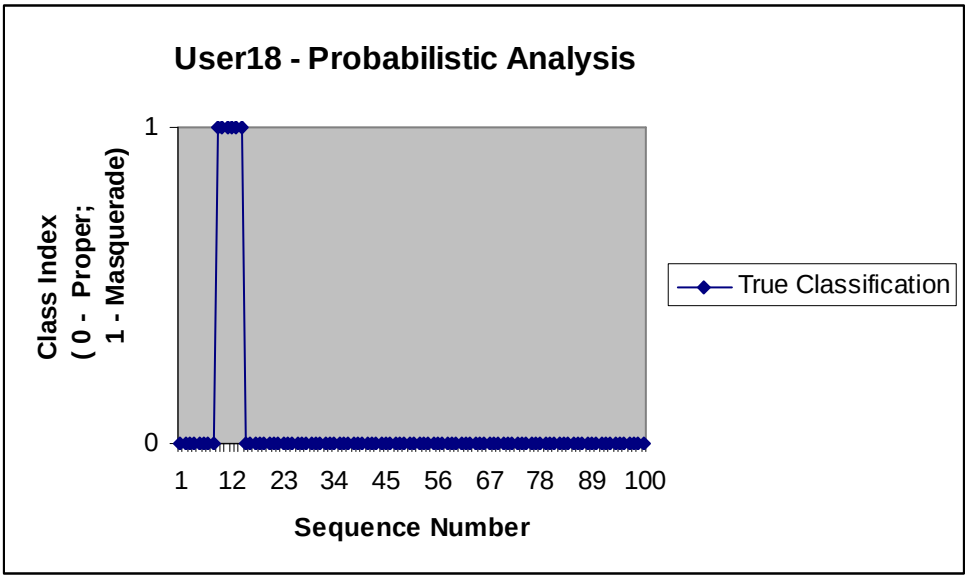

Figure 14.1: True classification of 100 Test Sessions for User18

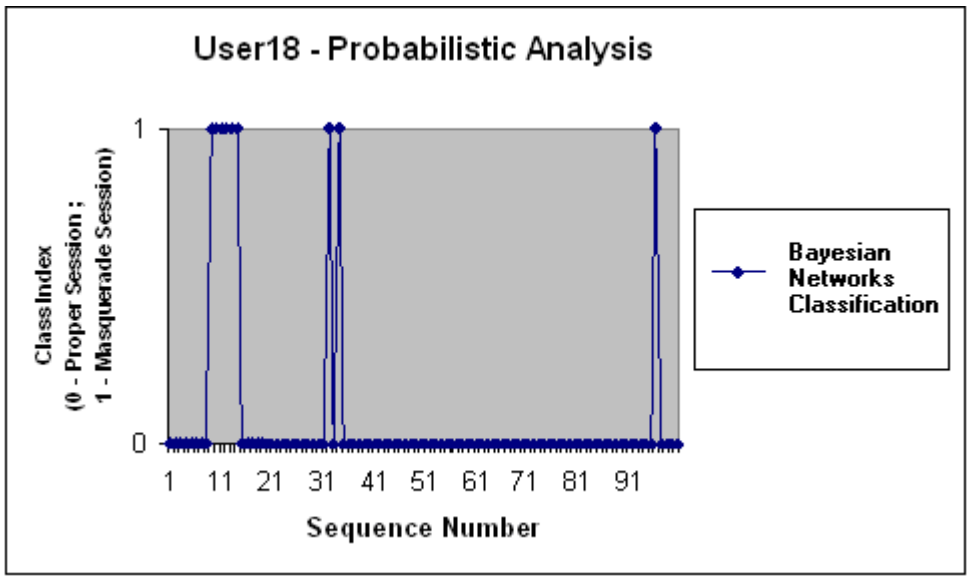

Figure 14.2: Bayesian Network Classification of 100 Test Sessions for User18

5.3.2 Clustering Analysis of Truncated UNIX Command Sequences

Like most of the clustering scenarios, clustering of commands was also carried out in two phases. Clustering requires command sequences. We can perform a simple analysis by considering a command session as a command sequence. The following criteria were considered while forming command sequences in our clustering analysis.

- If sequence length < session length then we formed overlapping sequences in both the training and testing phases.
- If sequence length = session length then we formed overlapping sequences only in the training phase.
- Sequence length >=10 and <=100

For instance, let us consider a command session of length 5 containing five commands, {vi, csh, pico, netscape, pine}. An example of sequences of length 4 would be {vi , csh, pico, netscape} and {csh, pico, netscape, pine}. These sequences are our overlapped sequences. To explain clustering of command sequences let us divide our discussion into two phases namely training and testing.

5.3.2.1 Training Command Sequences

The main goal of the training phase is to create user profile that would be used to test command sessions during the testing phase. The Sequence Creator first creates command sequences and Cluster Creator then clusters these sequences using our clustering algorithm in section 4.3. A profile of user is finally constructed with the center sequences.

In order to cluster a command sequences with center sequences we use a "similarity function" that has to satisfy the following criteria i.e. the Sum of Longest Common Subsequence (LCS) and Match Count Adjacent Reward Polynomial bound (MCAP) between the sequence and it's center should be greater or equal to a threshold value. If this criterion is satisfied then we cluster the sequence with the center sequence and then a new cluster center will be recalculated from all the sequences in the cluster. This is done to adjust to the change that was caused due to addition of a new sequence to the cluster.

We performed Clustering analysis on the same 4 users upon whom we performed Probabilistic analysis. This time Schonlau's truncated UNIX command data was used.

During the training phase the first 5000 commands in the user log file was used to build a profile for that user. The remaining commands in the log file were used to as test commands. The testing phase is explained below.

5.3.2.2 Testing Command Sequences

The commands after 5000 commands were analyzed in sessions of 100 commands. The cluster centers in the user profile formed during the training phase were used and a match between these centers and the test sequences were established based on the same similarity criteria that was used in training. Actually using the same similarity criteria doesn't produce accurate results. So to account for more accuracy each test sequence was rated based on the three methods explained in section 4.5. Our clustering analysis of test sequences was also based on these methods. We analyzed the test sequences of User1, User9, User10 and User18 based on these three rating mechanisms for a sequence length of 100. The results of these analyses are shown below.

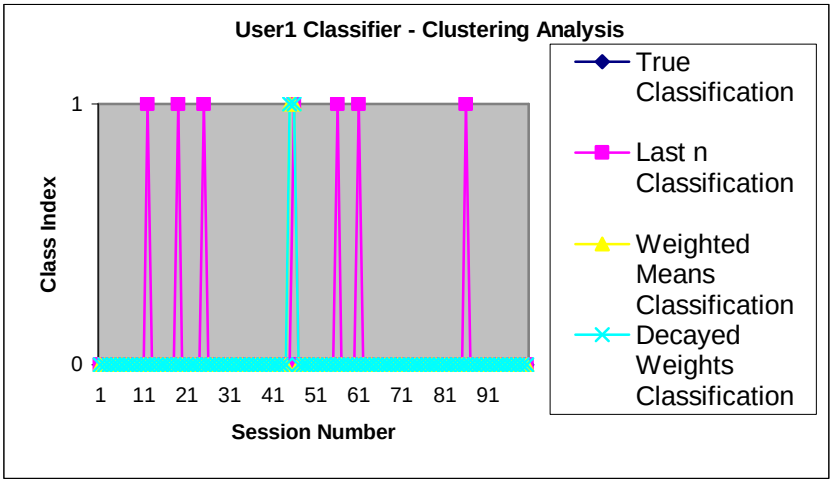

Figure 15: Classifications of 100 Test Sessions based on Clustering Analysis for User1.

From Figure 15 we find that Last n Sequence rating method produces more false alarms than Weighted Means and Decayed Weights methods. Test Sessions 12, 19, 25, 46, 56,62 and 86 were classified as masquerade sessions by Last n rating method. Weighted means and Decayed Weights methods produced sessions 45 and 46 as masquerades.

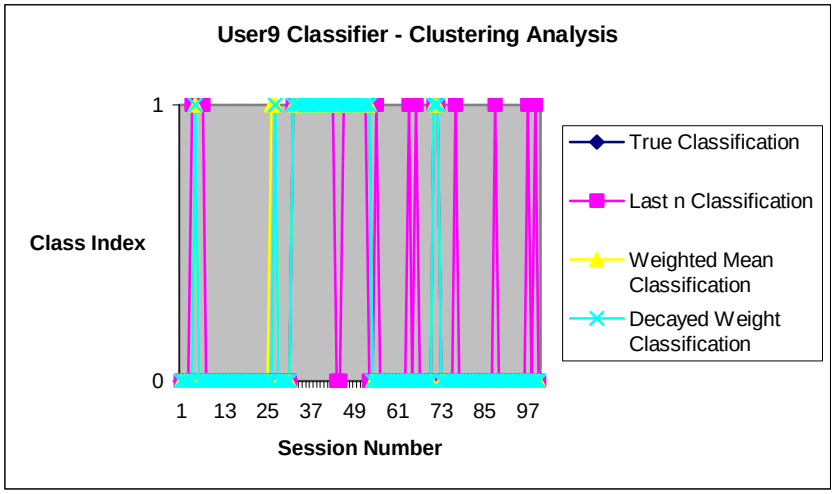

Figure 16: Classifications of 100 Test Sessions based on Clustering Analysis for User9.

Last n Sequence rater produced the worst results for User9; it classified test sessions 4, 5, 6, 7, 55, 64, 66, 77, 88, 96 and 99 as Masquerade sessions. So the false alarm count is 11. There were also two missing alarms in this classification, i.e. sessions 44 and 45 were classified as proper sessions. Weighted means had 3 false alarms (Test sessions 5, 26 & 27) and no missing alarms. Finally Decayed weights produced a better result with just two false alarms (Test sessions 5 and 26).

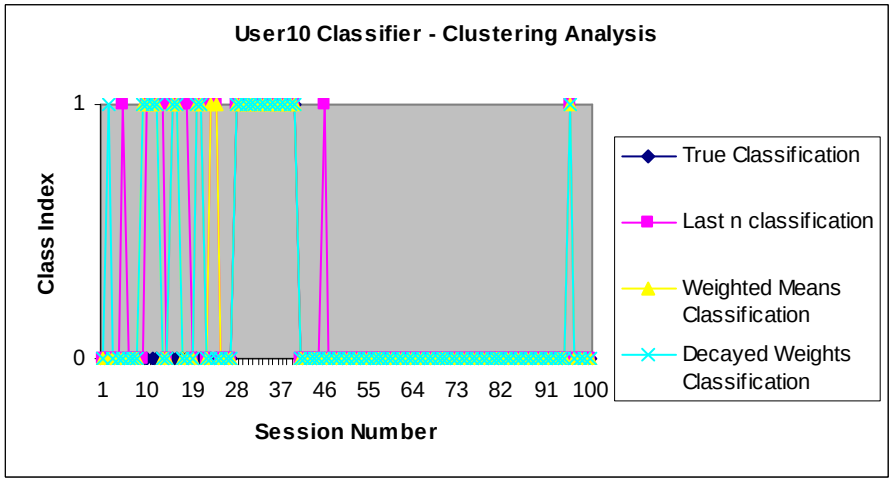

Figure 17: Classifications of 100 Test Sessions based on Clustering Analysis for User10.

For User10 Last n sequence rater produced 15 false alarms but there were no missing alarms. This is better compared to the previous user, as there were two missing alarms, which is worse. Weighted means produces 11 false alarms and Decayed weights stayed on 9.

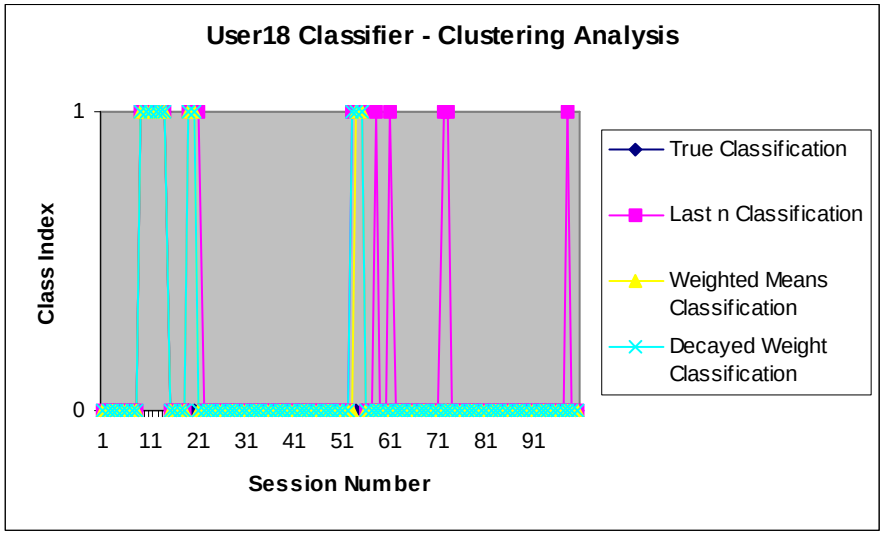

Figure 18: Classifications of 100 Test Sessions based on Clustering Analysis for User18.

For User18 there were no missing alarms and 9, 4, 5 were the false alarms produced by Last n, Weighted and Decayed weights sequence rating techniques respectively. Surprisingly here the Decayed weights method produced one extra false alarm. This might have been due to the slightly incorrectly tuned accept threshold value.

The graphical representation of clustering analysis is not very clear since three techniques where displayed on to the same graphs. Table 3 provides the results of clustering techniques in a clearer manner. The table is divided into 4 sections, one each for User1, User9, User10 and User18 respectively. In User1's section we provide a column for the test session number (SNo.). This can be used as a reference for the other sections. Each of the four sections has 4 columns representing the True Classification (TC), Last n Classification (LN), Weighted Means Classification (WM) and Decayed Weights Classification (DW) of the test sessions. True classification columns have 0's for proper sessions and 1's for masquerade sessions. The other three columns of each section have a "+" to represent correct classifications and a "-" to represent incorrect classifications. Correct classifications include true positives and true negatives and incorrect classifications include false positives or false alarms and false negatives or missing alarms.

User1					User9				User10				User18			
S No.	TC	LN	WM	DW	TC	LN	WM	DW	TC	LN	WM	DW	TC	LN	WM	DW
1	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
2	0	+	+	+	0	+	+	+	0	+	+	-	0	+	+	+
3	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
4	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
5	0	+	+	+	0	-	-	-	0	-	+	+	0	+	+	+
6	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
7	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+

8	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
9	0	+	+	+	0	+	+	+	0	+	-	-	1	+	+	+
10	0	+	+	+	0	+	+	+	0	-	-	-	1	+	+	+
11	0	+	+	+	0	+	+	+	0	-	-	-	1	+	+	+
12	0	-	+	+	0	+	+	+	0	-	-	-	1	+	+	+
13	0	+	+	+	0	+	+	+	0	-	+	+	1	+	+	+
14	0	+	+	+	0	+	+	+	0	+	+	+	1	+	+	+
15	0	+	+	+	0	+	+	+	0	-	-	-	0	+	+	+
16	0	+	+	+	0	+	+	+	0	-	-	-	0	+	+	+
17	0	+	+	+	0	+	+	+	0	-	+	+	0	+	+	+
18	0	+	+	+	0	+	+	+	0	-	+	+	0	+	+	+
19	0	-	+	+	0	+	+	+	0	+	+	+	0	-	-	-
20	0	+	+	+	0	+	+	+	0	-	-	-	0	-	-	-
21	0	+	+	+	0	+	+	+	0	-	-	-	0	-	+	+
22	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
23	0	+	+	+	0	+	+	+	0	-	-	+	0	+	+	+
24	0	+	+	+	0	+	+	+	0	-	-	+	0	+	+	+
25	0	-	+	+	0	+	+	+	0	+	+	+	0	+	+	+
26	0	+	+	+	0	+	-	+	0	+	+	+	0	+	+	+
27	0	+	+	+	0	+	-	-	0	+	+	+	0	+	+	+
28	0	+	+	+	0	+	+	+	1	+	+	+	0	+	+	+
29	0	+	+	+	0	+	+	+	1	+	+	+	0	+	+	+
30	0	+	+	+	0	+	+	+	1	+	+	+	0	+	+	+
31	0	+	+	+	0	+	+	+	1	+	+	+	0	+	+	+
32	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
33	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
34	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
35	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
36	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
37	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
38	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
39	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
40	0	+	+	+	1	+	+	+	1	+	+	+	0	+	+	+
41	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
42	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
43	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
44	0	+	+	+	1	-	+	+	0	+	+	+	0	+	+	+
45	0	+	-	-	1	-	+	+	0	+	+	+	0	+	+	+
46	0	-	-	-	1	+	+	+	0	-	+	+	0	+	+	+
47	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
48	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
49	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
50	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
51	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
52	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+

53	0	+	+	+	1	-	+	+	0	+	+	+	0	-	+	-
54	0	+	+	+	0	+	+	+	0	+	+	+	0	-	-	-
55	0	+	+	+	0	-	+	+	0	+	+	+	0	-	-	-
56	0	-	+	+	0	+	+	+	0	+	+	+	0	+	+	+
57	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
58	0	+	+	+	0	+	+	+	0	+	+	+	0	-	+	+
59	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
60	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
61	0	-	+	+	0	+	+	+	0	+	+	+	0	-	+	+
62	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
63	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
64	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
65	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
66	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
67	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
68	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
69	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
70	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
71	0	+	+	+	1	+	+	+	0	+	+	+	0	+	+	+
72	0	+	+	+	1	+	+	+	0	+	+	+	0	-	+	+
73	0	+	+	+	0	+	+	+	0	+	+	+	0	-	+	+
74	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
75	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
76	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
77	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
78	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
79	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
80	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
81	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
82	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
83	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
84	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
85	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
86	0	-	+	+	0	+	+	+	0	+	+	+	0	+	+	+
87	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
88	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
89	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
90	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
91	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
92	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
93	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
94	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
95	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+
96	0	+	+	+	0	+	+	+	0	+	-	+	0	+	+	+
97	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+

98	0	+	+	+	0	+	+	+	0	+	+	+	0	-	+	+
99	0	+	+	+	0	-	+	+	0	+	+	+	0	+	+	+
100	0	+	+	+	0	+	+	+	0	+	+	+	0	+	+	+

Table 3: Classifications for the 100 test sessions recorded for Clustering Analysis.

The Relative Operating Characteristic Curve for three clustering techniques and the Self-Consistent naïve Bayes technique is given below. The three clustering techniques and the Bayesian Network techniques are analyzed on their hit rate and false alarm rates produced while detecting masquerades on the 4 user log files.

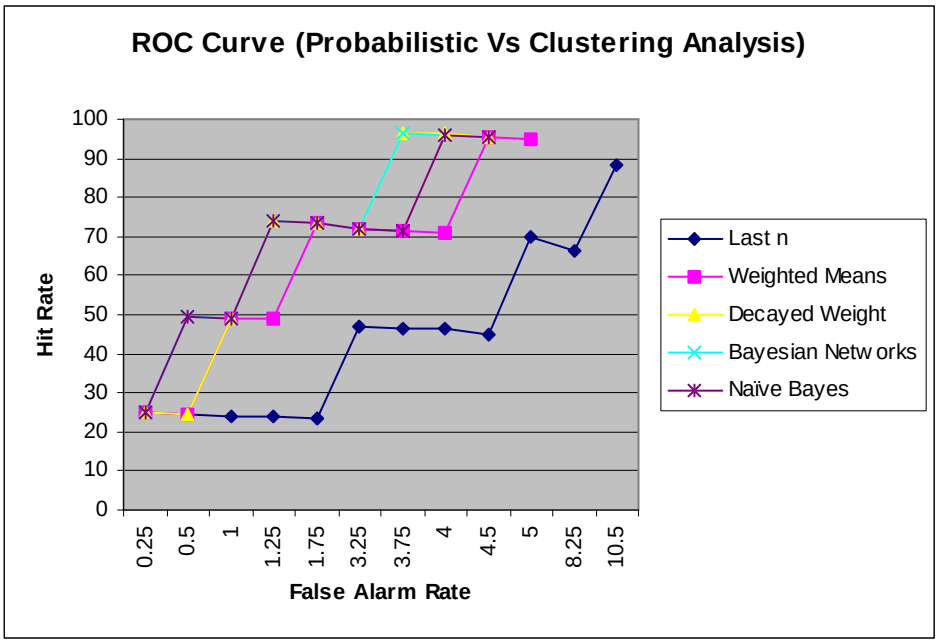

Figure 19: ROC Curve for the three Clustering, Bayesian Network and Naïve Bayes Classification techniques.

Table 4 below is the tabular representation of the Hit Rate and False Alarm Rate for the five classification techniques shown in Figure 19.

FAR	HR				
	LN	WM	DW	BN	NB
0.25	24.75	24.75	24.75	24.75	24.75
0.5	24.5	24.5	24.5	49.5	49.5
1	24	49	49	49	49
1.25	23.75	48.75	73.75	73.75	73.75
1.75	23.25	73.25	73.25	73.25	73.25
3.25	46.75	71.75	71.75	71.75	71.75
3.75	46.25	71.25	96.25	96.25	71.25
4	46.5	71	96	NA	96
4.5	45	95.5	95.5	NA	95.5
5	70	95	NA	NA	NA
8.25	66.25	NA	NA	NA	NA
10.5	88.5	NA	NA	NA	NA

Table 4: Hit Rates for the five classification techniques and their respective false alarm rates.

In table 4 FAR = False Alarm Rate; HR = Hit Rate; LN = Last n; WM = Weighted Means; DW = Decayed Weights; BN = Bayesian Networks; NB = Naïve Bayes; NA = Not Applicable. From the above table it is evident that probabilistic analysis produced a better result for masquerade detection than clustering analysis on truncated commands. Bayesian Networks classifier produced a false alarm rate of 3.75% for a hit

rate of 96.25%. This is comparatively better compared to previous techniques in the literature but again our analyses were limited to just 4 users and hence our results were better. Also the Decayed weight analysis produced a better result than the other two sequence rating methods. The false alarm rates produced by decayed weights technique for a hit rate of 95.5% was 4.5%. Weighted means produced a false alarm rate of 5% for a hit rate of 95%. Last n method had a hit rate of 88.5% for a missing alarm rate of 10.5%. The remaining 1% in Last n technique was missing alarm.

Moreover, our Bayesian networks classifier produced better results than the Naïve Bayes classifier. We obtained a hit rate of 95.5% for a false alarm rate of 4.5% for the Naïve Bayes classifier for the same analysis for which our Bayesian Networks classifier produced a hit rate of 96.25% with a false alarm rate of 3.75%. The Decayed Weight clustering analysis on truncated data produced almost identical results as Naïve Bayes classifier. Last n sequence rating analysis being very simple produced poor results.

We chose an accept threshold value for our clustering technique after a few empirical analysis and hence were able to produce slightly poor results for the Decayed weight clustering mechanism as compared to the probabilistic analysis. A well chosen accept threshold limit may possibly surpass the results of probabilistic analysis.

5.3.3 Feasibility of Clustering Analysis.

Training phase of clustering analysis was slow for the training set of 5000 commands and a sequence length of 10. This was because we had to analyze too many overlapped sequences and cluster them. It would be an overwhelming task on real-time if we need to cluster huge volume of command line data with a smaller sequence length.

One way to speedup clustering analysis is to increase sequence length to a limit that would produce better result in less time. We carried out our clustering analysis on a sequence length of 10 and the results produced were comparable to probabilistic analysis.

In real time one would expect an optimal sequence length Clustering algorithm to be implemented based on the volume of data. We can imagine an automated algorithm that adjusts its sequence length based on the volume of data to be tested. This could cause more work to form new clusters of different length. However this thesis considered a user log file of 15000 commands and there weren't much problems for the algorithm running to completion for a sequence length of 10. In reality building small command sequences to cluster would cost more time.

CHAPTER VI

CONCLUSION

Detecting masquerade using profile creation is just one component of a security system that also has other mechanisms to monitor and detect intrusions. These may include cameras, keystroke monitors and well-equipped systems for surveillance. The proposed command history analysis mechanisms, namely, Bayesian Networks and clustering methods further enhance the systems that already exist. Our proposed system further improves the security of systems since it is an automated approach.

Although over the years different mechanisms have been tried out to train user profiles and detect intruders, clustering techniques are still in their infancy. We proposed a novel Hybrid Bayesian network for masquerade detection as well as novel clustering algorithm for masquerade detection.

Results show that the hybrid Bayesian network produces the best results. However this was for a limited dataset and with a larger number of dataset or users, the clustering approach may produce better results. The simulation work for clustering mechanism and Bayesian mechanism was illuminating, especially as an elaborate study of three rating mechanisms showed that Decayed weights was better than the other two methods since it used the varying alpha parameter.

The ROC curve in figure18 shows that probabilistic analysis produced better results than clustering analysis. The probabilistic method was simple and produced slightly better results than clustering on Schonlau data. The false alarm rate and hit rate recorded by these techniques are explained in section 5.3.2.2. As explained before, we

analyzed only four users log files and recorded these results. Surprisingly the probabilistic analysis that didn't consider sequence information of sessions produced a better result than clustering analysis that considered sequence information in the form of LCS. A session can thus be considered as a bag of words in this case and still produces better results since we are mainly focusing on the frequency of commands seen in proper and masquerade sessions. This clearly explains the success of the Naïve Bayes classifier for session classification. Moreover our Bayesian networks classifier being a hybrid online-offline technique produced better results than Naïve Bayes classifier.

Developing a real-time Intrusion Detection System based on the clustering algorithm would require analyzing the data files from the disk rather than trying to analyze the data using RAM. This is because the log files are huge and it would take a lot of time to form and cluster sequences. A real time online detection with a user keying in the data on a system would be challenging and interesting. Only such an analysis would fully reveal the comparative accuracy of the clustering and probabilistic analysis for detecting masquerades.

6.1 Future Work

As we discussed in section 5.3.1.1 probabilistic and clustering analysis on enriched data could be promising. Furthermore, tuning the accept threshold value for optimal results could also be a promising analysis. This could be separately worked out for both the truncated and enriched data. Our limited analysis can be further extended for 50 users on the Bayesian Networks approach. Since Clustering approach was slow for a sequence length of 10 we had to limit our analysis to 4 users. A real time clustering

detector would require a better algorithm and sophisticated systems with huge RAM. Since we tested our algorithm with a sequence length of 10 we were not able to perform a comparative analysis of both the schemes on all 50 users.

All the proposed techniques in the literature and in this thesis produce false alarms to some extent. The false alarm rate has to be kept low. We can only achieve a false alarm rate reduction by proposing new methods. Since Concept drift cannot be avoided in real time false alarms will exist. Hence designing a 100% accurate real-time Intrusion detection system is impossible.

REFERENCES

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

2. Denning D.E., "An Intrusion-Detection Model", *Proceedings of the 1986 IEEE Symposium on Security and Privacy (SSP '86)*, pp. 118-133, IEEE Computer Society Press, April 1990.

3. Denning D.E., Neumann P.G., "Requirements and Model for IDES – A Real-Time Intrusion Detection System", *Proceedings of the 1986 IEEE Symposium on Security and Privacy (SSP '86)*, pp. 118-133, IEEE Computer Society Press, April 1990.

4. DuMouchel W., "Computer Intrusion detection based on Bayes factors for comparing command transition probabilities", *Technical report 91, National Institute of Statistical Sciences,* February 1999.

5. Guan Y., Ghorbani A., Belacel N., "Y-Means: A Clustering Method for Intrusion Detection", *Proceedings Canadian Conference on Electrical and Computer Engineering,* May 3-4, 2003.

6. Heckerman. D, "A Tutorial on Learning with Bayesian Networks", *Microsoft Research report*, MSR-TR-95-06, 1995.

7. Ju W., Vardi Y., "A hybrid high-order Markov chain model for computer intrusion detection", *Technical report 92, National Institute of Statistical Sciences*, February1999.

8. Kanungo Tapas, Mount David M., Netanyahu Nathan S., Piatko Christine D., Silverman Ruth and Wu Angela Y., "An Efficient K-Means Clustering Algorithm: Analysis and Implementation"**,** *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, December 2000.

9. Lunt Teresa F., Jagannathan R., Rosanna Lee, Alan Whitehurst, "Knowledge-Based Intrusion Detection", *Proceedings of the AI Systems in Government Conference,* 1989.

10. Marin Jack, Ragsdale Daniel, Surdu John, "A Hybrid approach to profile Creation and Intrusion Detection", *Proceedings, IEEE DARPA Information Survivability Conference and Exposition (DISCEX II)*, 2001.

11. Maxion R.A., "Masquerade detection using Enriched command lines", *Proceedings International Conference on Dependable Systems and Networks*, IEEE Computer Society Press, 2003.

12. Maxion R.A., Townsend T.N., “Masquerade detection using truncated command lines”, *Proceedings International Conference on Dependable Systems and Networks*, IEEE Computer Society Press, 2002.

13. Schonlau M., “Maquerading User Data”, http://schonlau.net/, last accessed on March 23rd 2005.

14. Schonlau M., Theus M., DuMouchel W., Ju W-H., Karr A.F., and Vardi Y., “Computer Intrusion: Detecting Masquerades”, *Statistical Science*, 16(1): 58-74, February 2001.

15. Szymanski Boleslaw K., Yongiang Zhang, “Recursive Data Mining for Masquerade detection and Author Identification, *Technical Report, ia04, Department of Computer Science, Rensselaer Polytechnic Institute*, 2003.

16. Teresa F. Lunt, Jagannathan R., “A Prototype Real-Time Intrusion-Detection Expert System”, *Proceedings of the IEEE Symposium on Security and Privacy*, April 1988.

17. Wang Yao, Vassileva Julita, “Bayesian Network Trust Model in Peer-to-Peer Networks”, *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004)*, September 2004.

18. Yung Kwong H., “Using Self-Consistent Naïve Bayes to Detect Masquerades”, *Stanford Electrical Engineering and Computer Science Research Journal*, 14-21, December 2003.

VITA

Karthic Gunasekaran

Candidate for the Degree of

Master of Science

Thesis: PROBABILISTIC Vs CLUSTERING ANALYSIS OF MODIFIED UNIX COMMAND LINES FOR MASQUERADE DETECTION.

Major Field: Computer Science

Biographical:

Personal Data: Born in Chennai, Tamil Nadu, India, on January 19, 1981, the son of R.Gunasekaran and Mrs. Dhanalakshmi Gunasekaran.

Education: Received Bachelor of Engineering in computer Science and Engineering from Madras University, Chennai, India in May 2002.Completed the requirements for the Master of Science Degree with a major in Computer Science at Oklahoma State University in July 2005.

Experience: January 2004 – December 2004: Supervisor of Pistol Pizza, on-Campus pizza shop at Kerr Drummond Dining Hall of Oklahoma State University, Stillwater. Internship offer from Statsoft, Tulsa from January 2005 to May 2005.