ANALYSIS OF MASQUERADE DETECTORS PERFORMANCE UNDER SYNTHESIZED SESSIONS

THESIS

Presented as a partial fulfillment of the requirements for the degree of

Master of Science in Electronics Engineering
Major in Telecommunications

Ing. Román Posadas López

Monterrey, N.L., December 2006
The members of the thesis committee recommended the acceptance of the thesis of Román Posadas López as a partial fulfillment of the requirements for the degree of Master of Science in:

Electronics Engineering

Major in Telecommunications

THESIS COMMITTEE

Jorge Carlos Mex Perera, Ph.D.
Advisor

Hugo Terashima Marín, Ph.D.
Synodal

Alejandro Parra Briones, M.Sc.
Synodal

Approved

Graciano Dieck Assad, Ph.D.
Director of the Graduate Program

December 2006
Este trabajo está dedicado con amor a:

Mis padres: Celia López Ocampo y Juan Posadas Aco,
Alejandra Lara B. por su amor incondicional,
Mis hermanos(as) por su ayuda y motivación.
Acknowledgments

I want to thank:

To my parents Celia López Ocampo and Juan Posadas Aco for their support and help given during my studies.

To my thesis advisor Jorge Carlos Mex Perera for all the time given, for their professional advice and friendship.

To my synodals, Hugo Terashima Marín, Ph. D. and Alejandro Parra Briones, M.Sc. for the valuable comments for the enhancement of this work.

To Alberto Martínez H. who helped me type this work.

To Tecnológico de Monterrey, campus Monterrey for giving me the opportunity to finish my graduate studies and to the Cátedra de Biométricas y Protocolos Seguros para Internet that supported this work.
Analysis of Masquerade Detectors Performance
Under Synthesized Sessions

Román Posadas López, M.Sc.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2006

Informatics security has nowadays become an important research topic given the impact of the computers for mankind. As computers become more important, so the interests, risks and informatics attacks. With this work we focus on a type of attack called masquerade attack, where someone impersonates other by using the other’s computer account privileges and accomplish malicious activities. We try to understand this problem and analyze the way masquerade detectors are built. The way these are built says too much about their limitations. These limitations could be used in order to build synthesized masquerade sessions that bypass such detection methods. These masquerade synthesized sessions are created by an intelligent type of masquerader that has enough knowledge of the normal behavior profile of the user to masquerade. In this thesis we analyze the relationship between the the performance of different masquerade detection methods under artificially created masquerade sessions. These sessions would be created using different properties and would affect differently to each method. The dataset provided by Schonlau, called SEA\(^1\), has been modified for including synthetic sessions created by masqueraders that we suppose have information about the behavior profile of the users intended to impersonate. As a consequence, this work provides an approach to synthesizing sessions when these are based on commands. The synthesizing of the sessions turns out to be more effective as more features are taken into account to create the masquerade sessions. We also propose a masquerade detection method that is more tolerant against synthesized datasets when these are built based on command frequencies and script frequencies. We compare the effects on six different methods that use frequency properties or sequential properties. These effects are shown by the known ROC (Receiver Operating Characteristics) curves. After analyzing the results, we could see that our proposed method outperforms the others, being capable of detecting masquerade sessions that the other methods could not detect.

\(^1\)SEA is available from http://www.schonlau.net.
Contents

List of Figures iii

List of Tables iv

1 Chapter 1 Introduction 1
1.1 Problem Description ........................................... 2
1.2 Objective ....................................................... 3
1.3 Justification .................................................. 3
1.4 Contribution ................................................... 3
1.5 Organization .................................................. 4

2 Chapter 2 Intrusion Detection Basics 5
2.1 Classification of IDSs ........................................... 6
  2.1.1 Masquerade Detection ....................................... 7
2.2 Masquerade Detection Methods ................................ 7
  2.2.1 Sequitur as a Grammar Extraction Algorithm .............. 8
  2.2.2 Hidden Markov Models ...................................... 10
  2.2.3 Non-Negative Matrix Factorization ......................... 12
  2.2.4 Uniqueness ............................................... 14
  2.2.5 Customized Grammar ....................................... 15
  2.2.6 Single Commands .......................................... 17
2.3 Summary ..................................................... 17

3 Chapter 3 Proposed Masquerade Detection Method 18
3.1 Schonlau Dataset ............................................. 18
3.2 Hybrid Proposed Method .................................. 20
  3.2.1 Session Folding Block ................................. 21
  3.2.2 Hidden Markov Model Block ......................... 23
3.3 Summary .................................................. 24

4 Chapter 4 Synthesis of Masquerade Sessions 25
  4.1 Summary .................................................. 28

5 Chapter 5 Simulation and Results 29
  5.1 Measurement of Detection Methods Performance .... 29
  5.2 Receiver Operating Characteristics (ROC) Curves .... 30
  5.3 Discussion ............................................... 34
  5.4 Implementation Issues of Detection Methods ......... 34
    5.4.1 HMM Based Detector ............................... 34
    5.4.2 NMF Based Detector ............................... 35
    5.4.3 Uniqueness ......................................... 35
    5.4.4 Customized Grammar ............................... 35
  5.5 Summary ................................................ 35

6 Chapter 6 Conclusions and Further Research 36
  6.1 Further Research ....................................... 37

References .................................................. 40

Vita ......................................................... 41
List of Figures

2.1 Classification of IDS ................................................................. 6
2.2 Left-to-right HMM ................................................................. 11
2.3 Matrix $V$ for a user with $m$ sessions and $n$ unique commands. ............... 14
2.4 Evaluation of a session with customized grammars. .................................. 16

3.1 Schonah et. al. dataset ............................................................. 19
3.2 Proposed detection mechanism architecture. ........................................ 21
3.3 Shows the grammar extraction steps. .............................................. 21
3.4 Session folding architecture ....................................................... 22

4.1 Histogram of user 1 training data. ............................................... 26

5.1 Performance of detection methods with SEA dataset. ............................ 30
5.2 Performance of detection methods with SEA-I dataset based on commands frequency. . . . . 31
5.3 Performance of detection methods with SEA-I dataset based on scripts frequency. .. 32
5.4 Performance of detection methods with SEA-I dataset based on scripts priority. .......... 33
5.5 Performance of detection methods with SEA-I dataset based on scripts length. .......... 33
List of Tables

2.1 Operation of the two Sequitur grammar properties. ............................... 9
2.2 Sessions used to build an HMM model .................................................. 11

3.1 Example of grammar extraction ............................................................... 21
3.2 Test Session Once Folded ................................................................. 23

4.1 Sea-I datasets. ............................................................................. 26
4.2 Average of commands and scripts ....................................................... 27
Chapter 1

Introduction

Informatics security became an important issue when it started to be a research topic in 1980 with the work of James P. Anderson, *Computer Security Threat Monitoring and Surveillance* [1]. Actual lifestyle has made security companies invent too many tools to guarantee integrity, confidentiality and availability of information technology (IT) systems. Intrusion detection refers to the real-time discovery of any activity that compromizes integrity, confidentiality and availability of an information technology system. An Intrusion Detection System (IDS) detects such activities and raises an alarm when these abnormalities occur.

There are many types of informatics attacks, the classification shown by Dorothy E. Denning [4] is:

- Eavesdropping and packet sniffing (passive interception of network traffic);
- Snooping and downloading;
- Tampering or data diddling (unauthorized changes to data or records);
- Spoofing (impersonating other users, e.g. by forging the originating e-mail address, or by gaining password access);
- Jamming or flooding (overwhelming a system’s resources, e.g., by an e-mail flood or HTTP requests);
- Injecting malicious code such as viruses and Trojan horses (via floppy disks or e-mail attachments);
- Exploiting design or implementation flaws (often buffer overflows, which overwrite other data and can be used to get control over a system);
• Cracking passwords and keys.

We focus on a form of spoofing, namely Masquerade Attacks. A masquerade attack is performed when someone uses other’s computer account to get access to the computer and by account privileges, accomplish anomaly activities. Such attacks can be detected by measuring deviations from normal behavior profiles. The way to construct normal behaviors is by the use of audit data, which can be, sequences of system calls, sequences of commands, folders opened, files accessed, time and location of login, keystroke timings etc.

The problem of masquerade detection has been highly studied with the work of Matthias Schonlau [18] (2001). Matthias Schonlau provided a dataset of 15000 commands for each of 50 users. This dataset was contaminated with masquerade data. From that point onwards there has been a lot of publications always proposing better methods of detection. All these methods have the disadvantage that work fine for only a reduced set of masqueraders (those found in the Matthias Schonlau dataset).

The Matthias Schonlau dataset has some disadvantages (see section 3.1), such as users labelled as masqueraders did not have such intention, they just worked as usual in a normal UNIX session, which is not a realistic masquerader condition. Publications that try to solve the masquerade detection problem on this dataset are not solving the case where the masqueraders act as real intelligent masqueraders, therefore we may not fairly judge those methods until tested with real masquerade sessions.

The problem of testing the detectors may be solved by creating a synthetic dataset, which can be the case where an intelligent masquerader creates a synthetic dataset using the behavior profile of the user intended to impersonate. This obviously is the worst case scenario, since, for the work presented here we suppose the masquerader has all the advantages (contains information about the behavior profile of the user to impersonate) in order to accomplish the malicious objective without being detected.

1.1 Problem Description

Once a masquerader illicitly obtains a computer account, he/she might be interested in accomplish his/her objective without being detected. From the masquerader point of view, the masquerade session will have to hide the real anomaly purpose. The real purpose is hidden by creating synthetic sessions with the behavior profile of the user to impersonate. If the masquerade session turns out to resemble very well the legitimate
user behavior, then the detector will not raise any alarm and the masquerader will successfully bypass the
detection mechanisms.

1.2 Objective

This thesis has several objectives. These are:

- Identify how the weaknesses of the masquerade detection methods can be used to create a masquerade
  synthesized session that hides real malicious purposes and therefore it is not detected.

- From the masquerader point of view, create synthesized masquerade sessions that bypass detection
  methods.

- Implement a masquerade detector that is robust against the previous synthetic sessions and measure
  its performance.

1.3 Justification

Informatics security becomes a critical topic when talking about bank transactions, databases, e-commerce
and private information. The impact of the subject is so important that security companies are continually
releasing products (software and electronic devices) that detect and prevent intrusions. In particular the
masquerader attack is a repetitive attack given that once the intruder obtains the computer account he will
continue using it. Besides if we consider that masqueraders continually improve the techniques to attack,
then we can see that masquerade detection represents a big opportunity to research.

1.4 Contribution

With the present work, we contribute in two aspects, the proposal of a masquerade detection method and the
approach at creating synthesized masquerade sessions. Both contributions lie on their respective hypothesis.

Contribution 1. We propose a hybrid masquerade detection method which combines the compression
of the audit information and also the use of hidden Markov models (HMM).

- Hypothesis 1. The use of a masquerade detection method based on hidden Markov models along with
  the use of a pre-processing mechanism where audit information is folded (compressed) provides a better
tolerance to masquerade synthesized sessions.
Contribution 2. We also provide an approach at synthesizing masquerade sessions when these are based on commands.

- Hypothesis 2. The creation of a synthetic masquerade session that resembles the best the behavior of certain user must take into account frequency properties and sequential properties at the same time.

1.5 Organization

This document is organized as follows. In Chapter 2, we give the basics about intrusion detection, the classification of IDSs and masquerade detection methods. Chapter 3 deals with a proposed masquerade detection method. Chapter 4 presents how masquerade sessions could be synthesized in order to bypass the methods previously shown in chapter 2. Chapter 5 shows the results and the performance graphics of the masquerade detection methods. Finally, Chapter 6 presents conclusions and future work.
Chapter 2

Intrusion Detection Basics

This chapter is intended to give the basics about IDS (Intrusion Detection Systems), its classification and some of the needed background to understand the Masquerade Detection methods that we treat throughout this thesis.

An intrusion, in computer science, may be defined as a set of actions that compromise the integrity, confidentiality and availability of an Information Technology (IT) system. The goal of an IDS is to detect such actions, prevent break-ins and raise alarms. Since intrusion detection has to do with data, the problem is inherently statistical [18]. The sources of data may be event log files, sequences of commands, system calls, time and date of files accessed, web pages visited, attack definitions, etc. From this data which is called audit data or training data it is possible to extract the main features and get a model of either normal behavior or abnormal behavior. Once the behavior model is obtained, then the classification of testing data is performed.

The extraction of main features has to do with frequency or transition properties, causal relationship [14], data mining [22] and so on. As classification methods we can find several approaches such as neural network [6], hidden Markov models [5, 7, 23], automaton [20], etc.
2.1 Classification of IDSs

Based on the 1999 publication *Towards a Taxonomy of Intrusion Detection Systems* [3], there is a classification of IDS according to several functional characteristics, see Figure 2.1.

![Figure 2.1: Classification of IDS](image)

Based on the detection scheme IDSs can be divided in other two categories, misuse IDS (MIDS) and anomaly IDS (AIDS). A MIDS looks for known malicious activity. They base their missing alarms to false alarms ratio in the amount of attack signatures or definitions they have, therefore they must be updated constantly in order to be capable of detecting the latest attacks. They make use of pattern recognition techniques. An AIDS builds a profile of normal behavior and test the computer activity against this profile. An AIDS also needs enough quantity of information to create this profile. The information needed by an AIDS may be sequences of commands, traces of system calls, files - folders accessed and even keystroke timing. Signature-based schemes (MIDS) gives less false alarms, however they are typically trivial to bypass simply by varying the attack slightly, much in the same way that polymorphic viruses evade virus checkers. Anomaly detection systems are more capable of detecting unknown attacks but they generate more false alarms.

Based on the behavior on detection, the IDS may be Active or Passive. The IDS is said to be Active if actively reacts to the attack by taking corrective or proactive actions. If the IDS merely generates alarms it is said to be Passive.

Based on the origin of audit data, there are two kinds of IDSs, network intrusion detection systems (nIDS), and hosts-based intrusion detection systems (hIDS). The formers act from the exterior of the systems they protect, identifying attacks towards one host or a complete network. This type of IDS analyzes only network traffic. The attacks may come from the Internet or from inside the network. A host-based intrusion
detection system acts like another software component inside the IT equipment it protects; they find software or operating system alterations and also events that may be related to an attack.

2.1.1 Masquerade Detection

A masquerader is a person that uses somebody else’s computer account to gain access and impersonate a legitimate user to complete a malicious activity. There are two types of masqueraders, An internal masquerader uses someone else’s account although they have their own account on a computer. External masqueraders illicitly use someone’s account because they do not have an account of their own on a computer.

Masquerade detection is concerned with the timely discovery of someone who has hidden his identity by impersonating a legitimate user on the computer. It is usually undertaken using an anomaly detection approach, since, as indicated in the previous section, aims at distinguishing any deviation from ordinary user behavior. This ordinary user behavior is built from a historical profile of the customs of a given user.

Based on the information used to make a decision, masquerade detection methods may be either local or global. A masquerade detection method is said to be local, if, when deciding whether a masquerader is present, it uses only the profile of the user to be protected. If the normal behavior of a user is also based on data from other users, it is said to be global. Since they are more informed, global detection methods are usually more accurate than local ones. However, they demand more computer effort.

Based on the validity of a user profile, masquerade detection approaches can be further split into two classes: i) those that update the user profile during the detection phase, and those that do not. Update might be local or global.

2.2 Masquerade Detection Methods

In this section we present the basics behind the methods used to detect masqueraders. The hidden Markov model [23, 7, 5, 25] is a classification method that builds a user model. This model is used to validate if a given working session corresponds to the behavior profile of the legitimate user. Another method based on the Non-Negative Matrix Factorization algorithm [26, 15] is used to extract features from the blocks of audit data associated with the normal behaviors. Deviation from the normal behavior above a predetermined threshold
is considered anomalous. *Customized Grammar* [11] is a method based on the *Sequitur* [13] algorithm and it uses only frequency properties of the grammar symbols associated with a given user. *Uniqueness* [19] is a method based on the idea of commands not previously seen in the training data may indicate a masquerade session. *Single Commands* [11] is another method based only on frequency of commands. Since one method and also the proposed one make use of the *Sequitur* algorithm, we will give an introduction of it.

### 2.2.1 *Sequitur* as a Grammar Extraction Algorithm

*Sequitur* is according with Nevill-Manning and Witten [13] an algorithm that infers a hierarchical structure from a sequence of discrete symbols by replacing repeated phrases with a grammatical rule that generates the phrase, and continuing this process recursively. What we get is a hierarchical representation (rules within rules) of the original sequence, which offers insights into its lexical structure.

For a formal description of the algorithm we must define $\Sigma$ as the set of terminals, which for this study would be UNIX commands. $N = \{n_1n_2...n_k\}$ is the set of nonterminals, which are the left hand side (lhs) of a production rule $P$. $S$ is the start symbol, also called main production, $S \notin N \cup \Sigma$. A production rule has the form $n_k \rightarrow x_1x_2x_3...x_n$ where $x_i$ is the right hand side (rhs) of a production rule, $x_i \in N \cup \Sigma$ and $n_k$ is the lhs of a production rule, $n_k \in N \cup \{S\}$. All the productions excluding $S$ are auxiliary productions.

Let $C = (c_i)$ be the sequence of elements $c_i \in \Sigma$ from which we will apply *Sequitur*. Everything starts by extracting elements from $C$ and assigning it to the main production; $S \rightarrow c_1c_2$. In that order *Sequitur* proceeds sequentially extracting elements from $C$ and adding elements to $S$. New productions are created and deleted maintaining the following two properties:

**Unique Digram.** No digram, i.e. pair of adjacent terminals or nonterminals, occur more that once in the grammar.

**Rule Utility.** Every rule is used more than once across all the right hand sides of the grammar.

*Sequitur* appends new symbols to rule $S$. The last two symbols of rule $S$ form a new digram. If this digram occurs elsewhere in the grammar, and the digram is not the rhs of any existing production, the constraint of *unique digram* is violated. To restore this problem, a new rule is created with the digram on the rhs, headed by a new non-terminal symbol. The two digrams are replaced by this new non-terminal symbol. Longer rules are created by the effect of the *rule utility* constraint. At first, short rules are formed
temporarily, and if subsequent rules continue the match, Sequitur creates a new rule that supersedes the shorter one and delete the latter.

As an example of how new rules are created using the *Unique Digram* and the *Rule Utility* properties we show Table 2.1 where successive commands of the sequence `rm,cd,ls,cd,ls,rm,cd,ls,cd,ls` are passed to the *main production* and a grammar is being constructed.

<table>
<thead>
<tr>
<th>symbol number</th>
<th>the string so far</th>
<th>resulting grammar</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>rm</code></td>
<td><code>S → rm</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>rm,cd</code></td>
<td><code>S → rm,cd</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>rm,cd,ls</code></td>
<td><code>S → rm,cd,ls</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>rm,cd,ls,cd</code></td>
<td><code>S → rm,cd,ls,cd</code></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><code>rm,cd,ls,cd,ls</code></td>
<td><code>S → rm,cd,ls,cd,ls</code></td>
<td><code>cd,ls</code> appears twice</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → rm,A,A</code></td>
<td>enforce digram uniqueness</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>rm,cd,ls,cd,ls,rm</code></td>
<td><code>S → rm,A,A,rm</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><code>rm,cd,ls,cd,ls,rm,cd</code></td>
<td><code>S → rm,A,A,rm,cd</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><code>rm,cd,ls,cd,ls,rm,cd,ls</code></td>
<td><code>S → rm,A,A,rm,cd,ls</code></td>
<td><code>cd,ls</code> appears twice</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → rm,A,A,rm,cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → B,A,B</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B → rm,A</code></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td><code>rm,cd,ls,cd,ls,rm,cd,ls,cd</code></td>
<td><code>S → B,A,B,cd</code></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td><code>rm,cd,ls,cd,ls,rm,cd,ls,cd,ls</code></td>
<td><code>S → B,A,B,cd,ls</code></td>
<td><code>cd,ls</code> appears twice</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B → rm,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → B,A,B,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B → rm,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B → rm,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → C,C</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>B → rm,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>C → B,A</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>S → C,C</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>A → cd,ls</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>C → rm,A,A</code></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Operation of the two Sequitur grammar properties.
In the symbol number 5, *Sequitur* adds the command `ls` to $S$, consequently the digram `cd,ls` appears twice. *Sequitur* creates the new rule $A$ with `cd,ls` as its right-hand side and replaces the occurrences of `cd,ls` by $A$. This illustrates the procedure for dealing with duplicate digrams. After symbol 8, a third `cd,ls` appears, and the existing non-terminal symbol $A$ replaces the third occurrence of `cd,ls`. This results in a new pair of repeating digrams, `rm, A`, shown in the next line of the table. *Sequitur* forms a new rule $B$, which replaces the two occurrences of `rm, A`. In Table 2.1, symbol 10 demonstrates the idea that longer rules are formed by the effect of the *rule utility* property, which ensures that every rule is used more than once. When `ls` is appended to rule $S$, the new digram $BA$ causes a new rule $C$, to be formed. However, forming this rule leaves only one appearance of rule $B$, violating the second constraint. For this reason, $B$ is removed from the grammar, and its right-hand side is substituted in the one place where it occurs. Removing $B$ means that rule $C$ now contains three symbols. This is the mechanism for forming long rules: form a short rule temporarily, and if subsequent symbols continue the match, allow a new rule to supersede the shorter one and delete the latter.

### 2.2.2 Hidden Markov Models

As defined by L. R. Rabiner [17], a Hidden Markov Model (HMM) is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols.

The behavior profile is expressed by the parameters of the HMM. These parameters are found applying the Baum-Welch re-estimation algorithm. This algorithm adjusts the model parameters (i.e. initial state distribution, state-transition probability distribution and observation symbol probability distribution) to maximize the probability of the observation sequence given the model.

An HMM is characterized by the following, see Figure 2.2.

$$A = \{a_{ij}\}, a_{ij} = Prob(q_j \text{ at } t + 1 | q_i \text{ at } t \} \text{ state transition probability distribution}$$

$$B = \{b_j(O_t)\}, b_j(O_t) = \text{observation probability distribution}$$

$$\pi = \{\pi_i\} = Prob(q_i \text{ at } t = 1 \} \text{ initial state distribution}$$

$$O = \{O_1, O_2, ..., O_T\} = \text{observation sequence (input sequence)}$$

$$T = \text{length of observation sequence}$$
$Q = \{q_1, q_2, ..., q_N\}$ hidden states in the model

$N = $ number of states

The elements $a_{ij}$ indicate the probability of going from state $i$ to state $j$. The elements $b_j(O_t)$ specify the probability of observing symbol $O$ in state $j$ at time $t$. We may relate this to a sequence of commands of a special user. As an example we will build a very simple HMM based on a profile resulted from five sessions of six commands each with only a possibility of four commands per symbol $C_1, C_2, C_3, C_4$. This is shown in Table 2.2.

<table>
<thead>
<tr>
<th>Session 1</th>
<th>Session 2</th>
<th>Session 3</th>
<th>Session 4</th>
<th>Session 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_4$</td>
<td>$C_3$</td>
<td>$C_4$</td>
<td>$C_4$</td>
<td>$C_3$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$C_1$</td>
<td>$C_1$</td>
<td>$C_3$</td>
<td>$C_4$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$C_4$</td>
<td>$C_3$</td>
<td>$C_1$</td>
<td>$C_3$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$C_2$</td>
<td>$C_2$</td>
<td>$C_4$</td>
<td>$C_1$</td>
</tr>
<tr>
<td>$C_4$</td>
<td>$C_3$</td>
<td>$C_4$</td>
<td>$C_3$</td>
<td>$C_5$</td>
</tr>
</tbody>
</table>

Table 2.2: Sessions used to build an HMM model

If we build a hidden Markov model of $N = 6$ states (one per command), the matrix $A = \{a_{ij}\}$ that represents the state transition probability of going from state $i$ to state $j$ is shown below. We can also see the observation probability distribution, where each row corresponds to each state of the model and the values are the command probabilities of being seen in that state.
Finally we must specify the initial state distribution $\pi$. Since the first command of a new session starts in state 1, the initial state distribution is:

$$\pi = \{\pi_i\} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

### 2.2.3 Non-Negative Matrix Factorization

Non-Negative Matrix Factorization (NMF) is a method developed by Lee and Seung [12] where in contrast to other methods such as principal component analysis (PCA) and vector quantization (VQ) it has non-negativity constraints. NMF is applied by Wang et al. to build normal behavior models in anomaly intrusion detection systems [26] where user behaviors are profiled by the frequency property.

In this section, the theory related to NMF will be briefly exposed. Given a database represented by a $n \times m$ matrix $V$, where each column is an $n$-dimensional non-negative local vector belonging to the original database ($m$ local vectors), we can obtain an approximation of the whole database ($V$) by:

$$V_{\mu} \approx (W H)_{\mu} = \sum_{a=1}^{r} W_{a \mu} H_a$$  \hspace{1cm} (2.1)

where the dimensions of the matrix $W$ and $H$ are $n \times r$ and $r \times m$, respectively, see Figure 2.3. Usually, $r$ is chosen so that $(n + m)r < nm$. These results in a compressed version of the original data matrix. Each column of matrix $W$ contains a basis vector while each column of $H$ contains encoding coefficients needed to approximate the corresponding column in $V$. The following iterative learning rules are used to find the linear decomposition:

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T W)_{\mu\mu}}$$  \hspace{1cm} (2.2)
The above unsupervised multiplicative learning rules are used iteratively to update \( W \) and \( H \). The initial values of \( W \) and \( H \) are fixed randomly. With these iterative updates, the quality of the approximation of Equation 2.1 improves monotonically with a guaranteed convergence to a locally optimal matrix factorization [12].

In order to use the NMF algorithm, in [26] the SEA dataset is partitioned into sessions and the frequency of individual elements (commands) in each session is counted. The NMF method uses the values of frequencies of commands seen in a session for a given user as entries for matrix \( V \), where \( V_{i\mu} \) is the number of times the \( i \)-th command appears in the \( \mu \)-th session.

By NMF, matrix \( V \) can be factorized into \( W \) and \( H \). Matrix \( W \) contains the extraction of underlying features of users, its columns of \( W \) represent the basis profiles and the columns of \( H \) are the encoding coefficients. A normalization vector \( C \) can be obtained so that

\[
CH = 1
\]

For testing purposes, a new matrix \( V \) is form using data from testing sessions and by using the previous matrix \( W \) a new matrix \( H \) is found with Equation 2.2. Let \( t' \) be a column vector from \( H \). If the features contained in any testing session deviate significantly from those of the normal behaviors learned in the training datasets, the factorization using the learned basis \( W \) will generate encoding coefficients \( t' \) that suffice

\[
|Ct' - 1| > \epsilon
\]

where \( \epsilon \) is the detection threshold. The above expression indicates that the tested session belongs to a masquerader.

As an example of a \( V \) matrix for a user who has \( m \) sessions and \( n \) possible unique commands per session we show Figure 2.3. Also the matrices \( H \) and \( W \) are shown.
2.2.4 Uniqueness

Uniqueness [19] is a method based on the fact that commands not previously seen in the training data may indicate a masquerade session. This method extracts global statistics, that is, the user profiles are based on the other user’s data. In this method, the fewer users that use a particular command the more indicative that a user that entered that command is a masquerader.

This method uses what is called popularity, which is an indicative of how many users use a particular command. A command has popularity \( i \) if only \( i \) users use that command. Almost half of the commands appearing in the training part of the dataset are unique with popularity one and it represents 3.0% of the data.

Uniqueness’s detection model is a session score, \( x_u \), given by:

\[
x_u = \frac{1}{n_u} \sum_{k=1}^{K} W_{uk} \left( 1 - \frac{U_k}{U} \right) n_{uk}
\]

where

- \( n_u \) is the length of the testing data sequence of user \( u \);
- \( n_{uk} \) is the number of times user \( u \) typed command \( k \) in the testing data;
- \( K \) is the total number of distinct commands;
- \( U \) is the total number of users;
$U_k$ is the number of users who have used command $k$ in the training data;

where the weights $W_{uk}$ are:

$$W_{uk} = \begin{cases} v_{uk}/v_k, & \text{if user u's training data contains command } k, \\ -1, & \text{otherwise,} \end{cases}$$

where $v_{uk} = N_{uk}/N_u$ and, $v_k = \sum_u v_{uk}$.

$N_u$ is the length of the training data sequence of user $u$;

$N_{uk}$ is the number of times user $u$ typed command $k$ in the training data.

From the above formulas it is easy to see that temporal ordering of commands in a given testing sessions is ignored. The fraction $(1 - U_k/U)$ is a uniqueness index: it is 0 if all users have used command $k$ before, it is 1 if none of the users has used it before. The weights $W_{uk}$ control whether the uniqueness index should be subtracted or added, depending on whether the command was seen before or not. Hence a user that uses commands similar to the ones used in the training data will tend to score high. The quantity $v_{uk}/v_k$ represents the command usage relative to other users. It reduces the score contribution of commands that other users often use and this user rarely.

This formula has the biggest absolute values for command $k$ when this command is not popular. It has a maximum positive value when command $k$ has little popularity and it is contained in the training data of the user $u$. It has a maximum negative value when command $k$ has little popularity and it is not contained in the training data of user $u$.

### 2.2.5 Customized Grammar

Latendresse [11] developed an intrusion detection system based on grammar extraction by the Sequitur algorithm created by Nevill-Manning and Witten [13]. By extracting hierarchical structures from a sequence of commands and generating a context-free grammar capable of building that sequence, he constructed user profiles based on the training data of SEA. Once he obtained the grammar (production rules) that represents the training data of each user, he computed the total frequency of its expansion for that user and also the frequency of that same expansion for the other users, that is the across frequency.
The production evaluation function used was:

\[ e(p) = l_p \frac{f_p}{f_p + \frac{F_p}{k}} \]

where

- \( p \) is a production of the session;
- \( l_p \) is the length of the expansion of production;
- \( f_p \) is the frequency of the expansion of the production;
- \( F_p \) is the across frequency of the expansion of the production;
- \( k \) is a tuning constant.

For a certain value of \( k \), the above formula has a maximum when \( l_p \) is long and that script has not been used by any other user. The formula has a minimum when the length of the production rule is 1 and the across frequency is high.

Finally, see Figure 2.4, let \( F \) be the set of productions found in a session \( s \), then the evaluation of \( s \) consists of the sum over all productions of \( s \), in symbols: \( \sum_{p \in F} e(p) \).

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_3 )</th>
<th>( p_4 )</th>
<th>( p_5 )</th>
<th>( p_6 )</th>
<th>( p_7 )</th>
<th>( p_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.5</td>
<td>2.3</td>
<td>8.5</td>
<td>4.3</td>
<td>2.3</td>
<td>5.6</td>
<td>11.6</td>
<td>4.3</td>
</tr>
</tbody>
</table>

\[ \text{SessionScore} = \sum_{p \in F} e(p) = 47.4 \]

Figure 2.4: Evaluation of a session with customized grammars.

This method just like Uniqueness makes use of global profiles where the normal behavior of a user is also based on data from the other users.
2.2.6 Single Commands

Another method shown by M. Latendresse [11] where only repetitive single commands were used was also evaluated. This is a simpler version than Customized Grammar, based only on the frequencies of the commands without global scripts and without taking into account their temporal ordering. Because the length of the productions is equal to 1, that is, single commands, then the evaluation function shown below is obtained.

\[ v(c) = \frac{f_c}{f_c + \frac{k}{F_c}} \]

where

- \( f_c \) is the frequency of the command \( c \);
- \( F_c \) is the across frequency of the command \( c \) among all other 49 users;
- \( k \) is a tuning constant.

The sum over all commands of a session gives the score of the session. Also it is a global profile method.

2.3 Summary

In this chapter, the basics about Intrusion Detection Systems was given, that is, the meaning of an intrusion in computer science, the classification of IDS’s and the introduction to Masquerade Detection, which will be the focus of this work. Some important concepts like AIDS, MIDS, training data, and normal behavior must be understood before going ahead. Before passing to the masquerade detection methods there is an introduction to the Sequitur algorithm. Sequitur must be well understood in order to comprehend the underlying concepts shown in this thesis and also to understand the proposed detection method. There is a brief description of five masquerade detection methods, namely a method based on Hidden Markov Models, Non-negative Matrix Factorization, Uniqueness, Customized Grammar and Single Commands. All these methods will be used for comparison purposes with the proposed method and therefore the philosophy behind every method should be understood.
Chapter 3

Proposed Masquerade Detection Method

In this chapter, we give the details about the proposed detection method, however, before going into such details, we will first study a dataset of UNIX sessions that has been widely used for testing the performance of the methods previously explained and it will also be used for testing the proposed one. At the end of the next section, we point out some disadvantages of this dataset, being these disadvantages the reason of the next chapter.

3.1 Schonlau Dataset

The Schonlau et. al (SEA)\(^1\) dataset consists of clean and contaminated data of 50 users. This collection was obtained using the `acct` auditing mechanism, it consist of 15000 UNIX commands without arguments for each of 70 users. Then, 50 users were randomly selected as intrusion targets, so that the remaining 20 users were used as masqueraders and their data was interspersed into the data of the other 50 users. In order to make data more tractable the user data is divided in blocks of 100 commands, each block is treated as a session.

\(^1\)SEA is available from http://www.schonlau.net.

The normal behavior profile of each user is obtained from the first 50 sessions, which are legitimate and not contaminated. This is the training data. The last 100 sessions is the testing data, since it may contain
masquerade information. A block is either totally contaminated or legitimate from the user. A graphical representation of the dataset is shown in Figure 3.1.

![Figure 3.1: Schonlau et. al. dataset](http://www.schonlau.net)

Masquerading data was inserted using this rule: if no masquerader is present, then a new masquerader appears in the next session with a probability of 1%. Otherwise, the same masquerader continues to be present in the next block with a probability of 80%. These probabilities are given trying to resemble the fact that once a masquerader gets the legitimate user account and logs in using that account, there is 80% of probability that the masquerader will do it again and appear in the next session. There are chances that no masquerader appears in the data of certain users.

There is a matrix\(^2\) that shows where the masquerade sessions are located in the testing data for each user. This matrix contains 100 rows and 50 columns. Each column corresponds to one of the 50 users. The further left column corresponds to user 1 and the further right column to user 50. Each row corresponds to a session of 100 commands, starting with session 51 and ending with session 150. The entries in the table are 0 or 1. 0 means that the corresponding session does not come from a masquerade. 1 means that it is a masquerade session.

The SEA dataset presents some problems when more realistic conditions are considered. The users who were labelled as masqueraders did not have knowledge about the behavior profiles of the users to be impersonated. Besides, they typed UNIX commands as they usually do in a normal working session. Thus, the data used as masquerade sessions is not intended to impersonate the users’ activity. Moreover, it is possible to find masquerade sessions in SEA with very simple and repetitive sequences that can be easily

\(^2\)Intrusion location matrix available from [http://www.schonlau.net](http://www.schonlau.net).
3.2 Hybrid Proposed Method

In the previous chapter the fundamentals of IDS’s and masquerade detection were given. Now we are interested in proposing a masquerade detector that would be robust against different types of masqueraders, not only the ones found in SEA dataset. The motivation behind this proposition is that in following chapters different types of masquerade sessions will be created, and the challenge is to detect the best such synthesized masquerade sessions. This new synthesized sessions can be seen as created by a special type of masqueraders whose main feature is that they have information about the behavior profile of legitimate users and use it to create synthetic sessions that bypass detection methods.

In the previous chapter we studied some masquerade detector methods. These methods work under different types of properties, that is, methods like Uniqueness, Single Commands and NMF do not take into account temporal ordering or sequential properties. They only work taking into account frequency properties. Methods like Customized Grammar and HMM do take into account sequential properties. Based on the hypothesis described in chapter one:

- Hypothesis 1. The use of a masquerade detection method based on hidden Markov models along with the use of a pre-processing mechanism where audit information is folded (compressed) provides a better tolerance to masquerade synthesized sessions.

we propose a local profile hybrid method that uses compression and hidden Markov models for masquerade detection. This method uses Sequitur to extract the grammar from the training sessions of a given user. The grammar extracted is used to create compressed versions of the training and testing user sessions. This is accomplished by substituting, where possible, the sequences of commands by non-terminals $n_k$. This process is referred to as session folding. The training compressed sessions are what we use as training data for the HMM model. Figure 3.2 shows an architecture of the detection mechanism.
3.2.1 Session Folding Block

The session folding block is basically composed of two parts, these parts are the grammar extraction and the compressor. In the grammar extraction phase Sequitur constructs a context-free grammar from the training sessions and extracts hierarchical structures that generate the sequence of training commands, see Figure 3.3.

![Diagram showing the grammar extraction steps.](image)

Figure 3.3: Shows the grammar extraction steps.

An example of the grammar for a single user obtained from the grammar extraction phase is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Length</th>
<th>Commands Sequence</th>
<th>Frequency</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>l=5</td>
<td>ls, rm, mkdir, cd, cp</td>
<td>f=10</td>
<td>P=1.666</td>
</tr>
<tr>
<td>R1</td>
<td>l=4</td>
<td>ps, kill, ps, top</td>
<td>f=7</td>
<td>P=0.933</td>
</tr>
<tr>
<td>R2</td>
<td>l=2</td>
<td>vi, gcc</td>
<td>f=5</td>
<td>P=0.333</td>
</tr>
</tbody>
</table>

Table 3.1: Example of grammar extraction

Each rule has associated a sequence length, the number of commands of that sequence, the frequency of that sequence in the training data and a priority. This priority \( P \) is computed taking into account the length of the production rule, the frequency of that production in the training data and also on the total amount of data in the training sessions. The priority of the grammar symbol (rule) is obtained as follows:

\[
P = \frac{L \times f}{N}
\]

21
where:

\( P \) is the priority of the grammar symbol;

\( l \) the size (length) of the grammar symbol;

\( N \) the total number of commands in the training data;

\( f \) is the frequency of the grammar symbol.

The compressor block works as shown in Figure 3.4. Sequences of commands are substituted by production rules extracted from the Sequitur algorithm. The way sessions are folded is based on the priority of the grammar symbol. Sequences with higher priority are preferred over those with a lower priority. The folded session may also contain single commands, since those may not be substituted for any production rule.

![Figure 3.4: Session folding architecture](image)

For testing purposes we fold the test session with the grammar obtained from the training. The degree of compression is a measure of how alike is the testing session with the training data, which in turn may indicate if the test session comes from the legitimate user or not. An example of how a test session looks like once it is folded is shown in Table 3.2.

As it can be seen in Table 3.2, the lines starting with an "R" mean that a sequence of commands has been substituted by a single rule. In lines where a number appears instead, no rule was substituted and the command (represented as a number) remains right in its place.
3.2.2 Hidden Markov Model Block

As depicted in Figure 3.2, the last stage of the detection mechanism is based on a HMM. The folded training sessions are used to train a hidden Markov model, which will model the normal behavior profile of any given user. For implementation purposes we built 50 hidden Markov models, one for each user of the SEA dataset. Once the testing session has been folded it is then evaluated by the trained HMM model to get the probability that the session was generated by the model.

In an HMM one or more starting and finishing states are specified. Possible transitions are made successively from a starting state to a finishing state, and the relevant transition probability and symbol output probability can be multiplied at each transition to calculate the overall likelihood of all the output symbols produced in the transition path up to that point. When all transitions are finished, the HMM
generates a symbol sequence according to the likelihood of a sequence being formed along each path. In other words, when a sequence is given, there is one or more transition paths that could have formed the sequence, each path have a specific likelihood that the sequence was formed by it. The sum of all the likelihoods obtained for all such transition paths is regarded as the likelihood that the sequence was generated by the HMM.

The method proposed here will be compared against the performance of the ones explained in the previous chapter. Part of the work explained here can be found in [16].

3.3 Summary

In this chapter we explained the proposed masquerade detection method, which is a hybrid method composed of two blocks, the session folding block and the HMM block. The session folding block performs the compression of the information and the HMM block performs the detection. At the beginning of this chapter there is an explanation of the Schonlau et. al. dataset used for testing the methods described in chapter 2 and the method proposed in this chapter. One of the key points that should be understood before going to the next chapter are the disadvantages of the SEA dataset, sec. 3.1, since it helps justify the reason of the next chapter.
Chapter 4

Synthesis of Masquerade Sessions

In order to have a more realistic dataset with which we can adequately evaluate the detection methods with more objective results and as part of the contribution of this work, in this chapter we present how the motivation of avoiding intrusion detection mechanisms takes the masquerader to create synthetic sessions that follow the same behavior profile of the legitimate user. The subject of creating or synthesizing sessions or attacks has been extensively mentioned in publications [21, 7, 24, 9] especially when talking about mimicry attacks. A mimicry attack is a type of attack where the malicious code is inserted into legitimate data (usually normal behavior sequences of system calls) in such a way that IDS’s do not detect the attack.

The insertion of attacks into normal sessions is not what will be done next. That could be further research, instead, artificial sessions that resemble the legitimate behavior are placed as masquerade sessions, making difficult to detectors raise alarms. Here we propose a scenario where the intruder has knowledge of the legitimate user behavior and uses it to build sessions with the victim’s profile, this way the intrusion will not be detected, since it has the same legitimate user’s customs.

As previously mentioned Original SEA dataset contains 150 sessions of 100 UNIX commands for each of 50 users, the first 50 sessions of each user are legitimate and thus constitute the training data, however in the last 100 sessions we may or may not have some sessions entered by a masquerader. The last 100 sessions of each user is the testing data. SEA has been very popular to test masquerader detectors, however masquerade sessions in SEA come from working sessions of other users who had no intention to act as intruders. This is, they did not masquerade their activities.
Following the philosophy of an intelligent intruder, two types of datasets are built, one based on single commands frequencies and other type based on scripts (repetitive sequences of commands). We synthesize one dataset of the first type and three datasets of the second type. We refer to any synthesized dataset as **SEA-I dataset** and this may be of any the forms shown in Table 4.1.

<table>
<thead>
<tr>
<th>Synthesized datasets</th>
<th>Based on Commands</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Based on Scripts</td>
<td>Priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frequency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Length</td>
</tr>
</tbody>
</table>

Table 4.1: Sea-I datasets.

We will explain how each of these datasets were created. First, we must understand that masquerade sessions of any particular user were built based on the training data of that user and no other. When SEA-I was built *based on commands*, the commands frequency statistics of a user were extracted, then a histogram of commands was built. The masquerade sessions were built following that same probability distribution. This way sessions for all users were created using the same command probability distribution of each of their training data. This idea can be clarified if we see the command frequency histogram of User 1 in Figure 4.1.

![Command Frequencies](image)

Figure 4.1: Histogram of user 1 training data.

When the datasets were built *based on scripts* we introduce a phase where grammar extraction takes place using **Sequitur**. Each of the grammar symbols extracted, including the set of terminals (single
commands) is assigned three values, the priority, the frequency and the length. When the synthesized dataset was build based on scripts frequency, grammar symbols (including single commands) were chosen giving preference to those that had a higher frequency. To choose a particular symbol, we generated random numbers according to an exponential distribution and the numbers obtained are mapped to pick a grammar symbol until a masquerade session of 100 commands is built. Every time a non-terminal was chosen, its equivalent in single commands was added to the masquerade session. The same procedure is followed for the two other types of masquerade sessions but instead of choosing based on frequency, we picked up the symbol based on priority and length.

The SEA-I datasets based on script frequencies has a large number of commands and scripts, this is because the symbols were chosen according to its frequency in the training data, however, the masquerade sessions based on script length and script priority are composed of a small number commands and scripts as it is shown in Table 4.2. Usually the symbols with higher priority are scripts, because the length is greater than 1 and also they may repeat more that once, consequently, we need less scripts to get the session of 100 commands. In the case of scripts created by length, we just need an average of 1.48 scripts to obtain the session of 100 commands.

<table>
<thead>
<tr>
<th>Script Dataset Property</th>
<th>Average of Commands and Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>Commands 26.39</td>
</tr>
<tr>
<td></td>
<td>Scripts 10.37</td>
</tr>
<tr>
<td>Priority</td>
<td>Commands 6.41</td>
</tr>
<tr>
<td></td>
<td>Scripts 8.37</td>
</tr>
<tr>
<td>Length</td>
<td>Commands 7.45</td>
</tr>
<tr>
<td></td>
<td>Scripts 1.48</td>
</tr>
</tbody>
</table>

Table 4.2: Average of commands and scripts

Once we created the masquerade sessions for all users, they were located in exactly the same positions as the original dataset. This builds **SEA-I dataset**. In fact only the masquerade sessions were modified from SEA. All the masquerade sessions of any particular user were created taking into account only the commands and scripts frequencies of that specific user.

Now we want to know the performance of our proposed method and the other five previously mentioned methods against SEA and SEA-I datasets.
4.1 Summary

In this chapter we describe how were created each of the four synthesized masquerade sessions. These sessions we created based on command frequency, script frequency, script length and script priority. All these datasets and also the original will be used for testing all the methods including the proposed one. The results are shown in the next chapter.
Chapter 5

Simulation and Results

In this chapter, we give the details about how to measure the performance of masquerade detection methods, that is, the description about the construction of the ROC curves. In the next section, the ROC curves of the five methods and the proposed one are presented along with an explanation of the behavior shown in each graph.

5.1 Measurement of Detection Methods Performance

The way for comparing the performance of the methods is accomplished by the usual Receiver Operating Characteristic (ROC) curves, which are parametric curves generated by varying a threshold from 0% to 100%, and computing the false alarm (false positive) and missing alarm (false negative) rate at each operating point. The false alarm rate is the rate at which the system falsely regards a legitimate user as a masquerader, while the missing alarm rate is the rate at which the system falsely regards a masquerader as a legitimate user. In general, there is a trade-off between the false alarm rate and missing alarm rate. The lower and further left a curve is, the better it is.

The methods here studied give results in two forms. Methods like the proposed one, the one based on hidden Markov models, Uniqueness and Customized Grammars give a score matrix of 50 columns by 100 sessions. Each score corresponds to each session. Methods like Command Frequencies and Non-negative Matrix Factorization give a masquerade matrix (like the one shown in Appendix A) for each different threshold, then multiple files are obtained, one per threshold.
To construct the ROC curve of the methods that give a matrix of scores, all scores are sorted, and a
number of possible values of thresholds are considered (basically each score). For each threshold a missing
alarm and a false alarm ratio is computed. For each threshold we have a point of the ROC curve. For
methods like Command Frequencies and Non-negative Matrix Factorization each file of results corresponds
to the results obtained for a particular threshold and have a particular missing alarm to false alarm ratio.
An iterative process with different thresholds must be run in order to obtain different points of the ROC
curve.

5.2 Receiver Operating Characteristics (ROC) Curves

As we can see in Figure 5.1 different curves have different performance in distinct regions. We are interested
in having a low false alarm rate and a low missing alarm rate. In Figure 5.1 we show the results of six
methods for SEA dataset. Single Commands seems to be the best of all six methods. This method is global
since it uses information of all other 49 users to detect masquerade sessions of a single user. The second
better method is NMF. It is the best of all the local profile methods for this dataset. Both best methods are
based on the frequency property.

![Figure 5.1: Performance of detection methods with SEA dataset.](image)

30
In Figure 5.2 we show the performance when SEA-I is built based on commands frequency. Here we can see that the best method is the hybrid one, it has the least false alarm rate for the same missing alarm rate than any other method. Also the most affected methods were NMF, Uniqueness and Command Frequencies, which are methods based on command frequencies. Methods based on sequential properties or grammar extraction seem to be more robust against masquerade synthetic sessions based on command frequency.

![Figure 5.2: Performance of detection methods with SEA-I dataset based on commands frequency.](image)

Figure 5.2: Performance of detection methods with SEA-I dataset based on **commands** frequency.

With the next experiments SEA-I is always based on repetitive command sequences (scripts). When these scripts are chosen based on script frequency, the curve shown in Figure 5.3 is built. As we can see the hybrid method is again the one that has the best performance of all, however all curves move to the right, indicating that the false alarm rate increases. This is due to the fact that the synthesized sessions resemble very well the legitimate ones and it is more difficult to detect the masquerade sessions. The second better method is the one based on HMM. This robustness of the markovian methods is because the number of commands in this synthesized dataset helped the method to detect masqueraders, given that these commands do not usually followed the transition statistics of the model.
Figure 5.3: Performance of detection methods with SEA-I dataset based on scripts frequency.

With the following two SEA-I datasets, see Figures 5.4 and 5.5, the hybrid method does not perform the best. In fact, for the best curve, low levels of false alarm rate have very high missing alarm rate. The curve that outperforms the others corresponds to the method based on HMM, and the best operating point could be 50% of missing alarms and 50% of false alarms for both Figures. With these last two figures, we may suppose that the synthesized sessions have been sufficiently well created, so that no method may detect them, these type of masquerade sessions are a combination of sequential and frequency properties, which makes them very hard to detect.
Figure 5.4: Performance of detection methods with SEA-I dataset based on scripts priority.

Figure 5.5: Performance of detection methods with SEA-I dataset based on scripts length.
5.3 Discussion

In this section we will discuss about the hypothesis described in section 1.4.

The first hypothesis is:

- Hypothesis 1. The use of a masquerade detection method based on hidden Markov models along with the use of a pre-processing mechanism where audit information is folded (compressed) provides a better tolerance to masquerade synthesized sessions.

Hypothesis 1 may be proved by Figures 5.2 and 5.3 where the proposed method turns out to be the best of all the methods shown in those figures. As a prove of the improvement in performance added by the session folding mechanism, we can see Figure 5.1 where we may see the performance of the HMM alone and the HMM with the pre-processing mechanism.

The second hypothesis is:

- Hypothesis 2. The creation of a synthetic masquerade session that resembles the best the behavior of certain user must take into account frequency properties and sequential properties at the same time.

Hypothesis 2 may be proved by Figures 5.4 and 5.5. The datasets used to plot the characteristics in these two figures take into account grammar symbols with large command expansion, which makes these datasets to resemble very well the normal behavior profile of the users, thus, making very difficult to methods detect the masquerade sessions.

5.4 Implementation Issues of Detection Methods

In this section, some issues in the implementation of the detection methods are presented. This issues have to be taking into account in order to obtain the characteristics shown in this chapter. All these detectors were implemented on a 2.0 GHz Intel Pentium 4 personal computer, 1 GB RAM and Linux Suse 10.0.

5.4.1 HMM Based Detector

We build a discrete model for each user, and the corresponding testing sessions were validated by each model. The observation probability distribution has to do with the different number of commands in the testing sessions, The number of states for the model can be 100 which equals the number of commands issued
in a given testing session. With these initial parameters the mean and variance for the observation vectors are calculated for each state. This model can be further trained by the use of the Baum-Welch algorithm.

For the implementation of the detector using Hidden Markov Models, we used the Hidden Markov Model Toolkit (HTK)\(^1\) which is a tool for manipulating hidden Markov models developed by the Cambridge University Engineering Department (CUED). We did not perform update.

### 5.4.2 NMF Based Detector

The implementation of this method followed the improvements shown by Mex-Perera et. al. [15] where the matrix \(V\) passes through a normalization phase that takes into account the fact that commands commonly typed by the legitimate user in the training phase are less relevant to identify masqueraders. This is another local profile method and no update was performed in our implementation.

### 5.4.3 Uniqueness

The thresholds used for plotting the ROC curve can be obtained from www.schonlau.net. We did not apply update for this method.

### 5.4.4 Customized Grammar

For the purpose of this paper we used the version of the experiment where no global scripts are used and the evaluation is based on the frequency of the commands of the user and across all users. As an additional variation we did not apply update.

### 5.5 Summary

In this chapter we presented a description about how to build the ROC (Receiver Operating Characteristics) curve in order to measure the performance of the detection method. We could also see the results and the ROC curves of the six methods (five for comparison purposes and the proposed one) for each of the five datasets used to measure the performance of the methods. There is an explanation of the curves behavior. After these curves, there is a section with the discussion of such results and finally some implementation issues of the methods used here.

\(^{1}\)HTK is available from [http://htk.eng.cam.ac.uk/](http://htk.eng.cam.ac.uk/).
Chapter 6

Conclusions and Further Research

The masquerade detection is a topic of huge importance when talking about Informatics Security. If a person with a malicious purpose obtains a computer account different than his/her, then with high likelihood will return. This makes the problem repetitive. We can add that an intelligent masquerader may build sessions that bypass detection methods as shown in this work. Many actual systems do not even have a masquerader detector in it. All before facts are enough circumstances in favor of masqueraders to make the subject important.

In this thesis we can observe the weaknesses of some detection methods. The first conclusion is that basically a masquerade detection method can be bypassed creating synthetic sessions based on the same property by which the method works, that is, if any method works using the command frequency, then synthetic sessions created using that same property will be very difficult to detect. If a method works based on sequential properties, then a synthesized session based on scripts would bypass detectors. This can be shown in Figure 5.2 where methods like Uniqueness, NMF and Single Commands are methods that work based on command frequencies and these methods are the most affected in their performance for this dataset.

The masquerade sessions of some datasets could be detected 5.1, 5.2 and 5.3 however, the masquerade sessions from datasets based on script length and script priority could not been detected appropriately. That is, those masquerade sessions resemble very well the normal behavior profile of the users, making it difficult to detectors to classify adequately. The second conclusion is that in order to best synthesize a session based on commands, large expansion of grammar symbols (obtained from training data) must be
used and consequently the priority of the symbols is high.

All methods have different characteristics, this performance is measured by the ROC curve. The proposed detection method is the best when SEA-I is based on command frequencies and scripts frequencies, however when SEA-I is based on scripts priority and scripts length, all methods are severely affected because it is very difficult to distinguish the masquerade sessions from the training sessions. Then the third conclusion is that a robust masquerade detector would be one that works using frequency properties, sequential properties at the same time.

6.1 Further Research

As we can see all these methods are Passive, off-line, which give a result after a predetermined or fixed number of events (session commands). Some efforts are being done to implement on-line, real-time detectors [10], however these detectors could be more robust if they also used biometrics like the keystroke timings [8] and all properties above explained (frequency, sequential and causal relationship). Also the time between commands entered could be used, given that this could distinguish between scripts or one-by-one typed commands, which helps identify legitimate sessions from masquerade sessions.

Since this was only a sensibility study, real attacks in a form of commands should also been taken into account as a future work.
Bibliography


