Declaration

I hereby declare that this thesis was composed by myself that the work contained herein is my own except where explicitly stated otherwise, and that this work has not been submitted for any other degree or professional qualification except as specified.

Fernando Godínez
Atizapán, México
May 24, 2005

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES
DE MONTERREY
CAMPUS MONTERREY
PROGRAMA DE GRADUADOS EN COMPUTACION,
INFORMACION Y COMUNICACIONES

DOCTOR OF PHILOSOPHY
ARTIFICIAL INTELLIGENCE

On an Efficient and Scalable Architecture for Mimicry Attacks Detection Using Probabilistic Methods

by

Fernando Godínez Delgado

Monterrey, N.L., November 19, 2004

Dedication

To Marcos Godínez and Ma de Jesus Villareal.
Your wisdom and love will stay with us always.

Publications

Some of the work on this thesis has been previously published, in [Godínez et al., 2004a, Godínez et al., 2004b, Godínez et al., 2005a, Godínez et al., 2005b].

Acknowledgements

Graduate school has turned out to be a long, difficult, but rewarding journey for me, and I would not have made it this far without the help of many, many people. It is very difficult to acknowledge all the people that helped by providing wisdom, advice or just by listening, but I will try my best not to leave anyone out.

First and foremost, I would like to thank my adviser, Raúl Monroy, who was the one that suggested that I should pursue a PhD in the first place. Despite the fact that this research topic was not his area of expertise (now it is), I could not have asked for a better adviser. Raúl has an excellent sense of strategy, ... and a whip behind the door for when procrastination gets the better out of his students. Seriously I would have not finished my research without all his advise and help.

I would also like to thank my other adviser, Dieter Hutter, who was always a helping hand in every respect. His expert advise was much useful to finish this dissertation. I would like to thank everyone at the DFKI in Saärbrücken, Germany for all their help and support.

I would also like to thank all the people at "The Centre for Intelligent Systems" at the ITESM, Monterrey. In no particular order I would like to thank Hugo Terashima, Carlos Mex, Juan Nolazco, Ramón Brena, Rogelio Soto, Francisco Cantú, and Manuel Valenzuela for all their teachings and the support they gave me during my stay at the centre. Special thanks go to Carlos Mex and Juan Nolazco who were part of my thesis committee and took the time to read my manuscript and send their comments on it.

Thanks go for all the people in ITESM Estado de México where I did most of my research. Ricardo Swain-Oropeza as dean of research who is always aware of the needs of the graduate students. All the people in the security research group who shared every seminar listening to my research and giving good advice and pointing out deficiencies. The people in the verlab who were really good friends all the time we shared lab space.

To my family who always encouraged me to keep going and lo learn from my mistakes. My parents were always there as an inspiration and support. My brother and my sisters shared all my frustrations and joys during all this experience. My cousins who were there to give a helping hand and to listen to all my complaints and also to my theories even if they understood nothing about the subject. To Ricardo for those long nights talking about our dreams

and where we wanted to be. And all my aunts and uncles who were always pending that I was doing alright. The Barrera–Godínez family that accepted me as a son not only while I was in Monterrey but ever since. My family is to large to mention by name everyone but you all know what role you played during this experience.

To my friends who were always there to provide their opinions and to let me see when I was not being productive. To Iván and his family, who shared a roof when I needed it the most and give me all his support. To Chava who was always aware of my research even though he is an architect. Roberto and Fernanda who were there for the good and the bad moments. To Gerardo, Victor(s), Eduardo, who always give their support and unconditional friendship. To Javier, Fernando and their families, who gave sound advice when I needed it the most. To Juan José Cornejo who always was available to give good or just listen.

To Milca who saw me get little to no sleep finishing this document and trying to get everything ready. She always stayed at my side just encouraging me. To Chocky who always believed in me and has seen me grow as an engineer and then as a scientist.

All tuition was supported in full by the "Centre for Intelligent Systems" of ITESM Monterrey. This research has been supported by research grants from CONACyT (33337-A), CONACyT-DLR (J200.324/2003) and ITESM (CCEM-0302-05).

Abstract

An intrusion detection system (IDS) aims at signalling an alarm for every activity that compromises a secure state of an IT system. It often amounts to detecting a known pattern of computer misuse, a deviation to ordinary, expected user behaviour, or a combination thereof. Regardless of which of these approaches is adopted, current Intrusion Detection Systems (IDSs) are easy to bypass.

This thesis addresses about the three most important limitations of existing IDSs: i) current IDSs are easily overwhelmed by the the amount of information they ought to analyse; ii) current IDSs are not sufficient to monitor dynamic environments where the monitored services are changed according to the needs of the organisation; and iii) current IDSs are easy to bypass using a *mimicry attack* (attacks that simulate normal sequences of system calls). These kinds of attacks simulate normal activity (eg traffic, interaction) by varying an attack signature in a way that does not affect the harmfulness of the attack.

Instead of creating a lightweight detection method capable of dealing with large volumes of information, at the probable cost of loss of accuracy, we focus on making intrusion detection more tractable, scalable and efficient (without compromising accuracy). We make intrusion detection more tractable by preprocessing the information. Whether it is a sequence of network packets or a sequence of system calls, the information an IDS analyses is often redundant in at least two respects: first, every entry in the sequence may contain spurious information; second, any sequence may contain redundant subsequences.

To make probabilistic intrusion detection more scalable, efficient and flexible, we propose a novel architecture that includes a service selection mechanism. Instead of analysing a single stream of data, the stream is partitioned in services, each of which is analysed by a very specialised sensor. New sensors can be added on demand; if a new service needs to be monitored another sensor is placed. To make mimicry attack intrusion detection more accurate (reduce false positives) we propose to divide attacks into smaller segments. For each segmentwe will create a detector that classifies the segment and all its variants. By combining these smaller detectors we hope to detect all variations of an attack.

By using rough sets we have identified key attributes to eliminate spurious information, without missing chief details. Using n-gram theory we have identified the most redundant subsequences within a sequence, substitution of these subsequences with a fresh tag results in a reduction of the

sequence length. To approach service selection, we suggest the use of hidden Markov models (HMMs), trained to detect a specific service described by a family of n-grams In this thesis, we introduce a method which is capable of successfully detecting a significant, interesting sub-class of mimicry attacks. The key behind our method's effectiveness lies on the use of a *word network* [Pereira and Riley, 1997, Young et al., 2002]. A word network conveniently decomposes a pattern matching problem into a chain of smaller, noise-tolerant pattern matchers, thereby making it more tractable and robust. A word network is realised as a finite state machine, where every state is an HMM.

In our experiments, our mechanism shows an accuracy of 93%. By contrast, the rate of false positive occurrence is only 3%. Our log reduction methods are among the best in reduction ratio and features a minimal loss of information. Ours is one of the first techniques to successfully detect a sub-class of mimicry attacks.

List of Figures

Contents

List of Tables

Chapter 1

Introduction

An intrusion detection system (IDS) is a computer security element that sends alerts when an intrusion is detected. An intrusion can be defined as any activity that compromises the secure state of an IT system. Based on the detection scheme, current IDSs can be divided in two categories, misuse IDSs (MIDSs) and anomaly IDSs (AIDSs).

Both kinds of IDSs have now been studied for more than 15 years and naturally the methods for intrusion detection have evolved. Nowadays the most effective techniques for detecting intrusions make a lot of use of probability and stochastic theory. These methods demand a lot of computational power and storage space to build suitable profiles. The main problems we have identified with current IDSs are:

- Undesirable number of false-positives and false-negatives.
- Vulnerability to complex attacks.
- High network loads: Computational complexity.
- Non scalability.
- Lack of robustness.
- Lack of a data source integration mechanism.
- Inefficient update of attack signatures.
- Lack of a terminology standard.

This thesis addresses the problems of: i) the vulnerability to complex attacks, namely mimicry attacks; ii) the computational complexity; and iii) the non scalability. By concentrating on these problems we aim at making probabilistic intrusion detection more tractable, scalable, efficient and flexible.

The contribution of this thesis is divided in two parts: i) we propose a novel architecture complemented with information reduction techniques specifically

designed to alleviate computational complexity for intrusion detection and to allow for higher scalability; and ii) we propose the use of speech recognition techniques to facilitate the detection of complex attacks, specially mimicry attacks. This contributions are the result of proving three hypothesis.

Hypothesis 1 By using pre-processing mechanisms we can improve the performance of current intrusion detection methods by operating on compact information. The mechanisms keep key information necessary for a proper intrusion detection.

To make probabilistic intrusion detection more tractable, we compact the information to be analysed without losing key information. This input information, either being a sequence of network commands or a sequence of system calls, is often redundant in at least two respects. First, any entry in either of these sequences, called *object* for short, may contain spurious information. Second, any sequence may contain a number of irrelevant objects. To remove these irrelevant objects we propose the use of a pre-processing mechanism consisting of two log file reducers. We reduced the number of attributes using rough set theory, and the length of sessions using n-gram language models.

The attribute reducer is used to filter out redundant attributes. Rough sets are an appropriate tool to discriminate redundant and spurious information. Compared with other techniques like GINI or ID3, rough sets are capable of treating incomplete information. Results from the rough set attribute reduction show that 66% of the attributes in log files were redundant and thus can be eliminated. A reduction in the number of attributes compacts the number of different elements to look for in intrusion detection.

Yet, performance of probabilistic methods, like for instance hidden Markov models (HMMs), are dependant on the length of the sequences to be classified. N-grams are used for session length reduction. The reduction is obtained by an analysis of the most frequent n-grams found in the sessions. The reason to select n-grams over other methods, like HMMs, is because it provides a similar cost-based search without non-determinism. Non-deterministic methods result in loss of information because similar sequences are treated as equal, therefore a reverse process to recover the original sequence is not possible. Moreover, a sequence containing an attack might be reduced with the consequential loss of the attack's evidence. Results using n-grams show that a 75% length reduction can be obtained using our method. Hence, together the two filtering methods yield a reduction of more than 85% in the log files.

Hypothesis 2 By using an architecture that allows one to plug-in specialised intrusion detectors, we can make intrusion detection scalable.

In order to have a higher degree of specialisation, resulting in more efficient, flexible and scalable IDSs, we have a selector which is capable of distinguishing the service a session belongs to using only the session header. The problem to find such a selector is narrowed down to characterising for each service a family of patterns. To characterise such families we have different options.

Neural networks are a universal classification system but have a fixed number of inputs and the header length is variable. Other methods commonly used to group similar sequences are HMMs, PCFGs, and dynamic programming. Some of these methods have been used in DNA and RNA sequence alignment [Sakakibara et al., 1994, Brown, 1999, Brown, 2000, Krogh et al., 1994, Hughey and Krogh, 1996, Mamitsuka, 1997]. PCFGs are computationally more expensive than HMMs and dynamic programming. Dynamic programming on the other hand relies on the number of differences between a selected sequence and the tested sequence while HMMs calculate the probability of a symbol in the sequence given the previous symbol. Experimentally HMMs are more precise in classifying sequences with variations above a threshold. Our observations show that variations in service headers are above this threshold, so our service selector is based on HMMs.

To improve scalability, our selector allows for new detection modules specialised in different services to be plugged in to the system. Efficiency is a result of dividing intrusion detection in a series of small specialised modules. Flexibility is achieved because the service discrimination is done by classifying a service initial behaviour, and not its configuration.

Hypothesis 3 We can detect mimicry attacks with the help of hidden Markov models and word networks.

Current IDSs have problems when dealing with complex attacks. In the case of a misuse IDS only a couple of modifications to an attack are necessary to bypass the IDS. Wagner and Soto [Wagner and Soto, 2002] have demonstrated that host based anomaly IDSs are incapable of detecting a special kind of attack called mimicry attack. Kendall [Kendall, 1998] has also demonstrated the ineffectiveness of MIDSs to detect mimicry attacks. Mimicry attacks have the same semantics as known attacks, however they are syntactically different. This difference is given by variations on the system calls of the original attack and insertion of system calls that have no effect on the harmful state of the attack (no-ops). The number of possible variations for a mimicry attack are infinite. Normalising system calls can not be done straight forward for all system calls. There are some system calls (A and B) that can be substituted by the same system call C, since A can not be substituted by B an attack signature might be lost if we normalise A and B with C. Also, filtering out no-ops is a difficult task since the no-op condition of a system call depends on the context it is found. Moreover, normalisation and filtering are not obvious.

1.1 Layout of this Dissertation

Chapter 2 shows the current state of the art in misuse and anomaly intrusion detection, as well as current trends in log file reduction. We also illustrate problems with current IDSs.

Chapter 3 describes trends and limitations of other architectures and describes how our architecture helps to overcome these limitations. We de-

scribe the way the architecture helps to solve the illustrated problems of current IDSs, namely: i) the high information loads; ii) the non scalability; and iii) the vulnerability to complex attacks (mimicry attacks).

Chapter 4 investigates the problem of attribute reduction. We compare other techniques used for attribute reduction against rough sets by using a reduction sample. We show why rough sets are good at dealing with problems, such as incomplete information, other techniques can not deal with. We include a description on the use of rough set theory for attribute reduction as well as a discussion on how other methods compare to ours.

Chapter 5 describes why the session length affects effective intrusion detection. We describe why we selected n-grams as our length reduction method by comparing its performance against other reduction techniques. We show the use n-grams models for session reduction as well as its impact on misuse detection. We also show a comparison of our method against other reduction methods used in intrusion detection.

Chapter 6 describes our service selection mechanism, it gives special attention as to why we chose HMM to classify services. It also explains the rationale for using a service selector based on a complex method and not just using the port number. We show classification results using normal sessions as well as reduced ones.

Chapter 7 illustrates the problem of detecting mimicry attacks. We begin by showing that the problem is not trivial and that filtering system calls can not be done directly. We explain why fragmenting attack signatures and using HMMs and word networks is a good method for detecting such attacks. Then we describe how our misuse IDS is built and how attacks are characterised and finish by presenting detection results using both normal and reduced sessions.

Chapter 8 summarises the results of the dissertation with our conclusions and proposed future work.

Appendix A describes the BSM log file format.

Appendix B describes the ARPA language model file format.

Appendix C contains a description of the HTK file format for HMMs and word networks

Chapter 3

A Novel Architecture for an IDS

3.1 Introduction

As described in chapter 2, we identified three deficiencies of current IDSs: i) they get easily overwhelmed for the amount of information they ought to analyse; ii) they lack scalability; iii) they are prone to mimicry attacks. This chapter presents a novel architecture (Fig. 3.1) that makes IDSs more scalable, tractable, and more tolerant with respect to mimicry attacks.

Usually an IDS is presented as the audit source, a sensor (data collector), an analyser (detection method), and a response unit. Many problems an IDS faces, like scalability, flexibility, or efficiency, are not related to the detection method but to the IDS architecture.

A log file is composed of a sequence of objects, each of which have a number of attributes. This input information, whether it being a sequence of network commands or a sequence of system calls, is often redundant in at least two respects. First, any entry in either of these sequences, called *object* for short, may contain spurious information. The same information can be extracted with a fewer number of attributes. And second, any sequence may contain a number of irrelevant objects. The sequences of objects in a log file are the result of both human interaction and automatic sequences generated by a computer. Within these sequences it is common to find many repetitive subsequences.

Based on these observations, we propose the use of information filters and reducers as first elements of our architecture. To make intrusion detection more tractable and allow for faster detection results, we compact the information to be analysed without losing key information. Using rough set theory (described in §4.5), we have successfully identified the object attributes that characterise best the object information without missing important details. Using n-gram theory §5.3), we have successfully identified those subsequences of objects that most frequently occur within a session and then folded them up with a fresh

tag, thus reducing the session length.

Current IDSs are tied to the service they are monitoring. If a new service needs to be monitored, then the whole IDS needs to be reconfigured. The same happens when the configuration of the monitored services changes. If an IDS is monitoring a service attached to some port and the port is changed, then the IDS needs to be reconfigured. To make an IDS scalable, we suggest to structure intrusion detection as a collection of sensors, each of which is specialised to a particular service. To approach service discrimination, we suggest to use hidden Markov models (HMMs) (described in §6.3) trained to pinpoint what n-gram family (characterising a service) the given header of an input sequence is likely to belong to. Service discrimination is the third module of our architecture.

Besides the deficiencies with current IDSs, we have found desirable characteristics: i) more accurate detection methods; ii) lower level of false alarms (in order to make an IDSs more reliable); iii) faster response time. To make an IDS more accurate, we propose the use of probabilistic pattern recognition methods. These methods need large amounts of information for proper training. They ought to analyse huge amounts of information to perform intrusion detection. For these methods to have a faster response time, we propose a modular architecture which allows to incorporate modules that: i) reduce the amount of information to analyse; ii) increase the scalability and flexibility of the IDS; and iii) increase the true positive rate and reduce the false positive rate of the IDS.

The last two modules of our architecture contain the actual intrusion detectors. The fourth module is a misuse intrusion detection module which is based on HMMs and Word Networks (see §7.3). For the fifth module we have proposed an anomaly detection scheme relying on probabilistic context-free grammars to describe normal user behaviour and detect behavioural deviations.

The remainder of this chapter is organised as follows: §3.2 is a description of the problems faced by current architectures; §3.5 describes the inputs and output for our architecture; §3.6 present other architectures and how they compare to ours; §3.3 describes the proposed architecture for an IDS; §3.4 outlines the chief characteristics of our architecture; finally in §3.7 we will present conclusions regarding our architecture.

3.2 Problem Description

As described in 2.8 we have identified limitations with current IDSs, in this chapter we will tackle the following: i) problems to deal with large data volumes; ii) computational complexity of the more effective detection methods; iii) lack of scalability; iv) lack of flexibility.

The large data volumes an IDS has to analyse get in the way of both, a real time detection and the use of more complex detection techniques, such as probabilistic sequence recognition methods. These probabilistic methods have a computational complexity directly related to the length of the sequence to be analysed, and to the size of the vocabulary for the sequences. Our hypothesis is that by reducing both the size of the vocabulary and the length of the sequences,

we can reduce training and parsing times of the probabilistic methods without an impact to intrusion detection.

On the problems of lack of flexibility and scalability, as the monitoring requirements evolve, it becomes difficult to add new components to a typical IDS. Our hypothesis is that by using a modular architecture (one module per service) it becomes feasible to monitor any number of services, as long as the system's memory and CPU allow it. It is also common to tie the IDS configuration to the system configuration. When the system configuration needs to be changed, i.e. change the port of a given service, the IDS must be reconfigured, this results in less flexibility. Our hypothesis is that by using a characterisation of the analysed data that is independent of system configuration, the detection process becomes more flexible.

3.3 An Architecture for an IDS

In this section we present a novel architecture for an IDS. The architecture is novel in three respects: i) it incorporates two pre-processing mechanisms, one for attribute filtering, and one for session folding; ii) it incorporates a service discriminator, to separate multiplexed input sequences to its individual services; iii) and it incorporates a hybrid detection scheme by including both, a misuse detector (MIDS) and anomaly detector (AIDS) [Debar et al., 1999]. The IDS simultaneously checks the input for misuse and anomaly. Fig. 3.1 shows the IDS architecture.

Now we will briefly describe the role of each module of our architecture.

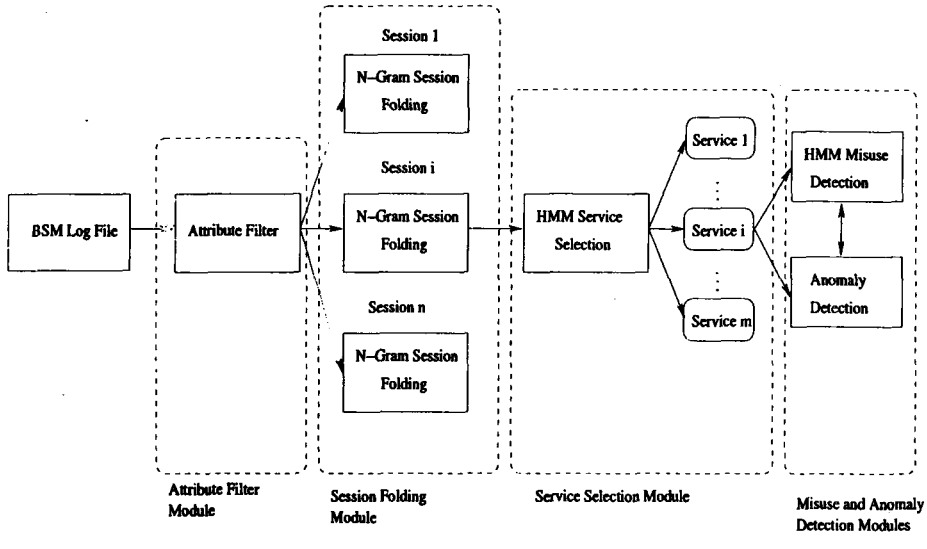


Figure 3.1: IDS Architecture

3.3.1 Attribute Filter Module

Audit log files can be seen as a two dimensional array of data. Rows stand for objects, in our case an object is a system call, and columns as the object's attributes. Object's attributes often provide repetitive and superfluous information. In order to remove redundant information we use an attribute filter based on rough set theory. The filter is explained in detail in chapter 4. We need only to notice that this filter takes a log file with all its attributes as input. After the filtering process, the modules outputs objects with a reduced number of attributes, usually about 64% of the original number of attributes. A BSM log file object can have up to 51 attributes. But usually an entry has half that number. We normalised this number of attributes to the most significant ones. The reduced output is passed to the session folding module.

3.3.2 Session Folding Module

The session folding module takes as input the objects delivered by the attribute filter module, and substitutes common subsequences to reduce session length. The rationale behind this procedure is that there exist repetitive subsequences among different services. Using an n-gram model analysis, as described in chapter 5, we identify key subsequences in the form of n-grams. Such n-grams are used to reduce other sessions. The obtained reduction factor is 4. Also the number of n-grams used in reduction is very small; we only used 19 out of 200 possibilities for reduction. This only adds a small number of new objects to the object vocabulary. The importance of a small object vocabulary is that HMMs training time depends on the size of the vocabulary as well as sequence length. Other methods like HMMs can be used to find repetitive subsequences, but with our method, all the reductions are deterministic and we can return to the original sequence. HMMs have an different role in our architecture, which is described below.

3.3.3 Service Selection Module

Folded sequences of system calls are the input for the service selection module and its output are the same sequences but separated by services. This module is a discriminator that uses hidden Markov models to calculate the probability that a given session belongs to a certain service. This selection reduces the search space for the next stage of the IDS. If we know that a session is a `telnet`, `smtp`, or `ftp` session we only need to test for specific attacks to detect misuse, or specific profiles to detect anomalies. The selection process is throughly described in chapter 6. With sessions separated by services, our misuse detection module only tests for attacks belonging to that service. Thus it specialises the intrusion detection by services.

3.3.4 Misuse Detection Module

Not only are hidden Markov models (HMMs) useful for service selection, they can also be useful to detect similarities between a session and known attacks. If the session is similar enough then an alarm is raised. The input of the misuse detection module is a service discriminated and folded sequence of system calls, and its output is the probability that such a sequence can be generated by a given HMM.

HMMs have been successfully used for intrusion detection [Warrender et al., 1999, Qiao et al., 2002, Yeung and Ding, 2003]. But as reported in all these papers, HMMs take a large amount of time for training. Our reduction methods can be helpful to improve the training times reported for both, the experiments by Warrender et al. and by Qiao et al.

Wagner and Soto [Wagner and Soto, 2002] have also studied the disadvantages of using only short sequences as a detection base using HMMs. As described in chapter 2, Wagner and Soto demonstrated the inability of current IDSs to detect mimicry attacks. The main advantage of our misuse detection method is that it is capable of detecting a variety of mimicry attacks. The misuse detection module will be described in detail in chapter 7.

3.3.5 Anomaly Detection Module

By folding a sequence of system calls, the creation of a reliable profile is more feasible. By reliable, we mean that it faithfully describes the profiled user. The module will process the input session and verify whether it is similar to some of the known profiles. If the session does not fit any of the profiles then an alarm is raised. This module can also be used to prevent false positive alarms in conjunction with the misuse detection module. If some behaviour matches a known attack but also a known profile it is probably normal behaviour. This will depend on some relation between the two modules.

By using the misuse detection module the rate of false negatives (unseen attacks) is reduced. Also by using the anomaly detection module unknown attacks are more easily detected. For the anomaly detection module we advocate the use of probabilistic context-free grammars. We use one such a grammar to describe service specific behaviour. A profile for each service is built from training data. The anomaly detection module is only part of our proposed architecture but was left as future work.

3.4 Architecture Characteristics

Our proposed architecture is well suited for environments with high information loads. Regardless the source, our architecture can be used to reduce information and alleviate the work load of an IDS. Even if the source are network packets, some spurious information can be identified and therefore discarded. It is also possible to discover repetitive patterns among data sequences and use a folding method as the one described in chapter 5.

Our reduction modules improve the performance of probabilistic pattern recognition methods like HMMs (already used in intrusion detection), or Probabilistic Context-Free Grammars (not used in intrusion detection). In both methods, order of training and parsing algorithms, is directly related to the length of the sequence to be analysed.

The output from the reduction modules is a sequence of reduced objects. The sequence is in the form of a number sequence. Any data source can be used, adding flexibility to the IDS. The service discriminator aims at selecting the particular IDS that will be used for the analysed sequence. This specialisation reduces the search space for the IDS making it more efficient. More services can be added to the IDS accordingly just by inserting a proper discriminator.

Using a hybrid architecture, and combining the output from both detection modules, the number of false alarms can be reduced, and the detection ratio increased. This is possible because of the feedback from the misuse detection module to the anomaly detection method. If a sequence is similar to a known attack but also to normal behaviour then the alarm ratio is lowered. If it is similar to an attack and not similar to normal behaviour the alarm is higher. If it is not similar to either an attack nor normal behaviour then it is regarded as an anomaly. In the next section we describe graphically how data is transformed in each module of our architecture.

3.5 Input and Desired Output

The input data for our architecture are BSM raw log files. These log files have a variable number of attributes for each object. They are normalised to a constant number of attributes i. Then the normalised logs are formated to a reduced number of attributes $j = i/3$. The normalised and reduced log files are separated into sessions and run through the session folding module and reduced sequences are the result. The number of objects in a sequence before folding is n, the number of objects after folding is m, notice that $n >> m$. From this point on all the process is run in parallel for each session. These folded sequences are separated by services. The service selector is like a multiplexer, each session is tested using the appropriate IDS (both misuse and anomaly). The final output is a set of probabilities indicating how similar is the sequence with the known attack base. The information flow in our architecture is shown in figure 3.2.

3.6 Alternative Architectures

Other architectures were analysed in order to understand their performance in terms of large data volume management, flexibility, and scalability. According to each architecture's description we made an hypothetical assumption on how the architecture will behave against configuration changes, new service monitoring, and large volumes of information. We only analysed host-based IDS architectures, there are other possible architectures such as network-based, or

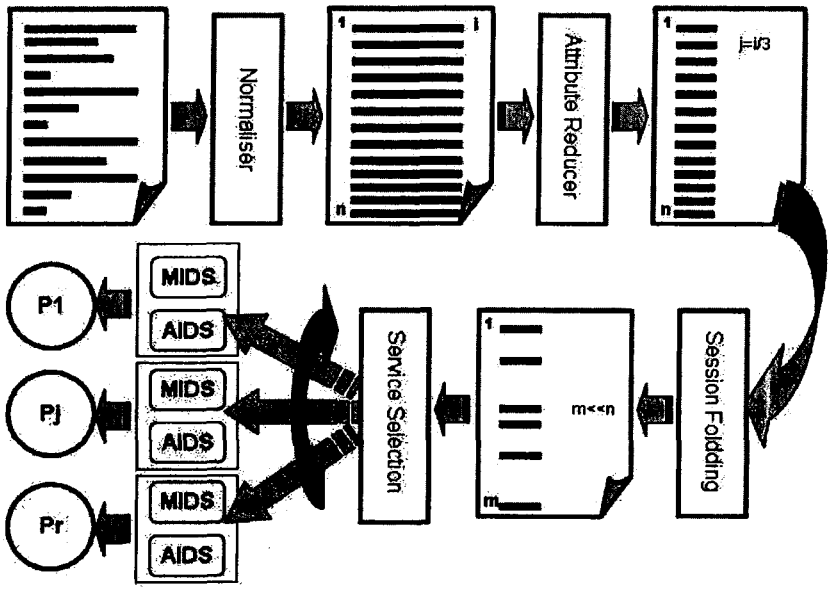


Figure 3.2: Architecture Cycle, input is a raw BSM log file, output is a probability for each monitored session.

distributed [Mell and McLarnon, 2000, Karlsson, 2001].

According to [Arvidson and Carlbark, 2003] most of current IDSs use a three level architecture. The first level is an information collector, the second level is an analyser, and the third level is the response unit. This typical architecture has as inputs raw data and as output a response event. In this sense the architecture is similar as the one proposed in this dissertation, we are based on the same three levels as well as similar inputs/outputs.

The collector processes the information and transforms it in a standard format for the IDS. This transformation mentions a possible (not necessary) reduction step, but reduction is not an explicit element of the architecture. The output information is fed to the analyser.

The analyser verifies if the data provided by the collector contains an attack. In case of a positive response the analyser triggers an event that is sent to the response unit. The analyser can be split into two elements: i) knowledge base; and ii) the classification engine. The knowledge base contains information related to the attacks it will detect. To determine if the data from the collector is an attack, the classification engine compares the data received from the collector with the one contained in the knowledge base.

The response unit decides the actions to follow depending on the results from the analyser. The unit has two complementary modules: i) policy rules; and ii) an event base. The policy rules determine the type of response based on the knowledge base from the analyser and a set of predetermined rules. This module

can filter out events that are known to be harmless. If this module is not present, the same standardised response is used for all the events and every event is to be considered. The architecture definition from [Arvidson and Carlbark, 2003] states that this module is optional. Finally, the event base records every event monitored by the IDS for a future analysis.

Even though the architecture considers information pre-processing, reduction is only mentioned and treated in a general manner. Reduction steps are never defined (the order or the kind of reduction). Pre-processing is presented as a black box where input data is transformed to fit the analyser regardless the methods used in the pre-processing. Even though an architecture definition only has to contemplate a general description independent of the implementation, it is necessary to at least describe the order of the steps to follow. It is very different to reduce the number of attributes first and then reduce the length of the object sequences. If the length reduction is performed before the attribute reduction, then the reduction rate might be smaller, because the larger the number of attributes, the larger the number of different objects in the sequence. The more variations between objects in a sequence makes it harder to find repetitive patterns to be used in a reduction the reduction. By contrast, our architecture has well defined pre-processing steps. This order is best to achieve greater data reduction.

Arvidson and Carlbark's architecture does not consider a mechanism to adapt to configuration changes in the monitored system, or the addition of new monitored services. The architecture proposes a knowledge base and a classification engine, this combination leads to extra work if different services want to be monitored. According to the architecture the same knowledge base is used for every monitored service. By contrast our architecture is designed as a series of different sensors each specialised in a given service. If a new service is to be monitored, then a new sensor can be added to the architecture. The problem of the configuration flexibility is approached by separating services according to their behaviour at system call level and not their configuration.

This architecture does not consider a hybrid detection model. This results in less efficiency (if the detector is anomaly based) or flexibility (if the detector is misuse based). In our architecture a hybrid detection model is proposed, but the module has not been developed yet.

3.7 Conclusions

The architecture presented in this chapter incorporates different modules that solve three of the identified problems of current IDSs. By incorporating a pre-processing mechanism we reduce the work load for the detection methods. With the use of a service discriminator, the search space for detection methods is greatly reduced by means of specialisation. The service discriminator adds scalability, because new services can be incorporated to the IDS. And by incorporating a hybrid architecture the detection ratio can be increased.

In chapter 6 we will describe the key element of our architecture the scala-

bility, flexibility and efficiency of the IDS.

All the theory used during the remainder of the thesis is described in the next chapter. If the reader is familiar with rough set theory, n-grams, hidden Markov models and word networks it is safe to skip the chapter and start reading in chapter 4.

Chapter 4

Attribute Reduction

4.1 Introduction

With growing amount of information flow in an IT system there is a need for an effective method capable of identify key elements in the data in order to reduce the amount of memory and time used in the detection process.

As we described in chapter 3, our architecture incorporates a pre-processing mechanism consisting of two modules. One of the modules is an attribute filter and the other is a sequence folding module, this chapter describes the former method.

A log file can be arranged as a two dimensional array. BSM log files have a variable number of attributes for each object, so before considering BSM log files as two dimensional arrays they have to be normalised with a standard number of attributes. After normalisation, the dimensionality of the array is calculated by multiplying the number of columns (attributes) by the number of rows (objects). To facilitate intrusion detection, the dimensionality of the log files must be reduced. This is called the dimensionality reduction problem. The dimensionality can be reduced in two ways: i) by reducing the number of attributes; and/or ii) by reducing the number of objects.

This chapter addresses the problem of dimensionality reduction using an attribute relevance analyser. Our experience shows that, in order to differentiate one object from another many attributes provide redundant information. We aim to filter out redundant, spurious information, and significantly reduce the amount of computer resources, both memory and CPU time, required to detect an attack.

Using rough sets, we have been able to successfully identify pieces of information that succinctly characterise computer activity without missing important details. We have tested our approach using various BSM log files of the DARPA repository. More precisely we used 8 days out of 25 available from 1998 to extract the reducts. We then used other 8 days from the 25 available from 1998 and 6 days out of 15 form 1999 as a test set. The results we obtained show that

we need less than a third part of the 51 identified attributes to represent the log files without significant loss of information.

This chapter is organised as follows: §4.2 describes the problem of attribute reduction. In §4.3 we show with brief examples the input for the attribute reduction module and desired output. §4.5 gives an introduction to the rough set theory used in this chapter. §4.4 is a brief explanation of why we selected rough set theory over other methods. In §4.6 we describe the methodology used in our experiments. §4.7 describes the experimental results. In §4.8 we contrast our method with existing attribute reduction methods. Finally some conclusions drawn from this experiments are discussed in §4.9.

4.2 Problem Description

A log file is a file containing sequences of events from some source. The source can be system calls, network traffic, information from a router, etc. Each of this events, which we call objects, has a number of values attached to it, which we call attributes. What we propose is to take this input, transform it into a two dimensional array with an heterogeneous number of attributes, and then take out all the unnecessary attributes from each object. This will leave a log with a reduced set of attributes for each object.

The set of attributes must be such that we keep at least 90% of information discernibility between objects. That is, if before reduction 100 objects were distinguishable, after reduction at least 90 objects must still be distinguishable. With less than 90% of information discernibility the effective detection rate diminishes as well as the false positive rate. We use Bayes theorem to prove this claim. D is the detection rate, FP is the false positive rate, and Di is discernibility between objects. $P(D|Di)$ is the conditional probability that the IDS detects an attack if it can discern between objects, $P(\overline{D}|Di)$ is the complement. $P(D|\overline{Di})$ is the conditional probability that the IDS will detect an attack if it can not discern between objects, $P(\overline{D}|\overline{Di})$ is the complement. $P(FP|Di)$ is the conditional probability of a false positive given full discernibility. Suppose a nearly perfect IDS with $P(D|Di) = 0.999$ and $P(D|\overline{Di}) = 0.01$. Using Bayes theorem we calculate the real detection rate for our IDS or $P(D)$ with:

$$P(D) = P(|Di)P(Di) + P(D|\overline{Di})P(\overline{Di})$$

here the detection rate without discernibility is negligible, so the IDS effectiveness is mostly affected by the discernibility. In the case of our nearly perfect IDS, a discernibility lower than 90% decrements the effectiveness below 90%. Supose the same nearly perfect IDS with $P(FP|Di) = 0.05$ and $P(FP|\overline{Di}) = 0.99$. With Bayes theorem we can calculate $P(FP)$ as:

$$P(FP) = P(FP|Di)P(Di) + P(FP|\overline{Di})P(\overline{Di})$$

with a discernibility of 90% we have an effective false positive rate of 18.9%, with a discernibility of 85% the false positive rate increases to 24%.

At the same time the reduction must keep the minimal number of attributes. The method used for reduction should also be capable of dealing with missing values for some attributes in each object. Depending on the source objects in a log file might have varying number of attributes. Most pattern recognition methods like GINI, ID3, C5 or k-means rely on an homogeneous number of attributes. And they usually need values for every attribute they use for the classification. If a dummy value is used to fill the missing attributes, the techniques above mentioned will generate imprecise decision trees. So the first condition for our log file is to have the same number of attributes for all its objects. This homogeneous system is also called an information system.

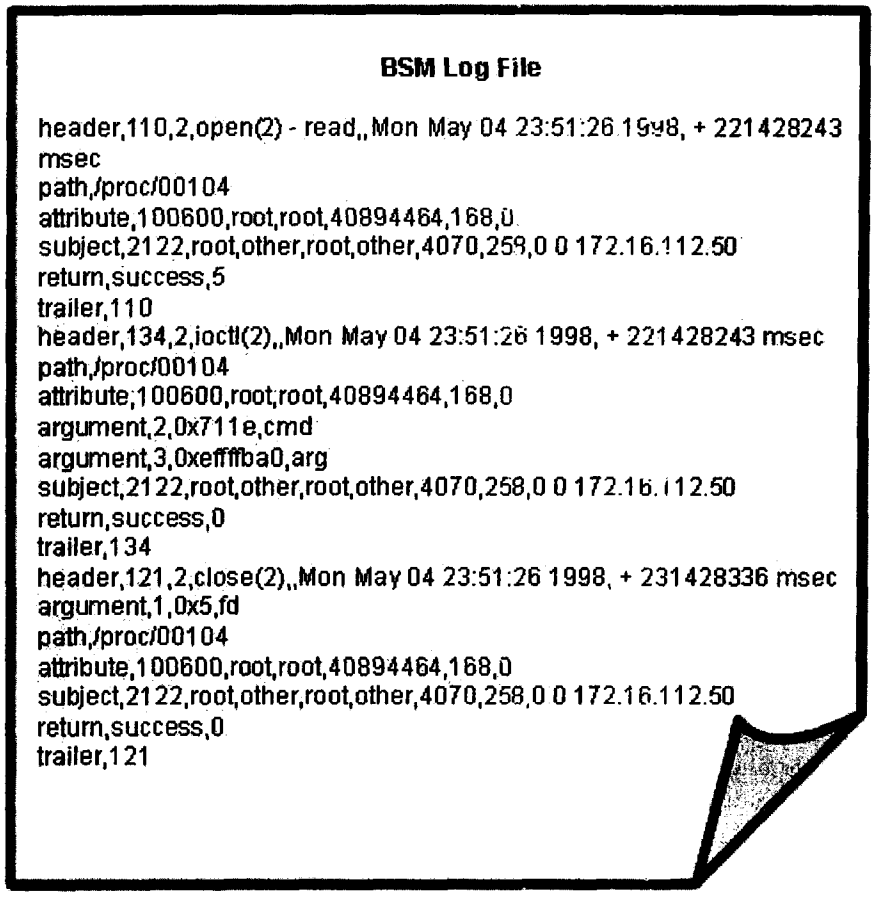
Even with an information system, intrusion detection still poses a big problem with respect to the amount of information that needs to be analysed. We believe that by reducing the number of attributes in an information system, the data load for the IDS should also be reduced.

4.3 Desired Input/Output

The attribute reduction module has as its input raw BSM log files in ASCII format as shown in figure 4.1. This kind of input is not suitable for a detection method that relies on a constant number of attributes. To find the most relevant attributes the first step is to transform the raw BSM log file into a two dimensional array (which in rough set theory is called an information system). This information system is then used to extract a reduced set of attributes. In figures 4.2, 4.3, and 4.4 we can appreciate the extent of a log entry for 6 objects and 51 attributes. The columns in bold text indicate the name of the attributes. In figure 4.5 we can appreciate the same 6 objects but with only 18 attributes. We can immediately appreciate the reduction advantages of using less attributes. For example attributes **Remote_IP** and **Machine_ID** have the same values in all the examples. In reality **Remote_IP** is present when a network connection is in place and **Machine_ID** is present when a process is being executed. However **Machine_ID** only has useful values when **Remote_IP** is present, so **Machine_ID** can be discarded. It could have been **Remote_IP** the discarded attribute but the algorithm keeps the first it founds.

4.4 The Use of Rough Sets for Attribute Reduction

In production environments, output data are often vague, incomplete, inconsistent and of a great variety, getting in the way of its sound analysis. Data imperfections rule out the possibility of using conventional data mining techniques, such as ID3, C5 or GINI. Fortunately, the theory of rough sets [Komorowski et al., 1998] has been specially designed to handle these kinds of scenarios. Same as in fuzzy logic, in rough sets every object of interest is associated with a piece of knowledge indicating relative membership. This knowl-



BSM Log File

header,110,2,open(2) - read,,Mon May 04 23:51:26 1998, + 221428243 msec
path,/proc/00104
attribute,100600,root,root,40894464,168,0
subject,2122,root,other,root,other,4070,259,0 0 172.16.112.50
return,success,5
trailer,110
header,134,2,ioctl(2),,Mon May 04 23:51:26 1998, + 221428243 msec
path,/proc/00104
attribute,100600,root,root,40894464,168,0
argument,2,0x711e,cmd
argument,3,0xeffffba0,arg
subject,2122,root,other,root,other,4070,258,0 0 172.16.112.50
return,success,0
trailer,134
header,121,2,close(2),,Mon May 04 23:51:26 1998, + 231428336 msec
argument,1,0x5,fd
path,/proc/00104
attribute,100600,root,root,40894464,168,0
subject,2122,root,other,root,other,4070,258,0 0 172.16.112.50
return,success,0
trailer,121

Figure 4.1: BSM log file in its raw ASCII format

Access_Mode	Owner	Owner_Group	File_Sys_ID	inode_ID	device_ID	arg_num-1
Integer	String	String	Integer	Integer	Integer	Integer
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	1
20000	root	root	8388608	51464	10747906	2
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	1
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	1
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	1
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	1

arg_val-1	arg_string-1	arg_num-2	arg_val-2	arg_string-2	arg_num-3	arg_val-3
Integer	String	Integer	Integer	String	Integer	Integer
3	fd	4	4026522828	pri	Undefined	Undefined
21256	cmd	3	4026391336	arg	2	1344063756
2051	uid	Undefined	Undefined	Undefined	Undefined	Undefined
2051	gid	Undefined	Undefined	Undefined	Undefined	Undefined
9	cmd	Undefined	Undefined	Undefined	Undefined	Undefined
2051	gid	Undefined	Undefined	Undefined	Undefined	Undefined

arg_string-3	arg_num-4	arg_val-4	arg_string-4	arg_num-5	arg_val-5	arg_string-5
String	Integer	Integer	String	Integer	Integer	String
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
strioctl:vnode	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

Figure 4.2: A BSM log file segment as an information system (attributes 1 to 21

arg_num-6	arg_val-6	arg_string-6	arg_num-7	arg_val-7	arg_string-7	exec_arg-1
Integer	Integer	String	Integer	Integer	String	String
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

exec_arg-2	exec_arg-3	exec_arg-4	exec_arg-5	exec_arg-6	exec_arg-7	exec_arg-8
String	String	String	String	String	String	String
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined

Socket_Type	Local_Port	Local_IP	Remote_Port	Remote_IP	Service	Audit_ID
String	Integer	String	Integer	String	String	String
Undefined	Undefined	Undefined	Undefined	0.0.0.0	Undefined	-2
Undefined	Undefined	Undefined	Undefined	0.0.0.0	Undefined	-2
Undefined	Undefined	Undefined	Undefined	172.16.112.50	Undefined	2051
Undefined	Undefined	Undefined	Undefined	172.16.112.50	Undefined	2051
Undefined	Undefined	Undefined	Undefined	172.16.112.50	Undefined	2051
Undefined	Undefined	Undefined	Undefined	172.16.112.50	Undefined	2051

Figure 4.3: A BSM log file segment as an information system (attributes 21 to 42)

Effective User_ID	Effective Group_ID	Real User_ID	Real Group_ID	Process_ID	Session_ID	Device_ID
String	String	String	String	Integer	Integer	Integer
root	root	root	root	96	0	0
root	root	root	root	96	0	0
2051	rjm	2051	rjm	264	257	0
2051	rjm	2051	rjm	264	257	0
2051	rjm	2051	rjm	264	257	0
2051	rjm	2051	rjm	264	257	0

Machine_ID	Descriptor
String	String
0.0.0.0	getmsg(2)|Undefined|success:0
0.0.0.0	ioctl(2)|/devices/pseudo/clone@0:udp|success:0
172.16.112.50	old setuid(2)|Undefined|success:0
172.16.112.50	seteuid(2)|Undefined|success:0
172.16.112.50	sysinfo(2)|Undefined|success:1
172.16.112.50	seteuid(2)|Undefined|success:0

Figure 4.4: A BSM log file segment as an information system (attributes 43 to 51)

Access_Mode	Owner	Owner_Group	File_Sys_ID	inode_ID	device_ID	arg_val-1
Integer	String	String	Integer	Integer	Integer	Integer
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	3
20000	root	root	8388608	51464	10747906	21256
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	2051
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	2051
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	9
Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	2051

arg_string-1	arg_val-2	arg_val-3	exec_arg-1	Socket_Type	Remote_IP	Audit_ID
String	Integer	Integer	String	String	String	String
fd	4026522828	Undefined	Undefined	Undefined	0.0.0.0	-2
cmd	4026391336	1344063756	Undefined	Undefined	0.0.0.0	-2
uid	Undefined	Undefined	Undefined	Undefined	172.16.112.50	2051
gid	Undefined	Undefined	Undefined	Undefined	172.16.112.50	2051
cmd	Undefined	Undefined	Undefined	Undefined	172.16.112.50	2051
gid	Undefined	Undefined	Undefined	Undefined	172.16.112.50	2051

Effective Usr_ID	Effective Group_ID	Process_ID	Descriptor
String	String	Integer	String
root	root	96	getmsg(2)|Undefined|success:0
root	root	96	ioctl(2)|/devices/pseudo/clone@0:udp|success:0
2051	rjm	264	old setuid(2)|Undefined|success:0
2051	rjm	264	seteuid(2)|Undefined|success:0
2051	rjm	264	sysinfo(2)|Undefined|success:1
2051	rjm	264	seteuid(2)|Undefined|success:0

Figure 4.5: An information system with a reduced number of attributes

edge is used to drive data classification and is the key issue of any reasoning, learning, and decision making [Félix and Ushio, 2002].

Other methods like principal component analysis and independent component analysis can be used to extract the most significant attributes from an information system. These methods rely on a co-variance matrix to calculate attribute relations. For the matrix to be effective, a numeric representation of the data is needed. Log files like BSM usually have a large number of attributes with alphanumeric values. A numeric representation of this values might lead to an incorrect selection of relevant attributes. By contrast rough set theory deals naturally with alphanumeric values, since it does not use the numeric values of the attributes to extract the reduct.

Another approach used to extract distinctive features from an information system is support vector machines. By executing many iterations the methods removes one useless feature. By contrast rough sets avoids many iterations and obtains a similar results in less time as demonstrated by [Zhang et al., 2004].

4.5 Introduction to Rough Set Theory

Knowledge, acquired from human or machine experience, is represented as a set of examples describing attributes of either of two types, condition and decision. ***Condition attributes*** and ***decision attributes*** respectively represent a priori and a posteriori knowledge. Thus, learning in rough sets is supervised.

Rough sets removes superfluous information by examining attribute dependencies. It deals with inconsistencies, uncertainty and incompleteness by imposing an upper and an lower approximation to set membership. Rough sets estimates the relevance of an attribute by using attribute dependencies regarding a given decision class. It achieves attribute set covering by imposing a discernibility relation. Rough set's output, purged and consistent data, can be used to define decision rules. A brief introduction to rough set theory, mostly based on [Komorowski et al., 1998], follows.

4.5.1 Rough Sets

Knowledge is represented by means of a table, so-called an ***information system***, where rows and columns respectively denote objects and attributes. An information system, $\mathcal{A}$, is given as a pair $\mathcal{A} = (U, A)$, where U is a non-empty finite set of objects, the ***universe***, and A is a non-empty finite set of ***attributes***.

A ***decision system*** is an information system that involves at least (and usually) one decision attribute. It is given by $\mathcal{A} = (U, A \cup \{d\})$, where $d \notin A$ is the decision attribute. Decision attributes are often two-valued. Then the input set of examples is split into two disjoint subsets: positive and negative. An element is in positive if it belongs to the associated decision class and is in negative otherwise. Multi-valued decision attributes give rise to pairwise, multiple decision classes.

A decision system expresses our knowledge about a model. It may be unnecessarily redundant. This is either because indiscernible objects may appear several times, or because some object attributes may be superfluous. To remove redundancies, rough sets define an equivalence relation up to indiscernibility.

Let $\mathcal{A} = (U, A)$ be an information system. Then, every $B \subseteq A$ yields an equivalence relation up to indiscernibility, $IND_{\mathcal{A}}(B) \subseteq (U \times U)$, given by:

$$IND_{\mathcal{A}}(B) = \{(x, x') : \forall a \in B.\, a(x) = a(x')\}$$

Where $x|\emptyset$ means the set of all x that satisfy $\emptyset$. *Reducts* of $\mathcal{A}$ are the least $B \subseteq A$ that is equivalent to A up to indiscernibility in symbols, $IND_{\mathcal{A}}(B) = IND_{\mathcal{A}}(A)$. Then the attributes in $A - B$ are considered expendable. An information system typically has many subsets B. The set of all reducts in $\mathcal{A}$ is denoted $RED(\mathcal{A})$.

An equivalence splits the universe, allowing us to create new classes, also called *concepts*. A concept that cannot be completely characterised gives rise to a rough set. A rough set is used to hold elements for which it cannot be definitely said whether or not they belong to a given concept. Rough sets are thus characterised by a boundary region. In symbols, let $\mathcal{A} = (U, A)$ be an information system. Then take $B \subseteq A$ and $X \subseteq U$. X can be approximated using the information in B by building the B-lower and B-upper approximations of X, respectively denoted by $\underline{B}X$ and $\overline{B}X$:

$$\begin{aligned} \underline{B}X &= \{x : [x]_B \subseteq X\} \\ \overline{B}X &= \{x : [x]_B \cap X \neq \emptyset\} \end{aligned}$$

where $[x]_B$ denotes the classes induced by the B-equivalence. So, objects in $\underline{B}X$ are members of X, while objects in $\overline{B}X$ might be. Then, the *B-boundary region of X*, which contains objects that cannot be definitely marked as members of X, is defined as follows:

$$BN_B(X) = \overline{B}X - \underline{B}X$$

A set is said to be *rough* if the boundary region is non-empty.

Potential rough sets can be automatically determined using a discernibility matrix. Let $\mathcal{A}$ be an information system describing N objects. The ***discernibility matrix*** (M_A) of $\mathcal{A}$ is a symmetric, $N \times N$ matrix, in which each entry, c_{ij}, captures the set of attributes upon which objects x_i and x_j differ:

$$M_A(i, j) = \{a \in A \;:\; a(x_i) \neq a(x_j)\} \text{ for } i, j = 1, \ldots, n$$

Adding the classes yielded by rough set analysis to the original attribute set, the reduction process can be applied a step further. The whole process is repeated until it no longer yields a reduction. The reduct extraction is extended with the concept of *dynamic reducts* presented below.

Dynamic Reducts

The main purpose of a *Reduct* is to find attribute subsets that hold the general patterns in the information system. Since it only takes a single noisy object in

$\mathcal{A}$ to change $IND_{\mathcal{A}}$, reducts as defined above are subject to incorporate noise in the data set. One way to avoid this problem is to extract *dynamic reducts* from $\mathcal{A}$.

The process of extracting dynamic reducts is a combination of normal reduct computation with re-sampling techniques. First we randomly take a set of subsystems $S = \{\mathcal{A}_1 \ldots \mathcal{A}_n\}$ from $\mathcal{A}$. Each subsystem $\mathcal{A}_i = (U_i, A)$. For each subsystem $\mathcal{A}_i \in S$ extract $RED(\mathcal{A}_i)$. Then from all reducts extracted calculate the number of occurrences in $\mathcal{A}_i \ldots \mathcal{A}_n$.

The reducts with a larger frequency are believed to be stable and hold more general information in $\mathcal{A}$ than $RED(\mathcal{A}_i)$. This set of reducts are also called the *generalised dynamic reducts* or $DRED(\mathcal{A}, \varepsilon, S)$. The ε parameter defines the minimal number of occurrences of a reduct to be included in $DRED(\mathcal{A}, \varepsilon, S)$. This relation is defined in the following equation:

$$DRED(\mathcal{A}, \varepsilon, S) = B \subseteq A$$
$$\text{where } \mathcal{A}_i \in S, \text{ and}$$
$$\frac{|RED(\mathcal{A}_i)|}{|S|} \geq 1 - \varepsilon$$

The subsets of objects $U_i \subseteq U$ resulting in the extraction of dynamic reducts is the same as removing from the discernibility matrix the rows and columns corresponding to objects $x \notin U_i$. We can generalise this by eliminating individual entries in the matrix instead of entire rows and columns.

By using dynamic reducts we can find a minimum common reduct set which will give us the most stable reducts with minimal loss of information. We extended this concept by creating a single reduct which combines the attributes with higher frequency from all the reducts in the dynamic reduct set. Experimental results of this approach will be seen in section 4.6 but we can say that the number of attributes in the combined reduct is almost equal to the number of attributes in the largest reduct in the resulting set.

4.5.2 Rosetta

The Rosetta system is a toolkit developed by Alexander Øhrn and Komorowski [Øhrn and Komorowski, 1997] used for data analysis using rough sets theory. The Rosetta toolkit is composed of a computational kernel and a GUI. The main interest for us is the kernel which is a general C++ class library implementing various methods of the rough set framework. The library provides basic structures for data storage (decision tables, reducts, rules) and algorithms to transform the data (reduct calculation, rule generation, classification) and a series of data import/export methods. The library is open source thus being modifiable by the user. This is convenient since it has limitations, like the incapability of importing more than one information system. For reduct validation purposes it is necessary to import at least two different information systems, one to generate association rules and the other to test the generated rules. Anyone interested in the library should refer to [Øhrn and Komorowski, 1997].

Access-Mode	device-ID	exec-arg-1	Effective-Group-ID
Owner	arg-value-1	Socket-Type	Process-ID
Owner-Group	arg-string-1	Remote-IP	System-Call
File-System-ID	arg-value-2	Audit-ID	
inode-ID	arg-value-3	Effective-User-ID	

Table 4.1: Chief Attributes

4.6 Rough Set Application to Attribute Reduction

Ideally, to find a reduct, we would just need to collect together as many as possible session logs and run Rosetta on them. However, this is computationally prohibitive since one would require to have an unlimited amount of resources, both in terms of memory and CPU time. To get around this situation, we ran a number of separate analyses, each of which considers a session segment, and then collected associated reducts. Then, to find the ***minimum common reduct*** (MCR), we performed an statistical analysis which removes those attributes that appeared least frequently. In what follows, we elaborate on our methodology to reduct extraction, which closely follows that one outlined by Komorowski [Komorowski et al., 1998].

4.6.1 Reduct Extraction

To approach reduct extraction, considering the information provided by the DARPA repository, we randomly chose 8 logs, out of 25, from the year 1998, and put them together. Then the enhanced log was evenly divided in segments of 25,000 objects, yielding 365 partial log files. For each partial log file, we made Rosetta extract the associated reduct using Johnson's algorithm and selecting a 100% support count (see § 4.6.2).

After this extraction process, we sampled the resulting reducts and using an frequency-based discriminant, we constructed an MCR. This MCR is constructed in a way that it keeps most information of the original data minimising the number of attributes. The largest reduct in the original set has 15 attributes and our minimum common reduct has 18 attributes. This is still a 66.66% reduction in the number of attributes. The 18 chief attributes are shown in table 4.1.

Regardless of size variations, the extracted reducts contain similar attributes. These similarities are our indicators to construct the MCR. To appreciate these similarities, both, the shortest and the largest reducts extracted, are shown in tables 4.2, and 4.3 respectively .

Access-Mode	arg-number-1	System-Call
inode-ID	Process-ID	

Table 4.2: Shortest Extracted Reduct

Proprietary	device-ID	arg-value-2	Effective-Usr-ID
Owner-Group	arg-value-1	exec-arg-1	Effective-Usr-ID
File-System-ID	arg-string-1	Socket-Type	Process-ID
inode-ID	arg-number-2	Remote-IP	System-Call

Table 4.3: Largest Extracted Reduct

4.6.2 Reduct Algorithms

We will explore the two of the most used reduct algorithms. First we need to note that currently the algorithms supplied by the Rosetta library (the library is described in 4.5.2) support two types of discernibility:

- *Full*: In this case the reducts are extracted relative to the system as a whole. With the resulting reduct set we are able to discern between all relevant objects.
- *Object*: This kind of discernibility extract reducts relative to a single object. The result is a set of reducts for each object in $\mathcal{A}$.

The first algorithm for reduct extraction is [Johnson, 1974]. This implements a variation of a simple greedy search algorithm. This algorithm extracts a single reduct only.

The reduct B can be found by executing the following algorithm where $\mathcal{S}$ is a superset of the sets corresponding to $g_A(U)$ and $w(S)$ is a weight function assigned to $S \in \mathcal{S}$ (unless stated otherwise the function $w(S)$ denotes cardinality). The algorithm is described in algorithm 2.

Algorithm 2 Johnson's algorithm

$B = \emptyset$
repeat
  Select an attribute a that maximises $\sum w(S)$ and $\forall S, a \in S$.
  Add a to B.
  Remove $S | a \in S$ from $\mathcal{S}$.
until $(1 - |\mathcal{S}|) \geq$ **(Support Count)**
return B

Approximate solutions can be provided by interrupting the algorithm's execution when an arbitrary number of sets have been removed from $\mathcal{S}$. The support count associated with the extracted reduct is the percentage of $S \in \mathcal{S} : B \cap S \neq \emptyset$, when computing the reducts a minimum support value can be provided. This algorithm has the advantage of returning a single reduct but

depending on the desired value for the minimal support count some attributes might be eliminated. If we use a support of 0% then all attributes are included in the reduct, if we use a value of 100% then the algorithm executes until $\mathcal{S} = \emptyset$.

We will now explore the Genetic Algorithm described by Øhrn and Viterbo in [Viterbo and Øhrn, 2000]. In order to find minimal hitting sets the use of a genetic algorithm is proposed. The algorithm's fitness function is presented below. In the fitness function, $\mathcal{S}$ is the multi set given by the discernibility function, α is a weighting between subset cost and the hitting fraction, and ϵ is used for approximate solutions.

$$f(B) = (1-\alpha) \times \frac{cost(A) - cost(B)}{cost(A)} + \alpha \times min\left\{\epsilon, \frac{|[S \in \mathcal{S}]|S \cap B \neq \emptyset|}{|\mathcal{S}|}\right\}$$

The subsets B of A are found by an evolutionary search measured by $f(B)$, when a subset B has a hitting fraction of at least ϵ then it is saved in a list. The size of the list is arbitrary. The function *cost* specifies a penalty for an attribute (some attributes may be harder to collect) but it defaults to $cost(B) = |B|$.

If $\epsilon = 1$ then the minimal hitting set is returned. In this algorithm the support count is the hitting fraction multiplied by 100. That means that if we select $\epsilon = 1$ the returned hitting sets will have a support count of 100 which is the same as in Johnson's algorithm.

4.6.3 Algorithm Selection

Previous to reduct extraction, we tested on the performance of the two Rosetta reduction algorithms, in order to find which is the most suitable for our work. Both reduction algorithms (see section 4.6.2) were used to extract a reduct from 25 log files. Log files were selected considering different sizes and types of sessions. A minimum common reduct set containing 14 attributes was obtained after 25 extraction processes. This amounts to a reduction of 72.5% in the number of attributes. In general, both algorithms yielded similar total elapsed times. Sometimes, however, Johnson's algorithm was faster. As expected, the total elapsed time involved in reduct extraction grows exponentially with the number of objects to be processed. For a 1,000 object log file the time needed to extract the reduct is 3 seconds and for a 570,000 is 22 hours. Also the size of the reduct increases according to the diversity of the log. For a log file with 1,000 objects, we found a reduct of 8 attributes, while for one with 570,000 objects, we found a reduct of 14 attributes. Table 4.4 shows time performance for reduct extraction using both algorithms and logs of different sizes.

In tables 4.5 and 4.6, we show the difference in attributes among reducts extracted using Johnson's and the genetic algorithm respectively. The difference between the algorithms is just one attribute in both reducts, *Access-Mode* and *Owner-Group*. Eventually both attributes ended up in the minimum common reduct as shown in table 4.1.

However, for larger log files, the instability of the genetic algorithm based mechanism became apparent. Our experiments show that this algorithm is

Algorithm	# of Objects	Extraction Time (hh:mm:ss)	# of Attributes in Reduct
Johnson	1,000	00:00:03.32	4
Genetic Algorithm	1,000	00:00:04	8
Johnson	5,000	00:02:46	11
Genetic Algorithm	5,000	00:02:32	12
Johnson	25,000	01:03:51	13
Genetic Algorithm	25,000	Crashed	NA
Johnson	570,000	21:40:05	14
Genetic Algorithm	570,000	Crashed	NA

Table 4.4: Reduct Extraction Time

Owner-Group	arg-string-1	Effective-Group-ID
inode-ID	arg-value-2	Process-ID
arg-value-1	Effective-Usr-ID	System-Call

Table 4.5: Reduct extracted with Johnson's algorithm

unable to handle log files containing more than 22,000 objects, each with 51 attributes. This explains why our experiments only consider Johnson's algorithm.

Even though the algorithms accept indiscernibility decision graphs (that is relations between objects) we did not use them, both because we wanted to keep the process as unsupervised as possible and because in order to build the graphs we needed to know in advance the relation between the objects, which is quite difficult even for the smaller logs which contain around 60,000 objects.

4.7 Reduct Validation—Experimental Results

An *association pattern* is a pattern that, containing wild-cards, matches part of an example log file. Given both a reduct and a log file, the corresponding association patterns are extracted by overlaying the reduct over that log file, and reading off the values [Øhrn and Komorowski, 1997]. Then the association patterns are compared against another log to compute how well they cover that log file information. Thus, our validation test consists of checking the quality of the association patterns generated by our output reduct, considering two log files. The rationale behind it is that the more information about the system the reduct comprehends the higher the matching ratio the association patterns of

Access-Mode	arg-string-1	Effective-Group-ID
inode-ID	arg-value-2	Process-ID
arg-value-1	Effective-Usr-ID	System-Call

Table 4.6: Reduct extracted with the genetic algorithm

that reduct will have.

To validate our reduct, we conducted the following three step approach. For each one of the 14 log files considered through our experiments:

1. Use the output reduct to compute the association patterns;
2. Cross validate the association patterns against all log files in the same year, including the one used to generate them; and
3. Collect and analyse the results.

Our validation results for the year 1998 are summarised in table 4.7. Table 4.8 shows the results for the year 1999, and table 4.9 shows the results of cross-year validation. The association patterns have relations only between attributes present in the MCR. A set of patterns was generated from each log file.

These association patterns describe the information system with a covering percentage of the patterns over the information system. The first column in tables 4.7, 4.8 and 4.9 indicates the corresponding log file used to generate the association patterns. The first row indicates the log file used to test the covering of the pattern. By contrast if we were to generate a set of patterns using all attributes, then those patterns will cover 100% of the objects in the table that generated them. The result is the percentage of objects we were able to cover using the attributes in our reduct.

These experiments were conducted on a Ultra Sparc 60 with two processors running Sun Solaris 7. Even though it is a fast workstation the reduct algorithms are the most computational demanding in all the rough set theory. The order of the reduction algorithms is $O(n^2)$ with n being the number of objects. With 700,000 an overhead in time was to be expected.

Calculating the quality of the rules generated with the extracted reducts is also time consuming. The generation of the rules took 27 hours (we used the larger tables to generate the rules) and another 60 hours to calculate the quality of the generated rules. In order to test the rules with another table we needed to extend the Rosetta library. This is because of the internal representation of data depends on the order an object is imported and saved on the internal dictionary of the table (every value is translated to a numerical form). The library has no method for importing a table using the dictionary from an already loaded table. To overcome the above deficiencies we extended the Rosetta library with an algorithm capable of importing a table using a dictionary from an already loaded table. This way we were able to test the rules generated in a training set over a different testing set. We also tested the rules upon the training set.

Finally the quality of the reduction set is measured in terms of the discernibility between objects. If we can still discern between two different objects with the reduced set of attributes then the loss of information is said to be minimal. Our goal was to reduce the number of attributes without loosing the discernibility between objects, so more accurate IDSs can be designed. Our results show

Training Log	Testing log							
	A	B	C	D	E	F	G	H
A	93.7	89.7	90.8	90.3	90.9	91.1	89.9	90.9
B	90.9	93.1	91.2	91.3	90.9	92.4	91.4	90.1
C	90.2	90.8	92.8	90.7	92.1	90.1	90.6	91.0
D	90.3	89.3	91.3	93.1	91.5	90.5	92.9	91.1
E	89.8	89.1	92.2	92.3	93.4	89.9	92.1	90.1
F	89.7	89.3	91.4	92.8	90.7	92.9	92.3	90.8
G	89.2	90.3	91.5	92.6	91.2	90.7	93.1	90.6
H	90.1	89.5	91.2	91.3	90.8	90.2	90.9	92.5
Size	744,085	2,114,283	1,093,140	1,121,967	1,095,935	815,236	1,210,358	927,456

Table 4.7: Covering % of the Association Patterns, 1998 Log Files

Training Log	Testing log					
	I	J	K	L	M	N
I	92.9	91.3	90.7	90.8	91.3	91.1
J	87.7	91.7	89.2	89.3	89.8	89.7
K	90.6	90.7	92.3	90.5	91.1	91.0
L	89.8	89.5	90.1	92.4	90.6	90.1
M	87.1	88.9	89.1	87.8	92.1	87.2
N	86.8	87.3	88.5	87.5	89.7	91.8
Size	820,855	490,896	630,457	520,358	220,658	297,766

Table 4.8: Covering % of the Association Patterns, 1999 Log Files

Training Log	Testing log	
	A	I
A	93.7	87.1
I	88.3	92.9
Size	744,085	820,855

Table 4.9: Covering % of the Association Patterns, 1998 vs. 1999

that we proved that about 90% of a BSM log file can be described using a third of its original attributes.

Even though the entire reduct extraction process is time consuming, it only needs to be done once and it can be done off-line. There is no need for live data, the analysis can be done with stock data. Once the reduct is calculated and its quality verified, we can use the smaller attribute set that best describes the information system. Thus reducing the space required to hold the logs and the time required for a proper intrusion detection.

4.8 Related Work

Log files are naturally represented as a two dimensional array, where rows stand for objects (in our case system calls) and columns for their attributes. These tables may be unnecessarily redundant. The problem of reducing the rows and columns of a table, we respectively call *object reduction* and *attribute reduction.* To the best of the authors' knowledge, the attempts to reduce the number of attributes prior to clustering have been few while this is still a big concern as there might be unnecessary attributes in any given source of information.

Lane and Brodley use an instance based learning (IBL) technique to model user behaviour [Lane and Brodley, 1999, Lane and Brodley, 2000] that works at the level of `Unix` user commands. Their technique relies upon an arbitrary reduction mechanism, which replaces all the attributes of a command with an integer representing the number of that command's attributes (e.g. `cat /etc/password /etc/shadow > /home/mypass` is represented by `cat <3>`). According to [Lane and Brodley, 1999, Lane and Brodley, 2000], this reduction mechanism narrows the alphabet by a factor of 14, but certainly at the cost of loosing important information. This is because the arguments cannot be eliminated if a proper distinction between normal and abnormal behaviour is to be done. For example, copying password files may in general denote abnormal activity. By contrast our method keeps all these attributes as they are main discriminants between objects and thus an important source of information.

Knop *et al* have suggested the use of correlation elimination to achieve attribute reduction [Knop et al., 2001]. Their mechanism uses a correlation coefficient matrix to compute statistical relations between system calls. [1]. These relations are then used to identify chief attributes. Knop *et al*'s mechanism

[1] Knop *et al*'s method resembles Principal Component Analysis [Rencher, 1995]

relies upon an numerical representation of system call attributes to capture object correlation. Since a log file consists of mainly a sequence of strings, this representation is unnatural and a source of noise. It may incorrectly relate two syntactically similar system calls with different meaning. By comparison, our method does not rely on correlation measures but in data frequency which is less susceptible to representation problems. This concludes our revision of related work on attribute reduction.

4.9 Conclusions

Based on our results we identified the main attributes of a BSM log file without sacrificing relevant information needed to keep discernibility between different objects. The method is expected to produce similar results if applied to different multi-variable log files. The method is performed only once and off-line; it provides no overhead to the intrusion detection process. Moreover it provides a small set of attributes for intrusion detection with a balance between information descriptiveness and compactness. Our results show that rough sets is a method suitable for the reduction task. The result of applying our method is a 66% reduction ratio. Rival techniques have reported on a greater reduction, but at the expense of information loss.

Our reduction method only assures that the reduced set keeps minimal the information loss with respect to discernibility. In order to assure that the reduced information is sufficient for an effective intrusion detection, empirical tests must be conducted. As we will demonstrate in chapter 7, the reduction obtained should be sufficient to explore intrusion detection methods that are computational intensive and were prohibitive. This chapter will also show that we keep key information needed for intrusion detection.

Chapter 2

State of the Art and Limitations of IDSs

2.1 Introduction

As computer networks grow so does the number of computer attacks. Computer security is one of the topics of interest to all system managers nowadays. According to [The Computer Security Institute, 2001, The Computer Security Institute, 2003] losses associated with computer crimes, reported from 1997 to 2001, summed up $1,004,135,495 USD. In 2002 these losses raised to $455,000,000 USD and in 2003 to $201,797,340 USD. So attacks are a major concern to computer industry.

There are many kinds of mechanisms to protect a computer site against threats. A well-defined security policy should be the first line of defence in any site [Holbrook and Reynolds, 1991]. A site security policy includes a way of both preventing intrusions and providing counter measures. Prevention strategies range from user education to the use of a number of devices, including an Intrusion Detection System (IDS).

A computer attack can be as simple as the reception of a single packet containing an instruction to destabilise a host. It can also be as complex as a series of independent requests that together lead the site to an insecure state by compromising one of the hosts. Currently there is not reliable automated method for detecting intrusions. According to [Alessandri et al., 2001], up to 90% of the alarms (reports of suspicious activities) generated by most IDSs are false alarms. Some IDSs check only file integrity or the content of a single network packet. Most sophisticated IDSs also check for short sequences of low-level calls to the operating system but without paying attention to their content [Warrender et al., 1999, Tan and Maxion, 2002, Qiao et al., 2002, Yeung and Ding, 2003, Wagner and Soto, 2002].

There have been many attempts to solve the problem of detecting attacks in progress but all the methods fail to detect unknown attacks if those are more

complicated than a simple network packet or are generated by a trusted source. Or in the case of known attacks the detection methods fail to detect attacks disguised as normal activity. Also current intrusion detection methods seem to be incapable of providing acceptable low rates of false alarms and undetected attacks. In order to overcome the above problems, we have resorted to methods commonly used in natural language programming (NLP). These methods have proved to be succesful for speech recognition, which might be seen as a similar problem to sequence identification. We will make the detection based on session segmented in smaller parts. In speech recognition there are techniques that detect phonemes before words, and words before phrases. This segmented approach is the one that we think, will provide a more reliable method since it can extract more characteristics to make a better analysis.

Some of the most interesting works, have proposed analogies with the immune system (IS), examples of these methods are [Hofmeyr and Forrest, 2000], [Warrender et al., 1999] and [Kim and Bentley, 1999b, Kim and Bentley, 1999a]. In the IS self/non-self discrimination is based upon chemical bonds formed between protein chains and detector cells [Ferenčák, 1993]. The network traffic is viewed as the universe formed from the union of self and non-self. All these approaches have failed at detecting attacks disguised as normal traffic. By dividing the traffic and then grouping each the detection result for each segment we believe that we can outperform methods based on the IS.

The remainder of this chapter is organised as follows: §2.2 present an introduction to computer security, and terminology that is used throughout this dissertation; if the reader is familiar with general teminology and concepts of computer security, §2.2 can be skipped. §2.3 presents the taxonomy of IDSs; §2.4 reviews IDSs development and evolution; §2.5 describes different intrusion detection methods; §2.6 presents current trends in IDSs research; §2.7 explains a kind of attack that has proved difficult to current IDSs; in §2.8we give an overview what we consider to be the most distinctive limitations of current IDSs; finally, §2.9 briefly depicts our data source for intrusion detection.

2.2 Computer Security

In order to have a better perspective of the intrusion detection methods we need to know at least some basic concepts regarding computer security. If the reader is acquainted with computer security, it is safe to skip this section.

The computer security field is primarily concerned with protecting valuable data. There are three goals in computer security as defined in [Amoroso, 1994]:

- Confidentiality refers to the prevention of unauthorised disclosure of information.
- Integrity deals with the prevention of unauthorised modification of information.

- Availability prevents unauthorised withholding of information or resources.

The way these goals can be attained is defined in a security policy which can be implied or perfectly described. If an organisation wants to keep confidentiality, integrity and/or availability of some information it must constitute a security policy even if the policy is not well-defined. At least a general and superficial security policy should be defined.

As information we are protecting has some value, any failure in keeping it secure can be seen as an economical loss i.e. imagine the losses if the source code for an operating system is stolen or tampered with.

Some examples of these losses can be seen in the "Computer Crime and Security Survey" [The Computer Security Institute, 2001, The Computer Security Institute, 2003]:

> 35% of 186 respondents were willing and/or able to quantify their financial losses. These 186 respondents reported $377,828,700 in financial losses. (By contrast, the losses from 249 respondents in 2000 totalled only $265,589,940, in 2001 $377,828,700, in 2002 $455,848,000 and in 2003 it totalled $201,797,340 USD. The average annual total over the three years prior to 2000 was $120,240,180.).

We have seen the security goals and the economical risks in the failure of keeping them. Next we will analyse security in operating systems to explain some causes of vulnerabilities that can lead to an attack.

2.2.1 Security in Operating Systems

An operating system has conflicting needs: to share resources while protecting them. In the early days, security consisted of a lock and a key: the system was physically guarded and only authorised users were allowed in the vicinity. However with the advent of data communication, networking, the proliferation of personal computers, and modern telecommunication software, computer security has become much more difficult to achieve.

Systems that were once unaccessible have now become vulnerable to attacks, and because system security is a relatively recent problem, many systems have little protection built into them. The major problem is that system managers must balance two opposing goals: to keep the system accessible to its authorised users and to protect it from other people who have no right to access it.

Not all security breaks are malicious; some are purely accidental unauthorised use of resources. But some breaks stem from a purposeful disruption of the system's operation. Malicious or not, a break in security severely damages the system's credibility.

Accidental flaws are known as security holes due to the opportunity they give for an attack to take place. Most of the time the holes remain undetected. But sometimes a security hole goes public and an attack to take advantage of it is developed. This is called an *exploit*.

2.2.2 Security Holes

The two most repetitive accidental flaws are described below:

- Accidental incomplete modification of data occurs when non-synchronised processes access data records and modify some but not enough of the record's fields [Flynn and McHoes, 1991].
- Data values are incorrectly encoded when fields are not large enough to hold numeric values stored there. For example, when a field is too small to store a numerical value, FORTRAN will store a string of asterisks and COBOL will cut off a part of the higher order digits. Neither of these errors would be discovered at the time of storage; it would be discovered only when the value is retrieved. That is an inconvenient time to make such an unpleasant discovery [Flynn and McHoes, 1991].

These kind of errors or programming bugs are the main cause of security holes. And when discovered these security holes become exploits such as a not bounded buffer that leads to a possible overflow attack

The vast majority of attacks are caused by accidental flaws or the failure to follow a security policy. A description of some of these attacks follows.

2.2.3 Attack Description

Intentional unauthorised access (regardless its cause) is the most damaging attack in security, and the rest of this subsection will be devoted to some instances of it.

- Browsing refers to unauthorised users searching through storage, directories, or files for information they should not have the privilege to read. Storage refers to main memory or to unallocated space on disk or tapes. Sometimes browsing occurs after an active process has finished. When a section of main memory is allocated to a process, the data from a previous process allocated in that same memory section often remains in there, it is not usually erased by the system, and so it is available to a browser. The same applies to secondary storage [Tanenbaum, 1992]. Put the other way round, when a process deallocates memory, data remains until another data are written over that memory region.
- Wire tapping is a direct intervention over the transmission line. Wire tapping is said to be passive if an intruder does not change the content of a transmission and active otherwise. There are two reasons for passive tapping: to copy data while bypassing any authorisation procedures and to collect specific information (such as passwords) that will permit the tapper entering the system at a later time. Two methods of active wire tapping are "between lines" transmission and "piggy back" entry. Between lines does not alter the messages sent by the legitimate user, but it inserts additional messages into the communication line while the legitimate user

is pausing. Piggy back intercepts and modifies the original messages. This can be done by breaking the communication line and routing the message to another computer that acts as a host. For example, the tapper could intercept a log-off to the user, and then continue the interactive session with all the privileges of the original user [Amoroso, 1994].

- Repeated trials is a method used to enter systems that rely on passwords. If an intruder knows the basic scheme for creating passwords such as password length and symbols allowed to create it, then the system can be compromised with a program that systematically goes through all possible combinations until a valid combination is found. This is not as long a process as one might think if passwords are short. Since an intruder does not need to break a specific password, the guessing of any password allows entry to the system and access to its resources [Amoroso, 1994].

- Trash collection is an evening pastime for those who enjoy perusing anything and everything thrown out by the computer department: discarded computer tapes, disks, printer ribbons, and printouts of source code, programs, memory dumps, and notes. They can all yield important information that can be used to enter the system illegally [Amoroso, 1994].

- Trap doors are unspecified, non-documented entry points to a system. Trap doors can be either accidental or they can be put by a system programmer for future use. They may also be incorporated into a system by a "Trojan horse"[1] or a destructive "virus"[2] as described by [Amoroso, 1994].

Some of these attacks are caused by security holes while others are the result of a poor security policy (i.e. viruses and Trojan can be avoided by an anti-virus software). Some attacks can not be avoided like repeated trials. But all can be detected.

We have seen what kind of exploits an attacker may use to compromise a system. But we also need a categorisation of the attacks based on their goals to know what kind of data to protect. One such a categorisation is given below.

2.2.4 Attack Categorisation

It is notorious that not all attacks are similar. All attack components have specific goals that include obtaining information about a computer/network system with intent to use that information as part of an exploit, achieving some level of privilege on a computer system not specifically granted by a system administrator, or violating some explicitly stated security policy. Five major attack

[1] A Trojan horse program is defined as any program that is expected to perform some desirable function, but actually performs the other way round

[2] A virus program is defined as any Trojan horse program that has been designed to self-reproduce and propagate so as to modify other programs to include a possibly modified copy of the virus.

categories are presented in [Haines et al., 2001] to group attack components with regard to the actions and goal of the attacker. We discuss these categories below.

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

User to Root Attacks User to Root (U2R) exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system. There are several different types of U2R attacks. The most common U2R attack is the buffer overflow. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system.

Remote to Local Attacks A Remote to Local (R2L) attack occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorised access to a local account on a machine. The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or poorly configured system security policies.

Probes In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities. These network probes are quite useful to an attacker who is preparing a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools ([Farmer and Venema, 2002], [SAINT Corporation, 2002], [Focus, 2001]) enable even a very unskilled attacker to quickly check hundreds or thousands of machines on a network for known vulnerabilities.

Data Data Attacks involve someone (user or administrator) performing some action that they may be able to do on a given computer system, but that they

are not allowed to do according to site policy. Often, these attacks will involve transferring "secret" data files to or from sources where they do not belong.

Based on this categorisation we can say now that our method will focus on U2R and R2L attacks. Thus we will concentrate on user behaviour. In the next section we present different taxonomies for IDSs.

2.3 Classification of IDSs

IDSs are mainly categorised based on the detection scheme and the source of information. Based on the detection scheme there are two kinds of IDSs, misuse-detection and anomaly-detection [Alessandri et al., 2001].

Misuse IDSs (MIDSs) use a pattern matching procedure based on rules. This allows the IDS to detect any attack with a known signature. These kinds of IDSs are very effective against known attacks but have problems with detecting novel attacks of unknown signatures. It also fails to effectively detect stealth attacks (attacks hidden in valid patterns) or attacks distributed in multiple sessions. Detection methods used in MIDSs include expert systems, signature analysis, and data mining [Barbará and Jajodia, 2002].

An anomaly IDS (AIDS) relies on some kind of statistical profile abstracting out normal user behaviour; user actions may be observed at different levels, ranging from system commands to system calls. Any deviation from a behaviour profile is taken as an anomaly and therefore an intrusion. Approaches to anomaly detection include statistical methods, genetic algorithms, neural networks and data mining [Barbará and Jajodia, 2002].

Based on the data source an IDS, there are three kinds of IDSs, host-based, network-based and application-based IDSs.

Host-based IDSs usually use system and application logs to obtain records of events, and analyse them to search for an intrusion [Bishop, 2003]. The advantage of collecting information at log level is that is less susceptible to encryption.

Network-based IDSs use a variety of devices to monitor network traffic. This approach can detect network-oriented attacks, such as a DoS attack introduced by flooding a network [Bishop, 2003]. An IDS of this kind is capable of monitoring a large number of hosts at the same time.

Application-based IDSs monitor application behaviour. An application behaviour can be modelled in a way that deviations from known benign behaviour are signalled as an intrusion; e.g. a printing application should never try to read a password file.

Having reviewed how IDSs are classified, we now present a chronological development of IDSs.

2.4 An Account for the Development of the IDS Area

The first model of an IDS goes back to the mid 1980's with the (seminal) work of [Denning and Neumann, 1985]. Denning and Neumann suggested a centralised architecture. Their approach to Intrusion Detection focuses only on reachability of insecure states; traffic inspection is neglected. To detect an intrusion, the method uses a pattern matching procedure based on rules. Many commercial IDSs are based on Denning and Neumann's model. Example centralised IDS are Intrusion Detection Expert System (IDES) [Denning and Neumman, 1987], Network Intrusion Detection Expert (NDIX) [Bauer and Koblentz, 1988] and Network Anomaly Detection, Intrusion Reporter (NADIR) [Hochberg et al., 1993]. The pattern matching procedure used in these systems vary from one another, some IDSs target normal behaviour and others the opposite. Rule-based expert systems or weighted functions are often used to detect abnormal behaviour. By contrast, the use of covariance-matrix based mechanisms is preferred to detect normal behaviour.

Netowrk-based IDSs represent an alternative approach to centralised IDS. Instead of considering reachability of insecure states, a network-based IDS analyses only network traffic. A network-based IDS called Network Security Monitor (NSM) was first developed by [Heberlein et al., 1990]. A *Network IDS* combines the two approaches above into a single one. [Snapp et al., 1991] extended the Denning and Neumann model to a network model yielding a system called Distributed Intrusion Detection System (DIDS) the model has a distributed architecture. DIDS uses both statistical analysis and rule-based expert systems to detect attacks. Notice that even though the architecture is distributed, data is analysed on a single machine.

IDSs of the above sorts present three major weaknesses. First and most important, lack of robustness (e.g. if the central server crashes so will the whole system). Second, difficulty of maintenance. Being monolithic Al, any one such an IDS should be configured according to each host it resides in. Changes in network configuration result in unnecessary IDS maintenance. Third, inefficiency. System architecture causes high network loads leading to a long latency time. Fourth, scalability. As in the second problem, monolithic Al nature of the system makes it hard to add new hosts or nodes.

IDSs later evolved into a distributed hierarchical model. The architecture of a distributed IDS consists of three layers. The command and control system is at the top layer, information gathering is achieved at the middle layer and system monitoring at the bottom layer. Distributed IDSs are designed to monitor large scale networks and, unlike previous models, scalability is not an issue. Some projects in progress using this approach are The Graph-based Intrusion Detection System [Stainford-Chen et al., 1996] and Event Monitoring Enabling Responses to Anomalous Live Disturbances [Porras and Neumann, 1997]. In both systems each unit monitors a single area and sends the result to the layer above instead of sending all the audit trail. Thus nodes at uppermost layers

need only to analyse results from layers below to them. By sharing information between peers the model is able to detect coordinated attacks.

With this model the problem of network load is partially solved but the robustness problem still exists. If a node at a higher level crashes so will its descendants. And if a node crashes the detection of coordinated attacks becomes difficult to perform.

As for the time of writing these are the approaches used in commercial IDSs. We will describe in the next section the most common detection methods.

2.5 Most Common Methods in IDSs

This section aims to briefly describe the most important methods used in both commercial IDSs, and research IDS prototypes. We illustrate current methods' weak points so as to demonstrate that a more accurate method is needed.

Regarding data representation there are mainly two kinds of detection methods: context dependent and context independent. Both kinds of methods are described below along with the most representative frameworks of each method.

2.5.1 Context Dependent Methods

Context depending methods are characterised by the representation they use to make the detection. They use the data as is, e.g. when checking a network packet the method verifies source and destination addresses and compares the data without further treatment.

The most common method for context dependant intrusion detection is the rule-based one. Network traffic may be captured using a graph, where nodes represent host states and transitions requests to the host. Some states are labelled as insecure. An attack is then any path that leads to an insecure state. Therefore an analysis of that graph is a method for detecting such an attack. As a consequence of the graph representation an attack can be analysed using expert systems, statistical methods, and simple rule matching.

Expert System Methods

In the expert system approach a series of rules, each of the form if-then-else, is used to determine if the current session is malicious or not. This method is analogous to follow the transitions in a graph thus if an state reached by a transition is an insecure one then an alarm is triggered. The method has the advantage of scalability but not without a cost: detecting a large number of attacks normally implies a decrease in performance due to the large number of rules that need to be compared with.

In these methods both data representation and discrimination are done at a more abstract level. Since expert systems are used to verify if an action leads to an insecure state we need to know all the transitions in advance. Even with the aid of heuristics, the set of rules needed to represent a considerable number of attacks is enormous.

The same problem applies if the expert system is modelling user behaviour. If an attack is not contemplated in the rules the expert will system fail to recognise it. If the user behaviour is being modelled then any attack that matches the rules of certain user will be overlooked. Also if a user action differs from the rules defining its behaviour a false alarm will be triggered.

Some commercial and research IDSs that use the expert system approach are:

- NADIR™: Network Anomaly Detection and Intrusion Reporter [Hochberg et al., 1993]. NADIR monitors network authentication (Kerberos) and mass file storage activity (Common File System).
- AID: Adaptive Intrusion Detection system (research prototype) [Sobirey et al., 1996]. The expert system uses a knowledge base with state oriented attack signatures, which are modelled by deterministic finite state machines and implemented as rule sequences.
- EMERALD™: Event Monitoring Enabling Response to Anomalous Live Disturbances [Porras and Neumann, 1997]. The system has misuse detection method called eXpert which is based on the P-BEST expert system [Lindqvist and Porras, 1999].
- Expert-BSM™ [Lindqvist and Porras, 2001]. The system uses a forward-reasoning expert systems that analyses audit trails.
- NetSTAT: Network-based State Transition Analysis Tool (research prototype) [Vigna and Kemmerer, 1999]. NetSTAT represents state transition diagrams within its rule-base and uses them to seek out those state transitions within the target system that correspond to known penetration scenarios.

Statistical/Probabilistic Methods

The statistical detection method compares current behaviour profiles against historic (expected) behaviour profiles. In the statistical method an anomaly (which can be taken as an attack) takes place when current behaviour deviates beyond normal behaviour. With this approach detection is made upon known behaviour of the user.

Bayes is a popular probabilistic method that analyses effects or observable behaviour and gives a probability of an attack being the cause of that anomaly.

If we have a complete distribution function of user behaviour and attacks, statistical/probabilistic methods are the right choice in terms of accuracy and performance. If we knew those probabilities that would mean that we would know all the attacks and have examples of all the causes of them or we could describe completely user behaviour which is impossible.

If a profile is used to detect deviation from normal user behaviour then any valid action that deviates from that behaviour is signalled as an anomaly or any attack similar to normal behaviour will be missed by the IDS. In order

to learn new behaviour a probability describing such behaviour (the attribute on which is extracted the probability depends on the IDS) must be given. Also statistical methods fail to represent relational structures between components of the attacks and without these structures learning new attacks is a real problem.

Some commercial and research IDSs that use the statistical approach are:

- SIDS: Statistical Intrusion Detection System (research prototype) [Javitz et al., 1986]. This is a host based IDS which profiles user behaviour based on audit trails of the host.
- CMDS™: Computer Misuse Detection System [Proctor, 1994]. The statistical detection system compares current behaviour profiles to historic behaviour. The data is extracted from the audit trails of the host.
- NIDES: Next-generation Intrusion Detection Expert System (research prototype) [Anderson et al., 1995]. NIDES statistical analysis maintains historical statistical profiles for each user and raises an alarm when observed activity departs from established patterns of use for an individual.
- EMERALD™: Event Monitoring Enabling Response to Anomalous Live Disturbances [Porras and Neumann, 1997]. EMERALD is the continuation of NIDES and it includes an anomaly detection method called eBayes [Valdes and Skinner, 2000] which is based on the Bayes theorem. EMERALD analyse TCP sessions at periodic intervals.

Simple Rule Matching

Rule matching methods use a simple string compare approach. If a network packet or an audit log contains certain string defined in a rule then an alarm is triggered. A problem with this approach is that a large amount of space is needed to represent all the rules to be checked against. Therefore the method only analyses a small part of the universe. Even though the method is very accurate, in order to detect an attack a signature should be present and if the attack varies a little then another signature is needed. Rule matching is good because of the low false alarm ratio but they fail to detect unknown attacks. In order to do this a dataset containing all the possible attacks (both known and unknown) is needed which is virtually impossible.

Some frameworks that incorporate this method are:

- NID™, formerly NSM: Network Security Monitor [Heberlein et al., 1990]. NID can examine network packets for strings associated with known attacks. The system can be configured to monitor an specific domain (hosts or subnetworks).
- ASAX™: Advanced Security audit trail Analyser on uniX [Habra et al., 1992]. ASAX is a distributed audit trail analysis system. It is based on a rule-based analysis engine to detect known attack patterns.

- GrIDS: Graph Intrusion Detection System (research prototype) [Stainford-Chen et al., 1996]. The system builds activity graphs which approximately represent the causal structure of large scale distributed activities. These graphs are then compared against known patterns of attacks.
- Alert-Plus™ [Computer Security Products Inc., 2002]. Alert-Plus is a rules-based system that compares events recorded in an audit trail against custom-defined rules.
- Cisco Secure IDS™ (formerly NetRanger™) [Cisco Systems, 2002]. The NetRanger intrusion detection engine uses signature recognition, which can be either context- or content-oriented.
- SecureNet PRO™ [MimeStar, 2002]. The system uses string matching combined with Shared Decision Logic for signature analysis.
- Snort© [Free Software Foundation, 2002]. Snort is based on linear string matching of TCP/IP traffic dump with a set of rules.

A description along with some examples of context independent methods follows.

2.5.2 Context Independent Methods

Context independent methods generally transform input data into binary data so that the origin of data is irrelevant to the success or failure of the method. This is because the discrimination of classes is done at a bitwise level. Many of these methods have been proved to be effective in pattern recognition and optimisation. The possibility for failure does not rely on the method but on data representation i.e. which portion of data is used as the input or output to the method. If representation is adequate then the method will succeed in the detection, otherwise it will fail. In other words the problem with these methods centres around the data chosen to be the input and the way the programmer transforms these data so it can be used by the method. Since the method does not vary according to the input, it can be recycled for use with more data by just adjusting the representation of it. Unlike context dependent methods, context independent methods do not care about the origin of data, instead they only analyse bit positions in these data regardless its nature i.e. analysing a network packet a match can be made from bits originated from different parts of the packet (source address and port).

N-contiguous Bit Matching

LYSIS is a framework developed by [Hofmeyr and Forrest, 2000] where multiple data are represented as a single binary string. Detectors containing strings that do not match normal traffic are created. Then an n-contiguous bit matching is done over the strings to see if the traffic being analysed corresponds to something

abnormal. N-contiguous bit matching consists of comparing two binary strings using a window of length l, and if a similarity of at least n bits is found then there is a match. If n equals l then the strings must be identical to have a positive match.

Because the match is based on unseen traffic, the method is said to be of type negative selection. In LYSIS the binary string contains information of source and destination addresses along with the port. The match may use bits from source address and port or any other data. This framework deploys a lot of detectors along the network and if a new normal traffic is presented then all the detectors should be removed from the system or they take the risk of detecting this new normal traffic as an attack. Also the number of detectors must be huge in order to detect all unknown (not necessarily malign) traffic. If not enough detectors are deployed then a non-contemplated attack would be taken as normal traffic. When in mode of misuse detection (a non-self string is fed to the system and co-stimulated), detection rate reported is 75% and false positive rate of 6%.

Genetic Algorithms

Genetic algorithms (GAs) are "search algorithms based on the mechanism of natural selection and natural genetics". GASSATA [Mé, 1998] is a project that uses GAs to make the discrimination of abnormal situations. The GA presented in [Mé, 1998] works as follows: it creates a population of the host characteristics (properly represented as binary strings), selects the characteristics that best represent an insecure state of the host, and mixes the characteristics that fit these criteria evolving an optimum set of characteristics that represent an undesired state. GASSATA then uses these characteristics to make the detection.

According to [Mé, 1998], searching for 24 attack scenarios GASSATA reaches a 99% convergence by generation 100. The number of generations needed to find a solution increases as the number of scenarios is incremented. Time needed to find a solution also increments but in this case it is incremented exponentially e.g. with a 24 attack scenario GASSATA needs about 20 seconds to find a solution and with a 200 attack scenario it needs 600 seconds to find a solution. In order for the GA to know if the scenario will be a cause of an attack a set of examples must be provided. The GA is unable to know if a given scenario will be an attack unless someone provides that information. So the GA will be unable to detect unknown attacks since it does not know the scenario that leads to them. Also a GA demands a lot of computation to find a solution and that could consume resources on the machine hosting it.

Neural Networks

ACME! is a framework described in [Computer Security Research, 2002] where neural networks are used for intrusion detection. A neural network (NN) structure is a collection of parallel processors connected together in the form of a directed graph, organised such that the network structure lends itself to the

problem being considered as defined in [Freeman and Skapura, 1991]. In this framework the NN is trained with known examples of attacks so it can recognise variations of them. The NN in ACME! receives a stimuli vector from the network traffic at a given time and produces a numerical output. The output must be interpreted in order to know if the traffic represents an attack. The first problem with this method is that it only analyses traffic at a given moment and then the next moment but without relating the traffic over time. The problem with the NN approach is that if an attacks differs too much from the attacks used in the training phase then the IDS would be incapable of making a good detection. Another problem with NN is the difficult to acquire new knowledge, the NN must be trained all over again which is time consuming and need a fair amount of examples of a new attack in order to detect it correctly. Due to the generalisation of a NN, it must be incapable of detecting attacks disguised as normal traffic without having a high false-alarm ratio.

Context independent methods are not very scalable since the input in these methods often has fixed length. Due to this characteristic it is impossible to add features to the input pattern. Also if new patterns are to be recognised, re-compilation is necessary for the methods to be able to detect it.

Unlike context dependent methods, these methods are able to find hidden relations in data which is useful as characteristic extraction as a way of preprocessing. These characteristics can later be used in a context dependent method. Thus a combination of both methods can prove to be a useful IDS. Currently there is some development on new models. The problems that need to be addressed for IDS design will be described in the next section.

2.6 Focus of Current Research on IDS

Current research aims to overcome the problems existing with current IDS technology [Nolazco et al., 2004].

1. Undesirable number of false-positives and false-negatives.
2. Vulnerability to complex attacks.
3. High network loads.
4. Non scalability.
5. Lack of robustness.
6. Lack of a data source integration mechanism.
7. Inefficient update of attack signatures.
8. Lack of a terminology standard.

A great variety of current research in IDSs, is pointed in one common direction: distributed systems. Ours is With computer networks growing at an

exponential rate, the systems to guard this networks should develop at the same rate. Also there is a need for a portable IDS so it could be used among multiple architectures (contemporary networks mix POSIX based architectures with Windows™ or even Macintosh™) and the implementation should work transparently to administrators.

In the next section we analyse a special kind of attack, an attack that is disguised as normal behaviour which is called a *mimicry attack* [Wagner and Soto, 2002].

2.7 Mimicry Attacks

In order to explain mimicry attacks, we make the assumption that an attacker knows the detection method's inner working. When a detection method becomes popular is sensible to think that its algorithms and specifications will be publicly available. Thus, security through obscurity is not very reliable. A mimicry attack is defined by [Wagner and Soto, 2002] as a variant of an attack that aims to masquerade as ordinary, non-malicious behaviour [Wagner and Soto, 2002]. The deception is achieved by transforming the known attack in any conceivable way, provided that it does not lose its harmfulness. For example, in a host-based MIDS, where an attack takes the form of a sequence of system calls, we could build a mimicry attack out of other in either of three ways: i) by swapping one or more subsequences of system calls for other, functionally equivalent; ii) or by inserting purposeless system calls (which we henceforth call *no-ops*); or iii) by a combination thereof.

For a system call to be a no-op it needs not to modify the attack state. A system call that can be used as no-op is *chdir* with argument ".". Some system calls that can not be used as no-ops are `exit`, `pause`, `vhangup`, `fork`, `alarm`, and `setsid` since they have a negative impact on the success of the attack. Another way to create a mimicry attack, is to substitute any call to `read` on an open file with a call to `mmap` followed by a memory access.

Since it is a sequence of system calls, an attack is a very simple, restricted program, containing no recursion. Then, to approach mimicry attack detection, one might be tempted to use the normalisation procedure shown in Algorithm 1. This normaliser however is not realisable, because not every no-op can be dis-

Algorithm 1 Normalisation before detecting mimicry attacks

$I \leftarrow$ input sequence of system calls
$I \leftarrow$ the result of striping off every no-op from I
Parse I searching for a variant v of a known subsequence s of system calls
while there is one such a variant v **do**
  $I \leftarrow$ the result of replacing v with s
  Parse I searching for a variant v of a known subsequence s of system calls
end while

tinguished via a syntactical analysis. An no-op can be realised by at least two

means: i) a system call which on execution returns failure; and ii) a system call which on execution does not interfere with the internal workings of the attack. The second kind of no-op is therefore state dependant and cannot be syntactically characterised.

A normalisation approach also faces another technical difficulty: it is necessary to prove that no variants of any two distinct subsequences of system calls clash one another. Put differently, the normaliser will certainly yield a normal form, but this normal form is not necessarily unique. One has to prove convergence [Klop, 1992]. We will explore how to tackle mimicry attacks in more detail in Chapter 7.

There have been many attempts at detecting mimicry attacks [Schonlau et al., 2001, Maxion and Townsend, 2002, Scott et al., 2003, Boleslaw and Yongqiang, 2004], all will be covered in detail in chapter 7 to contrast our proposed solution.

This finishes our revision of the state of the art in intrusion detection. We now proceed to pinpoint the main flaws with current IDSs. These flaws are the motivation for the rest of the dissertation.

2.8 Summary of Current IDSs Limitations

As we illustrated most current IDSs are incapable of dealing with unknown attacks (mostly expert system and rule-based). Some are capable of detecting them (like the statistical methods) but lack the capacity of learning new user behaviour in a short period. Another problem with current IDSs is the necessity of a large dataset of known attacks to be able to detect them (NN, expert systems, rule-based, Bayes). Summarising, some methods are able to detect unknown attacks but fail in detecting hidden attacks, others precisely detect attacks even if they are disguised but need to know the attack in advance. Others are just too resource consuming. Some methods lack a relational capacity in order to learn new characteristics. Overall some methods have some advantages but none has enough advantages to make it a remarkable IDS.

We conclude that the main challenges that current IDSs face are:

1. High false positive ratio which decrement confidence in the IDS [Axelsson, 1999, Axelsson, 2000].

2. High information load, which incapacitates the IDS for a real time detection, and gets in the way of user profiling.

3. Lack of scalability. As computer networks tend to grow, so does the number of active services in the system. Current IDSs do not scale up to monitor all these new services.

4. Difficulty to detect mimicry attacks. A system call normalisation to filter out no-ops is not possible so new methods should be suggested.

5. Lack of robustness which makes the IDS vulnerable to attacks.

6. Lack of a data source integration mechanism.

7. Inefficient update of attack signatures. A misuse IDS to be effective, needs to have an updated signature data base.

8. Lack of a terminology standard. The communication between different security components needs a standard. To efficiently update security policies, standardised communication of each module result is needed.

The first four problems problems will be addressed in the following chapters, the rest of the problems are out of the scope of this dissertation. Before we finish this chapter we will describe the data source used in our experiments.

2.9 Data Source Used in Dissertation Experiment's

In this dissertation we restrict ourselves to the inspection of sequences of system calls, the system calls were extracted from the DARPA repository [Haines et al., 2001].

A system call consists of an mnemonic and a number of arguments. The number of arguments varies from one system call to other and there might be a few dozens of argument types in a typical operating system. Thus, if we were to use an sequence of system calls as the input to an intrusion detector, then we will have to consider every possible system call as well as each of its associated arguments. Since this would make the detection intractable, IDS often consider only the system call mnemonic, dropping out all of the arguments, hence missing possible key information.

2.9.1 BSM and the DARPA Repository

Being experimental, computer science involves methods that cannot be theoretically analysed. These methods ought to be thoroughly tested. To avoid suspicion, computer scientist often avoid building their own test set appealing to third parties for help. DARPA has become one such a third party in the context of testing IDSs [Haines et al., 2001]. Although the DARPA repository involves IDS analysis considering the output of various tools (TCP-dump, BSM log files and output of the UNIX ps command), we have selected BSM. This is because BSM describes the result of executing system calls and so the output data is less subject to encrypted attacks and might help detecting a distributed attack. BSM is part of the SunSHIELD Solaris© log system [SunSHIELD BSM, 2000]. Every security sensitive system call, makes BSM generate a new entry into the associated audit record.

The logs in the DARPA repository have daily log files and seven weeks for 1998 and five for 1999. The number of objects in each daily BSM log can vary from 200,000 to 1,300,000. Each object in a BSM file has a variable number of attributes depending on the system call that generated the entry. For a

detailed description of the BSM format, and possible attributes for each object, the reader is referred to appendix A.

In the next chapter we will present the theory of the methods used to tackle the afore-mentioned challenges.

Chapter 5

N-gram Session Reduction

5.1 Problem Description

IDSs have to analyse a huge amount of information to detect an intrusion. If the information overwhelms the IDS it might miss an intrusion. Therefore we need a method to reduce the amount of information an IDS analyses. In chapter 4 we described a method to reduce the number of attributes in a log file. Here we will deal with reducing the length of object sequences in a log file.

The problem in this case is to build a method which takes as input a session and removes from it redundant information so that the output is significantly shorter. At the same time the output must keep as many of the original tracks of an intrusion as possible.

A log file contains records describing the activity of a number of computer sessions. Each session consists of a sequence of system calls. From our observations we discovered that a session contains many repetitive subsequences.

We believe that folding the most repetitive non-attack subsequences in an audit session, significantly reduces its size with minimal impact on intrusion detection. A sample of these repetitive subsequences is shown in figure 5.1. The first column indicates the sequence of system calls that conform the n-gram; second column is the frequency of such n-gram; and third column is the tag used to substitute the subsequence when found. In figure 5.2 we can appreciate the insertion of the fresh tags instead of the original subsequence of system calls. The output from the reduction method is a shorter sequence of system calls. The output has a bigger vocabulary than the input.

The remainder of this chapter is organised as follows: §5.2 presents the reason to chose n-grams over other feature extraction methods like HMMs; §5.3 describes the technique used to fold the sessions; §5.4 gives a brief overview of how and why we chose data for identifying repetitive n-grams; §5.5 shows the use of the n-gram models to identify the n-grams with higher occurrence frequency; §5.6 summarises the reduct obtained throughout our investigations; §5.7 is a description of our validation experiments and the selected data for

Tri-gram	Frequency	Substitution tag
open(2)_-_read||success mmap||success open(2)_-_read||success	5795	abbrSysCall121
munmap||success mmap||success mmap||success	7615	abbrSysCall132
close(2)||success close(2)||success close(2)||success	79753	abbrSysCall148

Figure 5.1: Sample tri-grams with its associated frequency

1:fork(2)||success	18:close(2)||success
2:close(2)||success	19:open(2)_-_read||success
3:fcntl(2)||success	20:mmap(2)||success
4:abbrSysCall148	21:close(2)||success
5:execve(2)|date|success	22:open(2)_-_read||success
6:abbrSysCall121	23:abbrSysCall132
7:abbrSysCall132	24:mmap(2)||success
8:mmap(2)||success	25:close(2)||success
9:close(2)||success	26:close(2)||success
10:open(2)_-_read||success	27:munmap(2)||success
11:abbrSysCall132	28:open(2)_-_read||success
12:mmap(2)||success	29:close(2)||success
13:mmap(2)||success	30:ioctl(2)||failure
14:close(2)||success	31:ioctl(2)||failure
15:open(2)_-_read||success	32:abbrSysCall148
16:abbrSysCall132	33:exit(2)||success
17:mmap(2)||success	

Figure 5.2: Reduced session 7 substitutions using the 3 tri-grams in figure 5.1

such experiments; §5.8 shows the results obtained from applying the validation experiments to a collection of BSM log files and to the service selection module (see chapter 6); §5.9 contrasts our reduction methods with other proposed in the literature; finally, conclusions drawn up from our investigations appear in §5.10.

5.2 The use of N-grams as a Reduction Method

N-gram models are generally used to predict the next element in a sequence. To make this prediction, an n-gram model has to have a frequency analysis from which a probability of occurrence is extracted. This probability is then used to estimate the probability of occurrence of the next element in the sequence. Other methods can be used to predict the next element in a given sequence, e.g. HMMs.

N-grams can be used to identify the most repetitive subsequences in any sequence of elements. HMMs can also do this job. But the HMMs do not find an specific sequence, they find a family of subsequences, i.e. [Krogh et al., 1994, Hughey and Krogh, 1996]. If we were to use a method such as HMMs then we can substitute any subsequence described by the family for the same tag. This substitution allows for an attack subsequence to be substituted and regarded as a normal subsequence. By using n-gram models we guarantee a 1-to-1 substitution relation. That way, variations of an attack that pose as a normal sequence (mimicry attacks) are not substituted.

Another thing we request from an IDS is not to loose the capability to return to the original sequence. One reason for wanting the reduction process to be reversible is for forensic analysis. We must assume that after an intrusion is detected, the attacker already has done some harm to the system. Therefore, we will perform a forensic analysis on the data to follow all of the attacker steps. To follow the events we need the track of activity that lead to an insecure state of the system. When designing a security device it is necessary to assume that this device will interact with other devices. And the design of our methods is done to provide the most information to other devices in the security chain.

5.3 N-Grams Theory

N-gram theory comprises a collection of stochastic methods [Manning and Schütze, 1999]. If properly used, these methods can compute the probability that a sequence of symbols will occur in a larger, unseen sequence. Let an *n-gram* be a sequence of n symbols. In our case, each symbol denotes a system call. Then, according to the *n-gram language model*, the probability that symbol w_n will appear, given that the system call sequence $w_1, ..., w_{n-1}$ has already shown up is given by:

$$P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i | w_1, ..., w_{i-1})$$

The construction of an n-gram language model, henceforth called *language model* for short, is normally a three stage process [Young et al., 2002]. The first two steps are actually used for building the language model while the last one is used for validation purposes only. In the first step, each training sequence is analysed in order to extract every single n-gram and then count its occurrences. In the second step, the training sequence vocabulary is created and the language model is built using each n-gram count to estimate the associated probabilities. In the final step, the *goodness of the language model* is estimated by making it measure how many different symbols may follow a given sequence with equal probabilities. While computing this measure, called the ***perplexity of the language model***, an unseen test sequence set must be considered. The lower its perplexity, the better an language model characterises the test sequence set. This 3-stage process is iterated until the perplexity of the language model reaches an specified threshold level.

5.3.1 Language Complexity

Building an language model is an computationally expensive process. The number of all possible n-grams in a language grows exponentially on the size of n. More precisely, the number of possible n-grams for a vocabulary of size m equals $m^{n-1} \times (m-1)$. In actual situations, most of these n-grams will never show up, and so it is common practise to consider only the n-grams that arise in the training set.

Although the actual number of n-grams present in a training corpora is smaller than the theoretical number, almost all of the n-grams are used for probability calculation.

5.3.2 N-gram Probability Calculation

The most popular method for calculating the probability of n-gram occurrence is maximum likelihood estimation (MLE). MLE is based on a naïve frequency analysis:

$$P(w_1, ..., w_n) = \frac{C(w_1 ... w_n)}{N}$$

where N is the total number of instances used in training and $C(w_1 ... w_n)$ is the frequency associated to n-gram $w_1 ... w_n$. MLE, however, has a chief limitation: every sequence that does not appear in the training set will be assigned an occurrence probability equal to 0 and, what is more, will not be included in the language model.

Any method that gets around the limitation above mentioned is said to be a *discounting strategy*. Example discounting strategies include *add one*, based on Laplace's Law, *expected likelihood estimation*, based on Lidstone and Jeffreys-Perks's law, *held out estimation*, *cross-validation*, *absolute discounting*, *linear discounting* and *good-Turing estimation*. For information about these strategies, the reader is referred to Manning and Shütze's book [Manning and Schütze, 1999]. Throughout our experiments, we used

good-Turing estimation, (GT), since according to the reviewed literature GT greatly avoids the elimination of unseen sequences, without imposing a considerable penalty to an existing one. As described, MLE assigns a probability equal to 0 for unseen sequences. Therefore if a sequence is to be used it has to appear on the training corpora. GT avoids this situation by assigning a small probability to unseen sequences, and presents little computational overhead.

5.4 Data Selection for N-gram Extraction

We chose to use 5 log files out of the 35 available log files for the n-gram selection process. This number of log files form a representative sample, since according to [Martínez, 2003]:

$$n = \frac{N}{1 + Ne^2} \tag{5.1}$$

where n is the number of samples we need to take to represent a population of size N with a confidence $1 - e$ (or with an error tolerance of e). In our case we chose a confidence of 97.5% ($e = 2.5\%$). We have an average of 250 sessions in each log file. For a population of 8750 sessions we need around 1300 sessions as our sample. So we used 5 log files (close to 1250 sessions) as our representative sample. In order to take a representative sample for a year of log files (52 weeks, equivalent to 65000 sessions), considering the same 97.5% confidence, we would need 1500 sessions. Using 1250 sessions we have a confidence of 97.2% that this sample represents a year of data.

We only used sessions without attacks or anomalies. The rationale behind this is to select only n-grams that identify normal traffic and do not add noise to the intrusion detection process. The selected log files are the ones with less attack sessions, so after discarding attack or anomaly sessions we ended up with the highest possible number of sessions.

A BSM log file is composed of a set of sessions, each of which belongs to a given service. Moreover each session is formed by one or more processes. During our methodology we separated each of these sessions and concatenated all of its processes. Both the number of sessions and processes in each session are variable. By separating log files and ordering them by sessions, we extracted n-grams with high occurrence frequency. These n-grams belong to a single session without overlapping contiguous sessions. From now on, we will refer to these ordered sessions as the training sessions. The process of separating log files in sessions is done on-line. The rest of the steps are performed off-line.

We selected n-grams in two ways: i) extracting them from every available session; and ii) extracting them from service specific sessions. Each log file contains a finite number of sessions, each of which belongs to a given service. Some services have a more representative body among the log files. Services with lower number of sessions have n-grams that reduce that service greatly. Since these n-grams have lower frequency than the n-grams of most common services they might get lost in the process. That is why we assigned a priority that uses the number of sessions of a service instead of the total number sessions.

5.5 Session Folding Using N-gram Models

The log reduction is a two step procedure, the first step aims at selecting the n-grams with the highest repetition frequency and to attach their corresponding priority, we will call this process n-gram detection and tagging. The second step, is the actual reduction process which makes use of the tagged n-grams extracted in the first step for session folding. N-gram detection and tagging only needs to be performed once and can be done offline. Whereas the reduction is done in real time and for every session.

The following is a description of each step in the n-gram identification process: i) n-gram extraction; ii) n-gram frequency analysis and n-gram priority assignment. Later on we shall describe each step in the reduction process: i) n-gram comparison, aimed at avoiding overlapping; ii) subsequence substitution using a fresh tag.

Note that the n-gram identification process is not the same as the n-gram extraction process described in §5.3. Step 1 in the identification process is the same as language model creation process. Step 2 uses that language model frequency analysis and priority assignment.

5.5.1 N-gram Identification and Tagging

The first step toward session folding is to extract the n-grams with a higher occurrence frequency and occurrence probability, and then tag them with a priority. The priority is calculated using both a combination of services, and a service exclusive analysis.

N-gram Extraction

N-gram extraction consists of the application of a blind, exhaustive procedure. As a result, we obtained the n-grams that occur most frequently in the training sessions. Although in theory n-gram model creation should consider all possible n-grams, in practise only n-grams that exist within the training data are used. For example for a 10-gram with a vocabulary of 200 tokens, the possible number of sequences should be $200^{10} \times 199$. However, in our experiments, we found only 2291. N-gram extraction prunes all sequences with 0 occurrences. For the final language model a low probability is assigned to pruned sequences. In our calculations, we considered the n-grams that were pruned from the entire log file as well as those pruned from different services within that file.

N-gram Priority Assignment

Using the n-gram count and the language model, we identified the n-grams with a higher frequency or probability of occurrence. We considered n-grams of different sizes to find n-grams that provide a high reduction rate. If an n-gram is present in the training log files an occurrence frequency is assigned to it. If it is not present then the occurrence probability obtained in the language model is assigned to it.

Using either the n-gram occurrence frequency or probability, we estimated a reduction ratio later used as a priority Pr for every n-gram that was found. The priority is used to select which n-gram to use first in the reduction process. Not only does this ratio consider large n-grams with a high frequency, but it also considers the total number of system calls N on the training sequence sessions from which the n-grams were extracted. We calculated a reduction percentage for every selected n-gram, i.e. how much will a given n-gram reduce a log file.

If we use n-gram occurrence frequency f the n-gram priority Pr is calculated by:

$$Pr = \frac{n \times (f+1)}{N} \tag{5.2}$$

By contrast if we use n-gram occurrence probability P the n-gram priority Pr is calculated by:

$$Pr = P \times n \tag{5.3}$$

In both cases n is the size of the n-gram. Both equations provide a reduction ratio for the input n-gram. Using Pr we choose the sequences that provide a high reduction rate. Our n-gram selection criterion is divided in two: selection of n-grams with high Pr at log file level, and n-grams with high Pr at service level.

Service Exclusive N-gram Selection and Priority Assignment

It is possible that a certain n-gram has a high ratio, i.e. in an `smtp` session; but `smtp` sessions are a small segment of an entire log file. This is because some services in the training log files have a less representative body than other services. This situation might exclude n-grams with low reduction ratio at log file level from the reduction set, even though the reduction ratio within a given service is high. The same analysis presented for n-gram priority assignment is done at service level and it is our second criterion for n-gram selection. Assigning a priority to an n-gram is not sufficient to avoid overlapping.

5.5.2 Session Folding Using Tagged N-grams

N-grams tend to overlap with each other, they might intersect at some point. To avoid overlapping when making an n-gram substitution we used a priority queue approach to select the n-gram to substitute. The queue was used to substitute high ratio n-grams. We created a window of size equal to the largest n-gram in the queue. Once the window is full, we tested its content against all n-grams in the queue. The order of the priority queue is given by the ratio defined in equations (5.2) and (5.3). By substituting n-grams with higher ratio we guarantee that, even if there is an overlapping, only the n-grams that provide maximum reduction are used. Notice that by substituting an n-gram with a new symbol we are avoiding further substitution on that segment resulting in overlapping elimination. We avoid substitution because the newly added symbol is not present in any n-gram used in the substitution.

Log file reduction using n-grams is accomplished by n-gram substitution at session level. In n-gram model creation, overlapping is not considered, but only conditional probability. We need to take the concept even further and include Pr as a reduction measure. Whenever a high priority n-gram is found, it is replaced by a fresh symbol that substitutes the entire n-gram. The first time we use an n-gram, a new symbol is created and added to the symbol dictionary.

5.6 The Methodology in Action

We will provide evidence that as a result of applying our methodology, we obtained a large reduction ratio using a small number of n-grams. For the experiments reported in this chapter we used the CMU-Cambridge Statistical Language Modelling Toolkit [Young et al., 2002]. The toolkit provides a series of UNIX tools that facilitate language modelling. Each of the steps required for n-gram substitution and log file reduction is described below.

5.6.1 N-gram Extraction

By using the CMU toolkit we extracted the n-grams and the associated frequency. The analysis was made for each log file and also for each service. The results of such an analysis are presented in histograms shown in figures 5.3 (all services histograms), 5.4, and 5.5 (`telnet` histograms), and 5.6, and 5.6 (`smtp` histograms). Here, histogram's axises are shown as a right hand coordinate system. The x axis represents the n-grams size. The z axis represents the frequency f for that given n-gram. The y axis is the number of different n-grams of size n with frequency f.

The histograms are used to analyse the amount of n-grams that have a number of repetitions similar to a multiple of the number of different sessions included in the training data. If there are m sessions and the frequency of an n-gram is a multiple of m, then that n-gram is more likely to be common among every session. That is, we prefer n-grams that are repeated in a large number of sessions over n-grams that repeated many times in a couple of sessions. The former are more general n-grams and therefore provide a better reduction ratio for unseen sessions. The same histograms can be used to know in advance, how many n-grams to look for when making the frequency based selection.

In figures 5.3, 5.4, 5.5, 5.6, and 5.7 only n-grams of sizes between 2 to 50 are shown. In figures 5.8, and 5.9, we show the corresponding histograms to days 3 and 4 corresponding to n-grams of sizes 51 to 100. The reader can appreciate that there are n-grams of bigger sizes that still have a high occurrence frequency. Nevertheless, the number of n-grams with size above 50 and high frequencies is a lot smaller than the number of n-grams of size below 51. So we concentrate our n-gram extraction in n-grams of size smaller than 51.

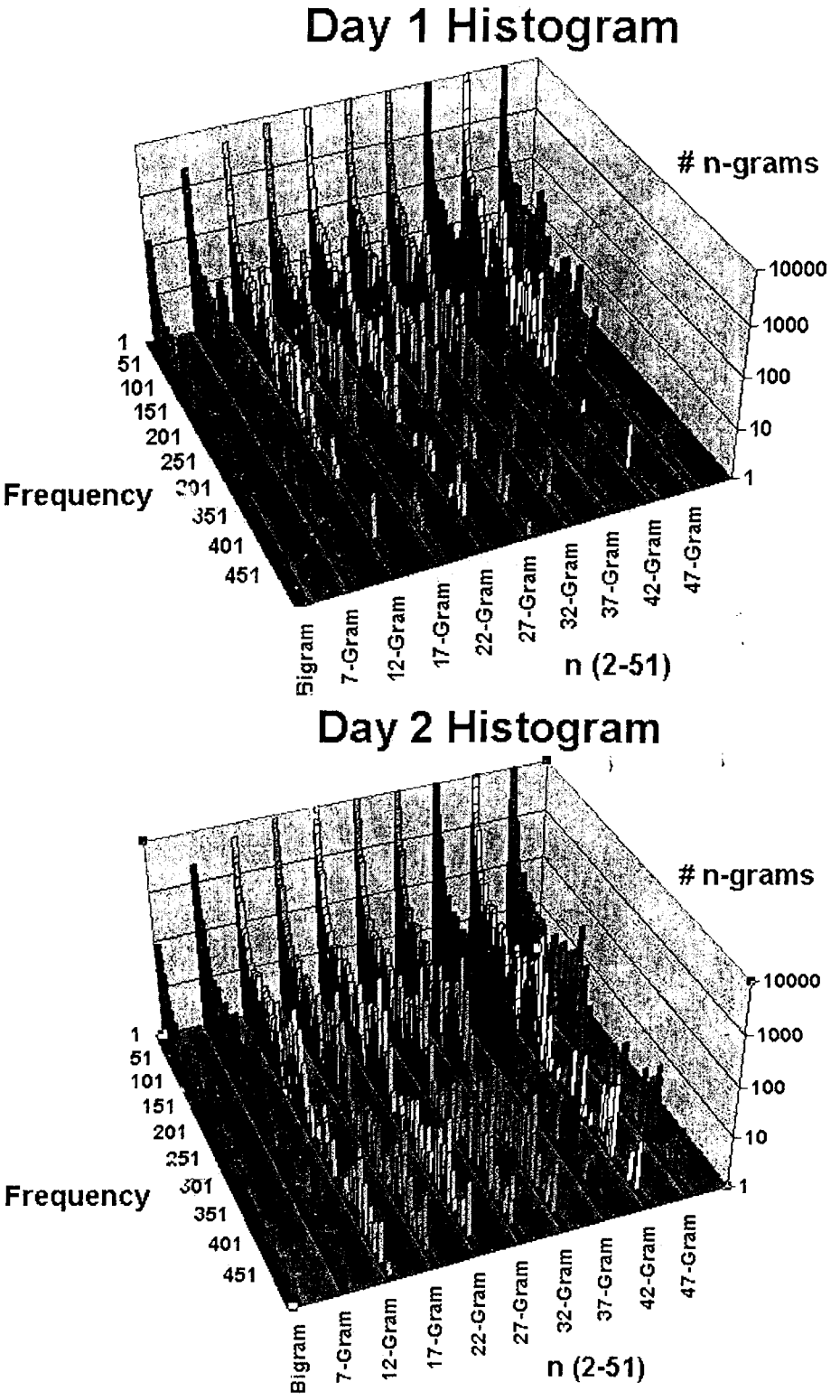


Figure 5.3: Mixed Services Histograms

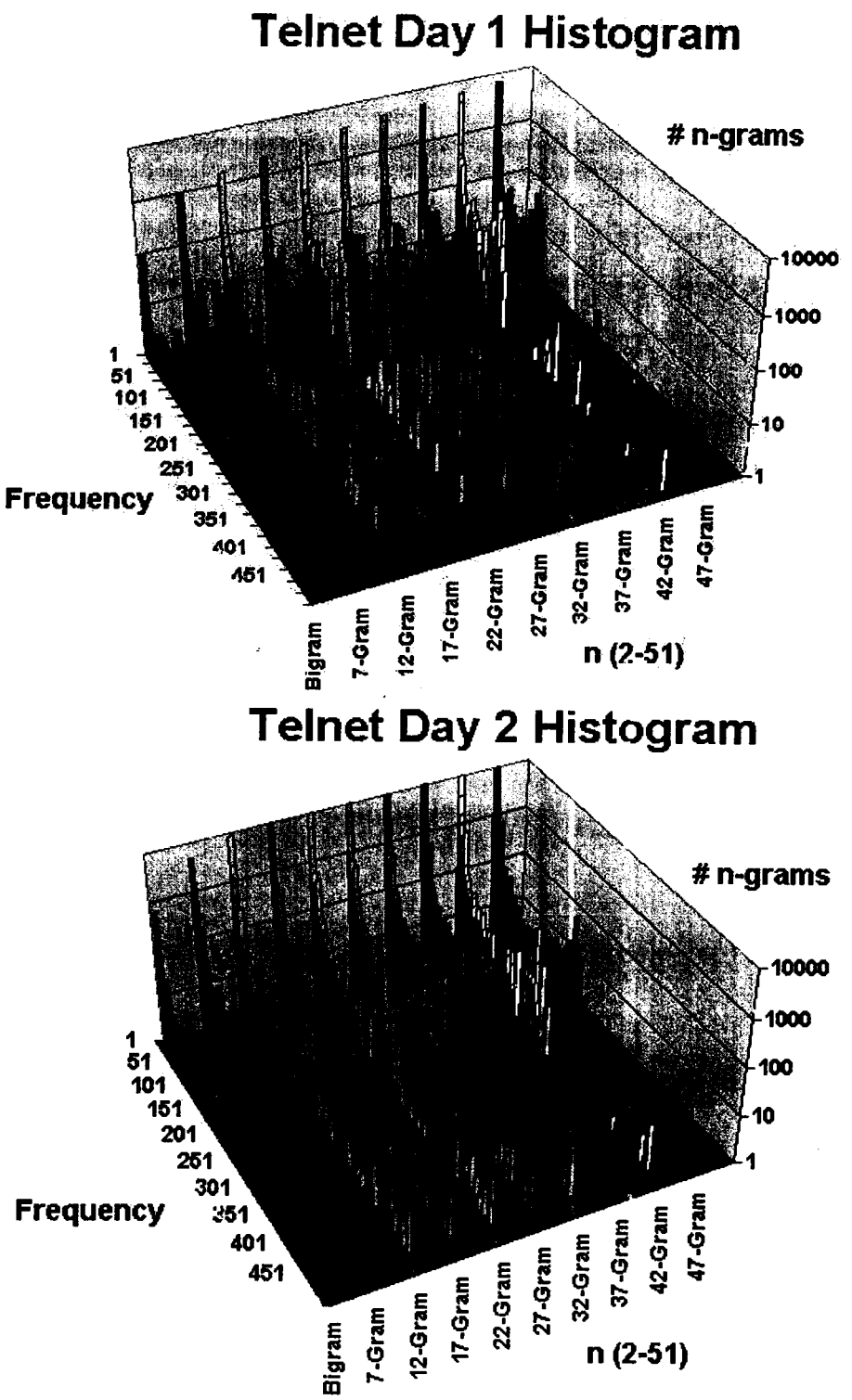


Figure 5.4: Telnet Histograms (a)

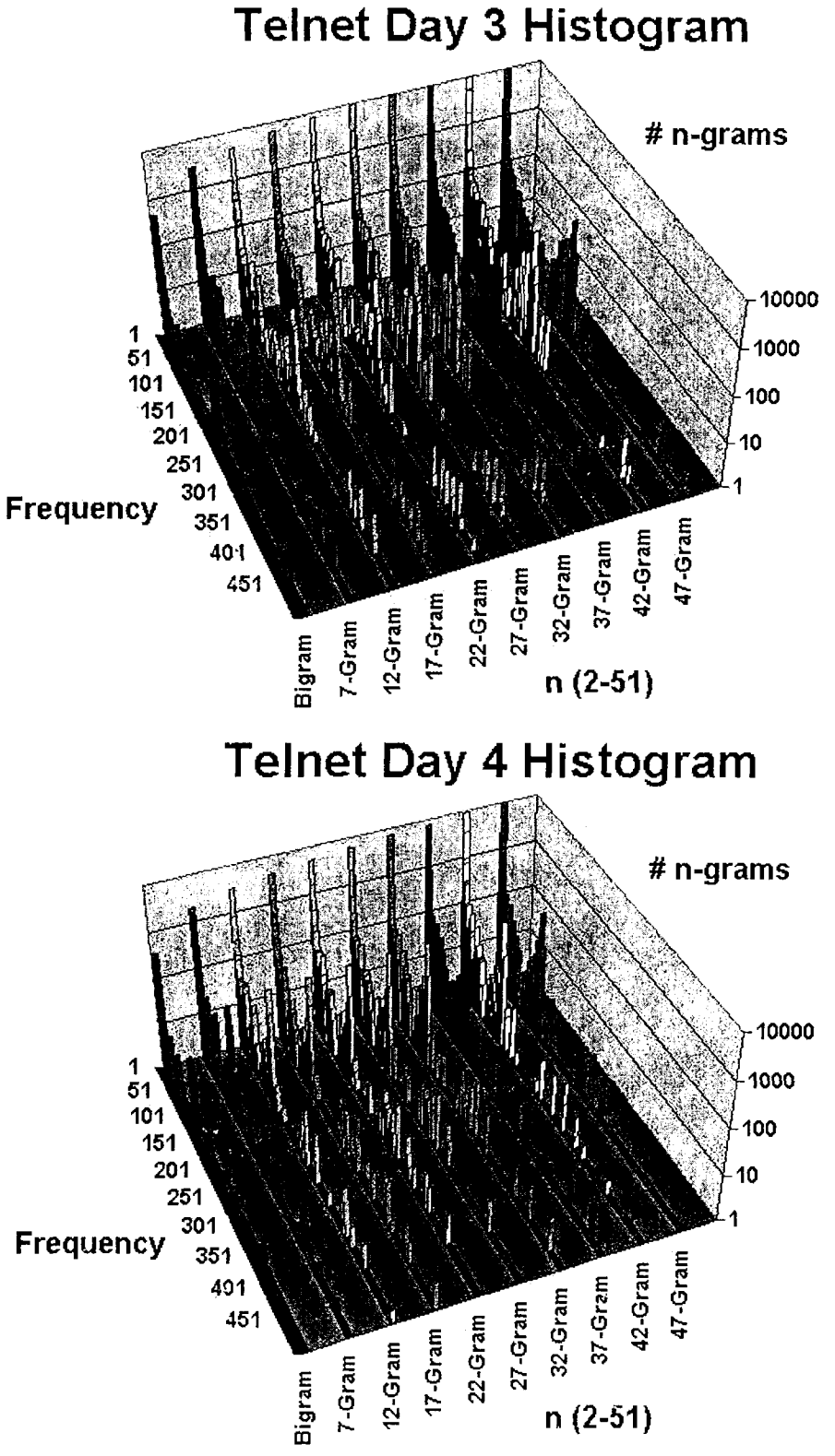


Figure 5.5: Telnet Histograms (b)

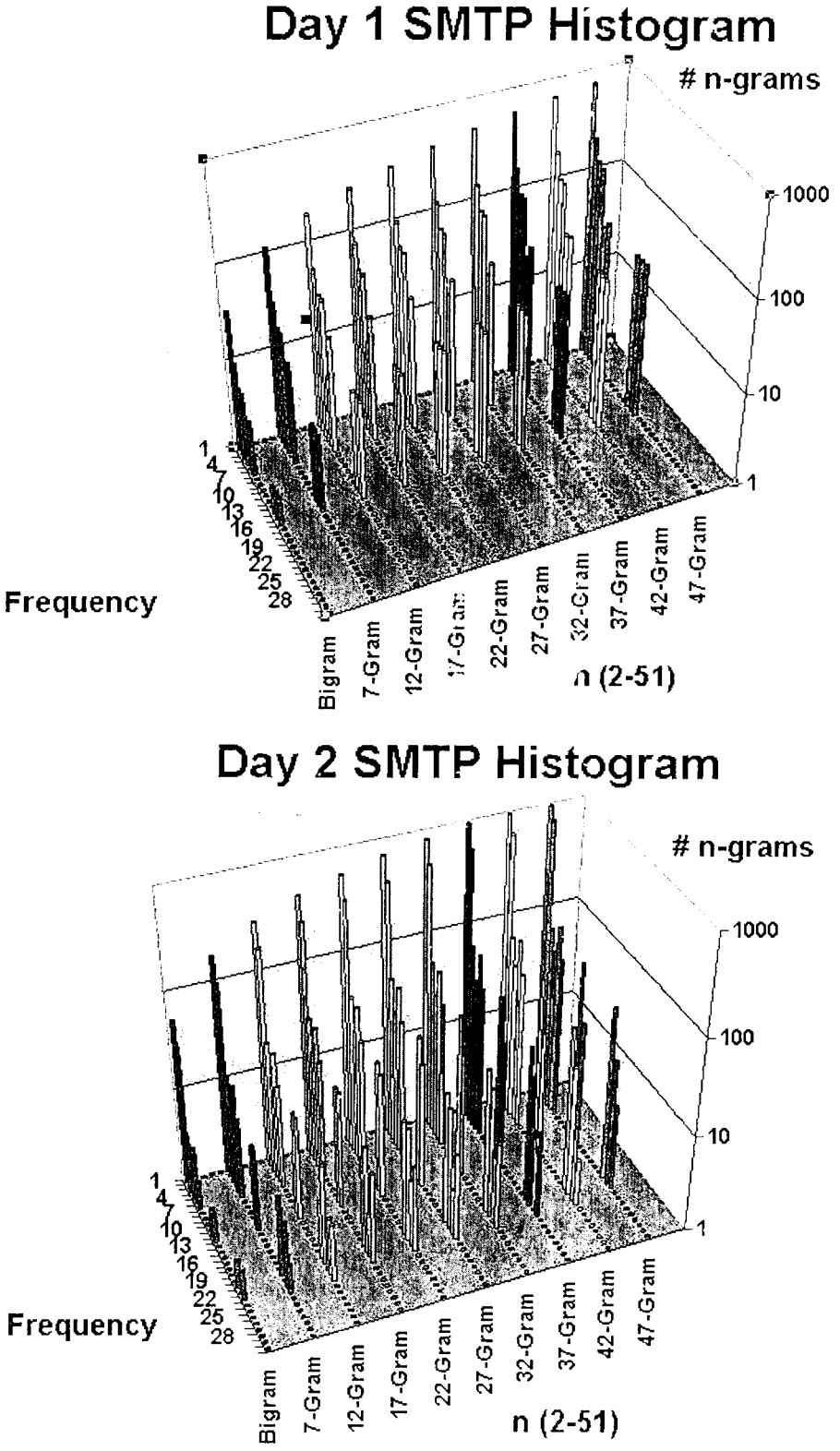


Figure 5.6: `Smtp` Histograms (a)

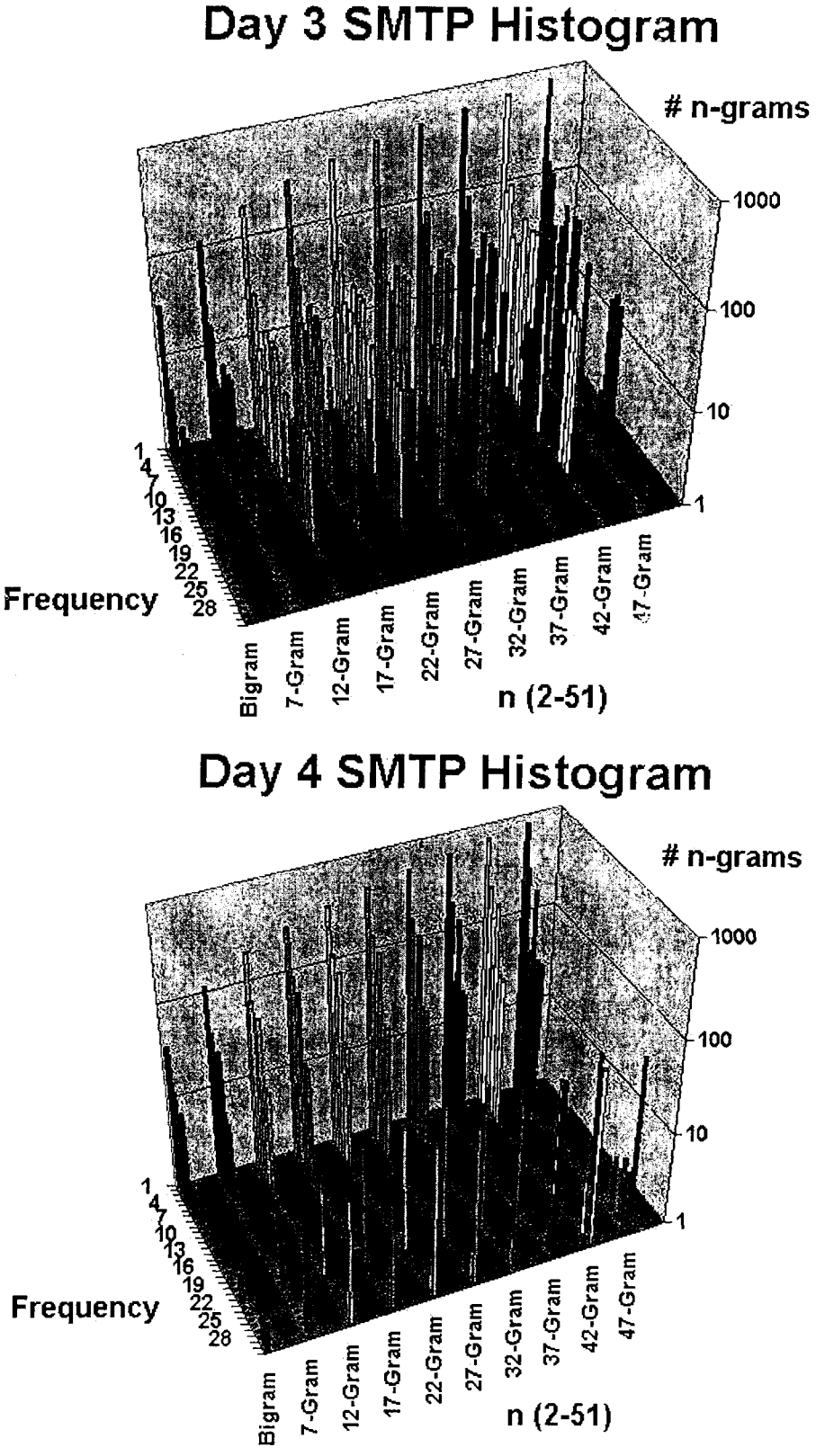
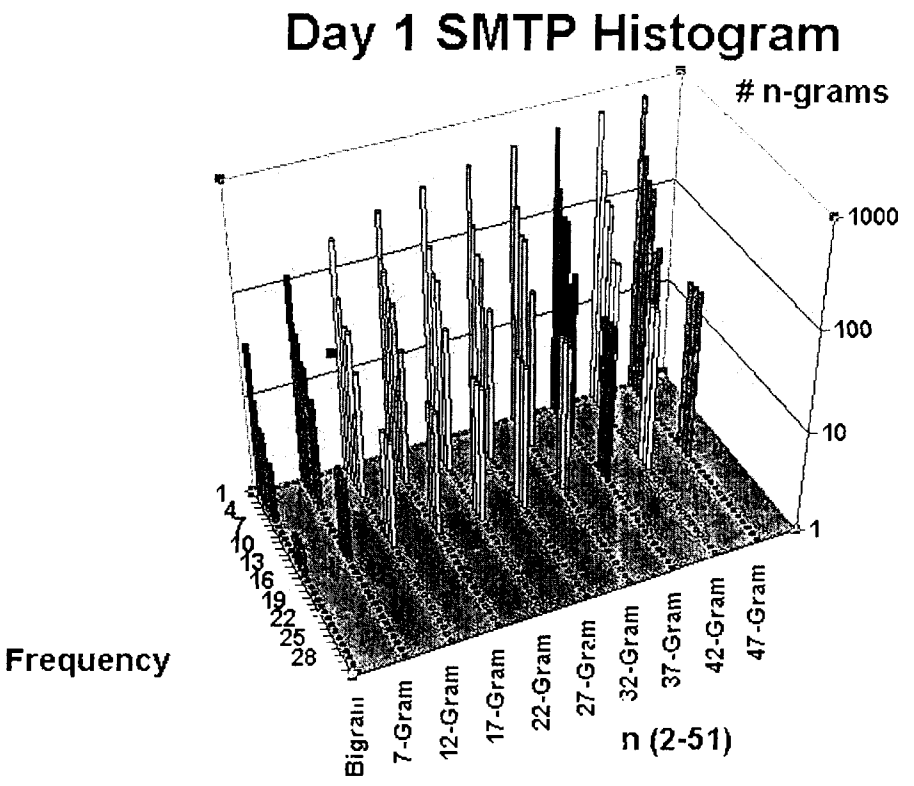


Figure 5.7: `Smtp` Histograms (b)



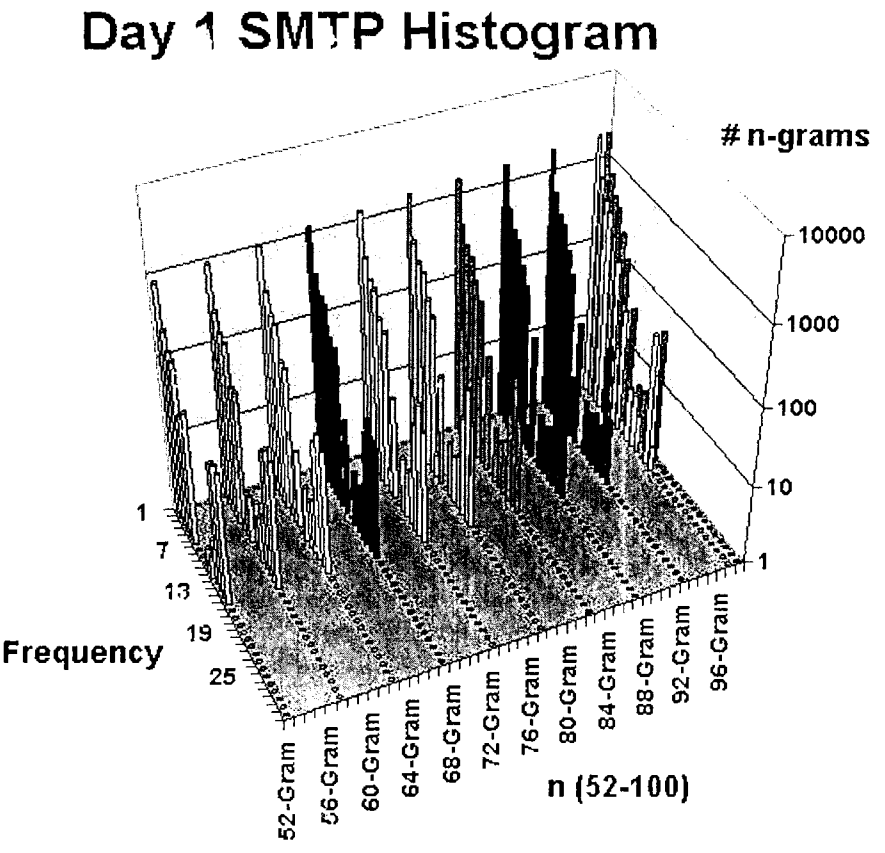


Figure 5.8: `Smtp` Histograms, N-gram Size > 50 (a)

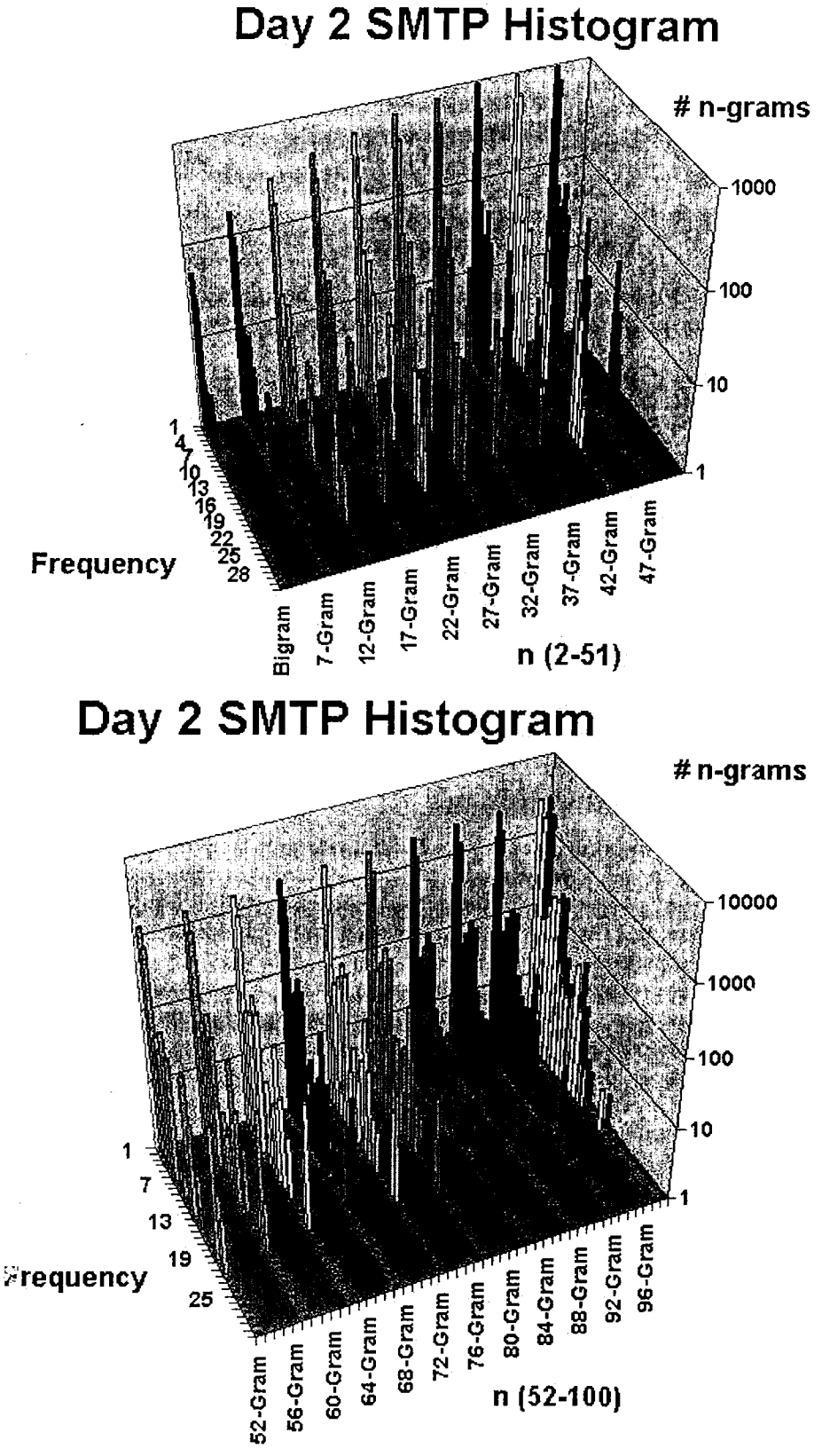


Figure 5.9: `Smtp` Histograms, N-gram Size > 50 (b)

unlink(2)||success ioctl(2)||failure:_Invalid_argument ioctl(2)||failure:_Invalid_argument	-0.0011
sysinfo(2)||success close(2)||success open(2)_-_read||success	-0.0025
open(2)_-_read||success old_setgid(2)||success open(2)_-_read||success	-0.0010

Figure 5.10: Sample tri-grams in a language model file. The last column is the logarithm of the conditional probability of such n-gram

1:mmap(2)||success	11:mmap(2)||success
2:close(2)||success	12:munmap(2)||success
3:open(2)_-_read||success	13:mmap(2)||success
4:mmap(2)||success	14:close(2)||success
5:mmap(2)||success	15:open(2)_-_read||success
6:munmap(2)||success	16:mmap(2)||success
7:mmap(2)||success	17:mmap(2)||success
8:close(2)||success	18:munmap(2)||success
9:open(2)_-_read||success	19:mmap(2)||success
10:mmap(2)||success	20:close(2)||success
$Pr_d = (2676 + 1) * 20/471668$	

Figure 5.11: Sample n-gram of size 20 with $Pr_d = 0.1135$

5.6. N-gram Tagging

Based on the occurrence frequency analysis we identified the n-grams with a desired number of repetitions. This analysis makes the extraction of such n-grams much easier and faster. From the extraction we chose 100 n-grams for the reduction. These n-grams are mostly the ones whose occurrence frequency is close to a multiple of the number of sessions. In figure 5.11 we can observe such an n-gram with its associated Pr value. Also, based on the language model probabilities we selected about 50 n-grams with a probability of occurrence above 98%, in figure 5.10 we show 3 tri-grams with such probabilities.

We also extracted 50 different n-grams using the analysis over separate services. Selected n-grams have an occurrence frequency similar to a multiple of the number of sessions for that service. This means that such n-grams are common to many sessions of that service. All these n-grams have an associated occurrence count, and the total number of system calls present in the original log file. These n-grams are shown in detail in our website[1]. Along with the size of the n-grams, the number of system calls will define the reduction ratio. The

[1] http://webdia.cem.itesm.mx/ac/raulm/ids

Log File Size	Frequency Extraction Time	Language Model Extraction Time	Histogram Generation Time
744,085	00:13:35	01:48:10	00:01:32
2,114,283	00:40:10	06:02:35	00:05:25
1,093,140	00:25:12	03:42:27	00:03:21
1,121,967	00:27:47	03:54:31	00:03:27
1,095,935	00:25:52	03:45:53	00:03:22

Table 5.1: Times needed for the different n-gram extraction process.

number of system calls will vary according to the day as we can appreciate in figure 5.4. That figure shows histograms for five days of telnet sessions. After extracting n-grams for every log file (both all sessions and service separated) a language model is generated.

We need to explain the selection of the *discounting strategy*. The one provided with the software we used is the good-Turing estimator. As described in §5:3, it is a great method to avoid elimination of unseen n-grams without imposing considerable probability reduction of existing n-grams.

The main overhead of using n-gram language models for reduction is the space required to hold a language model file. For a log file with 800,000 objects of size 17Mb, a file of 5Mb is needed to hold n-gram occurrence frequency and about 1Gb to hold the language model. The language model and an n-gram occurrence frequency file are used to extract key n-grams. Fortunately, the language model and the frequency file are temporal, as they are eliminated after key n-gram extraction. The time needes for the whole n-gram selection process is shown in table 5.1. The format for the language model is described in appendix B.

After selecting n-grams with high occurrence frequency or probability we calculated the reduction ratio. The reduction ratio will sort selected n-grams in a priority queue for subsequent replacement of such n-grams in the log files. Using the priority queue we substitute, or fold, a given session. Prior to the substitution we load any abbreviation dictionary that we have previously used in order to avoid repetitive abbreviations in sub-sequent reductions. The priority queue is then used to avoid overlapping.

5.6.3 Session Folding

The n-gram set used in the folding process and the dictionary are available in our website[2].With the use of the dictionary we selected the next abbreviation that will be used in the folding process. An abbreviation is only generated when an n-gram is used in a substitution, not all n-grams will be assigned an abbreviation. For example from the 100 n-grams selected based on occurrence frequency, only 11 were really used. From the 50 selected based on occurrence probability, only 5 were used and from the 50 selected from each service only

[2]http://webdia.cem.itesm.mx/ac/raulm/ids

3 were used. This means that about 89% to 93% of the selected n-grams were overlapping. Selected n-grams do not intersect, so after the analysis, subsequent reductions did not consider a priority.

Even by using only 9% of the selected n-grams, we obtained a reduction of 65% in the worst case and a reduction of 82% in the best case. We obtained an average reduction of 74%. This reduction was tested using the n-gram set generated from the first 5 BSM log files to reduce another set of 5 BSM log files. The result is a set of 19 n-grams that provide an average reduction rate of 74%.

5.7 Validation Experiments and Data Selection

To test the reduction capabilities of our method we chose to reduce log files from the 1998 and 1999 DARPA repositories using extracted n-grams from the 1998 repository. By using log files of different years we aim at showing the generality of our reduction method. As we will see, the results are pretty similar between reductions for each year. This is a proof that changes in user activity is not a critical factor for our reduction method. We specifically used the 5 log files used for n-gram selection plus another 5 logs from 1998 and 5 more from 1999. This is a total of 15 log files out of 40. As described in §5.4, this gives us a confidence of 98.8%. The data used in the validation process includes attack and anomaly sessions. The validation procedure is straight-forward, we use the extracted and tagged n-grams, and then apply the reduction methodology to avoid overlapping.

Nonetheless, we still need to prove that our reduction method keeps the information necessary to discern between two events regardless the reduction. To prove this we chose to use the same methodology as described in chapter 6 and to compare the results of using folded sessions against the results of unfolded sessions. We also use this comparison to show the HMM training time difference between folded and unfolded sessions. Another test is presented in chapter 7, where we compare the results of using our IDS with mimicry attacks over folded and unfolded sessions.

5.8 Validation Results

Extracted n-grams provide an average reduction of 74% within the training sessions. We also used the n-grams to reduce unseen sessions from 5 different log files from the 1998 repository, and 5 from 1999. As input we have an unseen log file and as output we provide the reduced log file. In tables 5.2, and 5.3, we show the reduction ratio over the validation data of 1998 and 1999 respectively (we will only show results for unseen data). The table columns are: log file ID, original number of system calls, compressed number of system calls, and number of n-grams used in the reduction. Last row of the table shows the results of applying the reduction to a file with only `telnet` sessions.

By training the HMMs we can validate the impact of our methodology in

Table 5.2: Validation Results, 1998 Log Files

Log File ID	Original Object #	Compressed Object #	Reduction %	Used N-grams
1	776,000	270,000	65.3%	7
2	1,800,000	486,000	73%	12
3	1,150,000	344,000	70.1%	5
4	801,000	175,000	78.2%	9
5	1,158,000	392,000	66.2%	5
`telnet`	209,000	48,000	77.1%	5

Table 5.3: Validation Results, 1999 Log Files

Log File	Original Object #	Compressed Object #	Reduction %	Used N-grams
1	820,855	248,719	69.7%	9
2	490,896	142,360	71%	8
3	630,457	198,594	68.5%	7
4	520,358	139,456	73.2%	11
5	220,658	52,296	76.3%	13

training times. As we expected training time is significantly reduced by using folded sessions. The reason for time reduction is that the order of the HMMs training algorithm is $O(n^2 l)$, where n is the number of states in the HMM, and l is length of the training sequence. In figures 5.12 and 5.13 we show training times for the HMMs presented in chapter 6. In each figure, training times for unfolded sessions and its folded counterpart are contrasted. Also in table 5.4 we show the comparative results for service selection using folded and unfolded sessions. The first column in the table indicates the service discriminator used. The first row indicates to which service a session belongs to. The table should be read as: the percentage of sessions of service n (first row) classified as service m (first column). The first percentage of each cell corresponds to unfolded sessions and the second result corresponds to folded sessions. The service selection process is explained in more detail in chapter 6. We can see from this table that not only does session folding keeps discernibility information, but it also reduces the number of false positives. The false positive reduction for intrusion detection will be explored in chapter 7.

Table 5.4: % of Correct Service Discrimination, Folded vs. Unfolded Sessions

HMMs	`telnet` headers	`smtp` headers	`ftp` headers	`finger` headers
`telnet`	100% vs. 100%	2% vs. 1.8%	1% vs. 1%	0% vs. 0%
`smtp`	1% vs. 0.8%	100% vs. 100%	2% vs. 1.6%	0% vs. 0%
`ftp`	0% vs. 0%	1% vs. 0.7%	100% vs. 100%	0% vs. 0%
`finger`	0% vs. 0%	0% vs. 0%	0% vs. 0%	100% vs. 100%

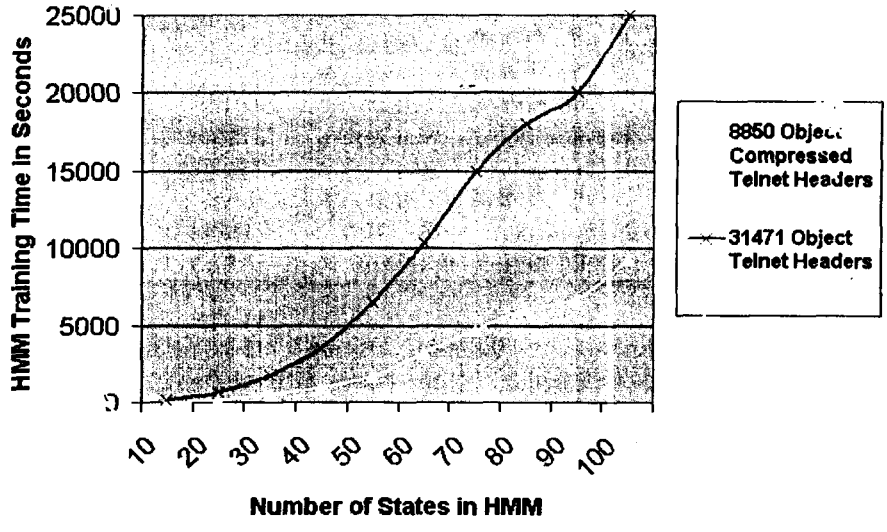


Figure 5.12: Telnet Service HMM Training Times, Unfolded Sessions vs. Folded Sessions

5.9 Session Length Reduction Methods

N-gram theory has been largely used in the context of natural language analysis, it also has been used in anomaly detection by Maxion and Tan [Maxion and Tan, 2002, Maxion and Tan, 2000], Marceau [Marceau, 2000], Wespi [Wespi et al., 1999], and Forrest et al. [Forrest et al., 1996]. All these papers present ways of using n-grams for anomaly detection but not for log file reduction. Therefore comparing our reduction method with these methods is out of the scope of this section. Log file reduction methods are presented below.

Marin et. al. [Marin et al., 2001] have suggested to use an expert system for log file reduction. Using a series of fuzzy rules the expert system discriminates objects that most frequently occur, using a series of fuzzy rules. Next it clusters the discriminated objects to form a small number of object classes. Even though its reduction factor is very high, this method is prone to a large false-negative intrusion detection rate. This is because it prunes out key information and is restricted to a 1-gram only. By contrast, we do not get rid of non-commonly used system calls and the scope of our reduction analysis is considerably larger. As we showed, this contributes to keep false-negative intrusion detection rate low.

Lane and Brodley have also addressed the problem of log file reduction [Lane and Brodley, 1999, Lane and Brodley, 2000]. They have proposed two main clustering methods, one based on K-centres and the other on greedy clustering. By applying both methods, each cluster contains a collection of objects sequences, which is then collapsed into only two objects: the cluster centre

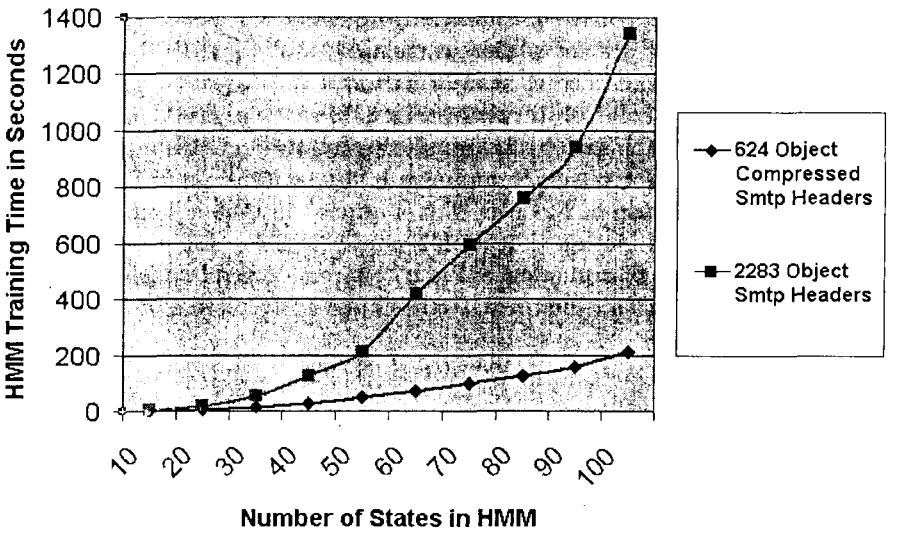


Figure 5.13: Smtp Service HMM Training Times, Unfolded Sessions vs. Folded Sessions

and the associated mean. The other object sequences are simply disregarded. Lane and Brodley's methods may yield a huge reduction ratio (e.g. 500 different sequences might be shrunk to only 30 ones or even fewer); however, eager sequence elimination inevitably leads to poor or incomplete profiles and therefore to an increase in the false-alarm detection rate. By comparison, even though our technique yields a lower reduction ratio, it does not have a negative impact on the false positive ratio.

Besides from the two methods above mentioned, Lane and Brodley have also explored two heuristic pruning techniques: least recently used (LRU) and least frequently used (LFU). In both techniques the reduction ratio is defined *a priori* and hence a predetermined number of sequences ought to be eliminated from the input session. Both techniques will of course produce a reduction ratio as high as indicated, but at the expense of losing of chief information. Lane and Brodley report on an increment in the false-positive and false-negative ratio (20% and 16% respectively). By comparison, even though our technique yields a lower reduction ratio, it does not have a negative impact on the false positive ratio.

Knop et. al. [Knop et al., 2001] have addressed a similar problem, but does not consider log file reduction. Rather than shrinking an input session, Knop et. al.'s method simplifies its content by calling similar objects with the same name. Let $\Sigma(seq)$ denote the alphabet of symbol sequence seq, then given a sequence s, the method will return a new sequence, s' such that length(s') = length(s) but that $|\Sigma(s')| \ll |\Sigma(s)|$. The method works as follows: it first correlates objects using a method similar to principal component

analysis [Rencher, 1995]. This way, two objects will be assigned a coefficient equal to 1 if they are completely correlated, and equal to 0 otherwise. Then using this correlation coefficient information, a K-Nearest Neighbours algorithm is applied for achieving object clustering. Object reduction amounts to selecting one object from a cluster discarding the rest of them. Once again, the alphabet simplification factor is very impressive but so is the loss of information. One important limitation of Knop et al.'s approach is that it is in general difficult to apply in the intrusion detection context. This is because the method requires some kind of numerical value to express correlation between objects to be applicable. This is unnatural, since almost every piece of data in sessions are strings.

5.10 Conclusions

Based on our results we conclude that we successfully reduced the number of objects in a log file with nearly no impact for intrusion detection. By identifying a small number of key n-grams we reduced BSM log files by a factor of 4. The number of key n-grams is small enough not to increase considerably the vocabulary of system calls. An increase in the vocabulary would impact on the training time of a method such as HMM. Our method allowed us to find a small number of n-grams that provide a large reduction. This reduction ratio is comparable to the ones proposed by rival techniques and even better in most cases. Moreover, our reduction method is capable of returning to the original set of system calls. By contrast, rival techniques are incapable of reverting the reduction process.

We also trained some HMMs with unfolded sessions and with folded session, and the difference in training times between them is a considerable improvement. That way we proved that using longer sequences to train HMMs is convenient. In chapter 7 we demonstrate how folded sessions are equally useful, in some cases better than unfolded sessions for intrusion detection.

Chapter 6

Service Selection

6.1 Problem Description

Usually IDSs keep the information used to make a detection in attack data bases for misuse detection, or profiles for anomaly detection. When looking for an attack, it is common that the IDS compares the session of interest (or packet in network IDSs) against the entire attack base or against the whole profile. This is why, even with compacted information, an IDS still analyses information that does not contribute to perform intrusion detection.

Generally speaking, attacks (for misuse IDSs) or profiles (for anomaly IDSs) are service oriented. Almost all `telnet` attacks will not have any effect in an `ftp` session. Similarly a profile extracted from `smtp` sessions will be very different from a profile for the same user extracted from `ssh` sessions.

There are IDSs, like Snort [Free Software Foundation, 2002], that can be configured to monitor a specific service. But these IDSs lack flexibility since they are dependant on the configuration of each service. The detection is usually tied to the port where the service is supposed to run. If we want to monitor more services, then the configuration for each new service is included in the IDS. If configuration for one service changes all the IDS needs to be adjusted.

We need an IDS with the following three characteristics: i) flexibility; ii) efficiency; and iii) scalability. These characteristics have been of interest for an IDS since Denning's paper [Denning and Neumann, 1985], and have been pointed out by Forrest et al. [Hofmeyr et al., 1997], Zamboni et al. [Balasubramaniyan et al., 1998] and Mell et al. [Mell and McLarnon, 1999, Jansen et al., 1999].

We believe that by separating intrusion detection as a series of highly specialised sensors, an IDS will be more flexible (adapt quickly to any system configuration), scalable (accept new services on demand), and efficient. With this degree of specialisation, intrusion detection can be focused on a specific service, and therefore test for fewer attacks than an all purpose IDS, or in the case of anomaly detection, it will compare against fewer profiles (only profiles

that are related to that specific service).

For services using port numbers below 1024 there is no need for a service discriminator, but for services not using reserved port numbers it is very useful to know in advance which kind of service we are monitoring. Moreover, if we know to which service belongs the session we are monitoring, we know which attacks we should check for (for misuse intrusion detection), or which profiles to use (for anomaly intrusion detection). Most of the current IDSs begin monitoring when a new process is started. They monitor everything in between a `fork` and an `exit` system call. By successfully discriminating the service that is being used, we begin monitoring even before the session process is started knowing which type of communication we have to expect.

The problem with this approach is to build a method which takes as input a stream of system calls (with sessions of many services) and separates this stream according to the service each session belongs to. The service of a session can be identified by the header of the session. But the headers have variations from one session to another. Thus, service classification is not a trivial problem.

The remainder of this chapter is organised as follows: §6.2 pinpoints the advantages of having a service discrimination module and describes why we chose HMMs as our discrimination method; §6.3 gives an introduction into HMMs theory; §6.4 is a brief description of how we selected the training data for our experiments, the training data is used to extract an HMM model; while §6.5 describes the methodology used to train all the HMMs. §6.6 is a summary of the validation data selection process and validation experiments; §6.7 presents the validation results; and §6.8 presents conclusions drawn from our experiments.

6.2 The Use of HMMs for Service Selection

Even though methods like neural networks and support vector machines can be used to classify similar events, they have a fixed number of inputs, so a variable length input as a session header is not suitable for these methods.

Each service such as `telnet`, `ftp`, and `smtp` has a distinctive session header. Two headers of the same service might differ depending on a number of factors. For example, headers include information about previous failed login attempts or different server states. The last 22 elements of two different `telnet` headers before the `fork` call can be seen in figure 6.2. The first header is from a non-incidental session. The second header is from a session executed right after a failed login attempt. Considering these differences we conclude that a straightforward string matching method is not suitable for successful service selection.

If we consider each session header as an n-gram, variations in the headers will produce different n-grams. With the use of a hidden Markov model (HMM) we group such n-grams as a family. HMMs are good at categorising multiple sequences as families. With todays computer power, parsing time for a sequence of moderate size using an HMM does not have a negative impact on the performance. However, HMM training is still a process that demands a lot

Session Element	Telnet Header 1	Telnet Header 2
1	close(2)||success	open(2)_-_read||success
2	chmod(2)||success	ioctl(2)||failure
3	chown(2)||success	close(2)||success
4	open(2)_-_read||success	open(2)_-_read||success
5	ioctl(2)||failure	ioctl(2)||failure
6	close(2)||success	close(2)||success
7	close(2)||success	close(2)||success
8	open(2)_-_read||success	open(2)_-_read||success
9	ioctl(2)||failure	ioctl(2)||failure
10	close(2)||success	close(2)||success
11	setaudit(2)||success	setaudit(2)||success
12	open(2)_-_read||success	open(2)_-_read||success
13	ioctl(2)||failure	ioctl(2)||failure
14	close(2)||success	close(2)||success
15	open(2)_-_read||success	open(2)_-_read||success
16	ioctl(2)||failure	ioctl(2)||failure
17	close(2)||success	close(2)||success
18	open(2)_-_read||success	open(2)_-_read||success
19	ioctl(2)||failure	ioctl(2)||failure
20	close(2)||success	close(2)||success
21	login_-_telnet||success	login_-_telnet||success
22	audit(2)||success	audit(2)||success
23	fork(2)||success	fork(2)||success

Figure 6.1: Difference between elements 1, 2, and 3 of two telnet headers

of computational resources as time and memory.

Stochastic context-free grammars (SCFGs) are another powerful method for classification. But the training and parsing times are still prohibitively expensive. Since header sequences are automatically generated, they can be modelled with a regular language which is provided by the HMMs, there is no need to use methods like as an SCFG, that would model header sequences as a context-free languages.

Clustering methods can also be used for service discrimination by creating a cluster for each service. But they lack the scalability we still obtain using HMMs. If a new service is added, the metric for each cluster should be recalculated, since the new service will create a new cluster and affect the already calculated clusters. And there is the problem of attribute representation. For clustering to work, a metric has to be used to calculate the distance of an object to the centroid of each cluster. How to transform attributes from system calls to numbers to calculate this distance is not an easy problem to solve and if the wrong transformation is used can lead to poor classification.

6.3 Hidden Markov Models

As defined in [Brown, 1999], HMMs are probabilistic generative models that output strings by moving through a discrete state space using Markov decisions indexed by time. At each point in time, t, the current state, π_t, generates symbols according to some probabilistic rule. The HMM has states, at each of which a symbol is generated, and arcs, each of which allows a transition into another state. There are two types of probability parameters, state transition probabilities and symbol emission probabilities, which are attached to arcs and states, respectively. The symbol emission probabilities determine which symbol is more likely to be produced from every state in the HMM. The state transition probabilities determine which state of the HMM to move depending on the current state. This state transition is important because each state has different emission probabilities, so the emitted symbols depend on the state transitions.

As pointed out in [Manning and Schütze, 1999], an HMM is useful when the model under consideration has internal events that generate an external (that is, observable) event in a probabilistic manner. An HMM can be used to probabilistically characterise whether or not a given event is member of a family of other events (a set of related events is called a family).

In an HMM one or more starting and final states are specified. Possible transitions are made successively from a starting state to a final state, and the relevant transition probability and symbol output probability can be multiplied at each transition to calculate the overall likelihood of the sequence of 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 has a specific likelihood. 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. According to [Rabiner, 1989], an HMM is specified as a five-tuple $< N, M, \prod, A, B >$ and it is common practise to call it the model λ:

- $N = \{s_1, \ldots, s_N\}$: the finite number of states in the model.
- $M = \{k_1, \ldots, k_M\} = \{1, \ldots, M\}$: the alphabet or set of all the output symbols.
- $\prod = \{\pi_1\}$: the initial probabilities of all states (probability at time t_0).
- $A = (a_{ij})_{i,j \in N} = (P(\pi_t = j | \pi_{t-1} = i))_{i,j \in N}$: the Markov state transition probability distribution. a_{ij} gives the probability of moving from state i at time $(t-1)$ to state j at time t $(i = 1, ..., N, j = 1, ..., N)$. A is a matrix of size N by N.
- $B = (b_i(c))_{i \in N, c \in M} = (P(c | \pi_t = i))_{i \in N, c \in M}$: the probability that symbol c is emitted by state i at time t. This is called the symbol emission probability matrix and is of size N by M.

6.3.1 Three Fundamental Questions of HMMs

There are three fundamental questions about HMMs:

1. How likely is a given observation sequence O given a model λ? $(P(O|\lambda))$. This is also known as parsing sequence O with model λ.
2. What was the state sequence that generated a given an observation sequence O?
3. Given an observation sequence O and a set of states, try to find the parameters $\prod$, A, and B. This is also known as training the HMM.

These questions are called evaluation, decoding, and learning respectively. To answer the first question the so-called *Forward-Backward* procedure is used. To solve the second one *Viterbi* algorithm is used. And for the third one *Baum-Welch* algorithm is the most widely used. The mathematics for these algorithms is beyond the scope of this document but if the reader is interested [Rabiner, 1989, Manning and Schütze, 1999] are definite reference materials. For the purpose of this dissertation we only need to know the algorithms used by the HMM tools and not how they are implemented. All along this dissertation we are interested in the first and third questions. Using observation sequences we want to calculate the HMM parameters. Then, we want to use calculate how likely is a sequence given a the HMM defined by the parameters we just calculated.

6.3.2 Common HMM Architectures

The architecture of an HMM refers to the number of states in the model and how the states are connected. The type of connection of the states is also referred as the HMM *topology*. There are two main topologies for HMMs: left-to-right architecture; and ergodic.

Left-to-right HMMs

The start state of the HMM is the leftmost state and state transitions can only be made to the same state or to a state to the right. In this architecture the parameter $\prod$ has a probability of 1 for the start state, in figure 6.2 the start state is 1 (the HMM always start on the same state). This topology is widely used in speech recognition and facial recognition [Cohen et al., 2003]. Variations of these topology include left-to-right skipping one state or left-to-right skipping many states. The state skipping versions are good for accepting noise and not emitting any output symbol.

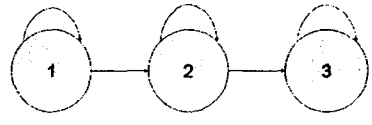


Figure 6.2: HMM with left-to-right topology

Ergodic HMMs

Any state in an ergodic HMM can be the start state. The start state is defined by $\prod$. In this architecture all states are fully connected as can be seen in figure 6.3. An ergodic model is more flexible than the left-to-right model. However with sufficient amount of training data an ergodic model will reduce to a left-to-right model. Ergodic models are good when it is not known the exact number of states of the problem of interest.

Meaning of HMMs States

Deciding the right number of states for an HMM and giving them a meaning is one of the hardest problems in HMMs. Usually HMMs are used when there is

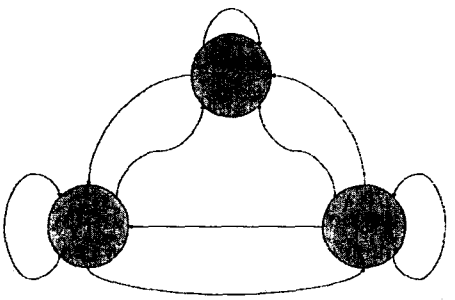


Figure 6.3: HMM with ergodic topology

no knowledge of the inner structure of a given problem and only surface observations are known [Abou-Moustafa et al., 2004]. One way to find the number of states for an HMM is empirically (as we do in this dissertation).

There are research problems where knowledge of the structure of the elements being analysed help to define the architecture of the HMM (to decide topology and number of states). An example of such a research problem is "part of speech tagging" (POS) [Manning and Schütze, 1999]. A typical POS problem is to partition a speech into phonemes. Then use the phonemes to try to identify syllables. A label is assigned to each phoneme. We know in advance how many labels we have and therefore create a state for each label. However, there are cases that show better classification results with larger number of states.

In POS *a priori* knowledge of the problem structure allows to define the HMM architecture beforehand. However, if the architecture is decided empirically, the meaning of the states can be deduced by answering question 2 of the HMMs. Given an observation sequence and a model obtain the sequence of states that produced such observation sequence. A correlation from the observation sequence and the state sequence can be made thereafter.

6.3.3 Common Uses of HMMs

In [Jaakkola and Haussler, 1998], HMMs are used in a DNA splice site classification problem, where the objective is to recognise true splice sites. The experiment was run with 2029 positive examples and 7321 false examples. The results were compared against a naïve Bayes approach and HMM showed a lower ratio in both false positive and false negative errors.

Another field is multiple sequence alignment of protein families and domains as seen in [Krogh et al., 1994]. Both HMM and PROFILESEARCH© [Accelrys Inc., 2002] (a technique used to search for relationships between a protein sequence and multiple aligned sequences) perform better in these tests than PROSITE (a dictionary of sites and patterns in proteins) [Swiss Institute of Bioinformatics, 2002]. The HMM is reported to have

a slight advantage over PROFILESEARCH©in terms of lower rates of errors.

HMMs have been used in speech recognition. In [Rabiner, 1989] a number of approaches are given in order to solve this problem but due to the complexity of the problem no definitive results have arisen. HMMs have also been widely used mostly in anomaly intrusion detection, e.g. [Warrender et al., 1999, Tan and Maxion, 2002, Qiao et al., 2002, Yeung and Ding, 2003]. In general terms what these approaches do is to train HMMs with valid sequences of system calls. A valid sequence is such that it was generated by a user of the system. Then they parse new unseen sequences of system calls, and based on the probability that such sequences were generated by the trained HMMs these IDSs can determine if the sequence is an anomaly or not.

6.4 Data Choice for Service Selection

As training data we selected a number of sessions for every service. The training sessions are non-attack sessions from a 40 day repository. Using the same scheme as in chapter 5, we calculate the number of training sessions using equation (6.1).

$$n = \frac{N}{1 + Ne^2} \tag{6.1}$$

From table 6.1 we can appreciate the sample size required for each service (with a confidence that the sample represents the entire population of 98%). We will need 226 sessions for `ftp` service, and 583 sessions for `finger` service. Since those numbers are very close to the available number of sessions (248 and 760 respectively), we an alternative approach an select a third of the total number of sessions as the sample population. This gives us a total of 83 `ftp` sessions, and 253 `finger` sessions.

Table 6.1: Available data and representative sample size with a confidence of 98%

	Population Size	Sample size
`telnet`	4784	1642
`smtp`	11152	2042
`ftp`	248	226
`finger`	760	583

6.5 A Methodology for Service Selection

The methodology to extract n-grams representing session headers uses a frequency analysis similar to the one proposed in chapter 5. We want to find those n-grams that characterise the beginning of each session of the same service. Services that need authentication might have two different n-grams; one

for successful and one for unsuccessful authentication. Such n-grams will allow us to identify the service a session belongs to.

We know that headers for the same service are not necessarily identical. Headers show small variations. The variations are repetitive among different sessions: the same variation will show up on many sessions. That is the reason why we only select n-grams with a frequency equal to the number of sessions of a given service. We repeat the frequency analysis incrementing the size of the n-grams until no n-gram has a frequency equal to the number of sessions. At this point we know the largest possible n-gram header common to all sessions. Larger n-grams are not guaranteed to be equal for every session. We also need to know how many different system calls exist between different headers, the length of the largest header, and the end of the header.

By extracting both the largest n-gram in the header, and the closest n-gram to the first `fork`, we can identify the entire header. After an identification of the headers for every session, we proceed to isolate them.

After isolation we concatenate each header corresponding to the same service. This concatenation creates a file containing headers of a given service. We call these concatenations n-gram families. To identify the members of these families we use HMMs.

In figures 6.4 and 6.5 we show training times (in seconds) for a number of headers for each service. HMMs training time is exponential, it depends on the number of states defined in the HMM and the length of the training sequences. The larger the number of states the better the classification, in practice the increment in performance stops at a certain number of states. There is no theoretical way to determine the ideal number of states in advance, however empirically we have found that for header n-gram families 30 states are enough to make a proper discrimination, i.e. to be able to distinguish between two different services.

From the five-tuple that defines an HMM we only need to provide two elements: the vocabulary M and the number of states N. The other three parameters $\prod$, A, and B are calculated by the Baum-Welch algorithm (mentioned in §6.3), this is also called training the HMM. M is provided by the problem itself as the set of different objects ,or vocabulary, in the training sequence. We use the set of possible system calls as our vocabulary. The methodology we use to find the right number of states N for an HMM is as follows. First, take an HMM with one single state. Train the HMM with a set of training examples. Then, parse a set of cross-validation examples with the HMM, thereby obtaining the probability $P(O|\lambda)$ where O are the cross-validation sequences. Next, add one state to the HMM and then test the enhanced HMM against the training set. If the improvement on the fitness measure is above a threshold then keep adding states. Otherwise, terminate. The algorithm is described in Algorithm 3. This method's main problem is that HMM training is in general time consuming, but is done offline and once finished it can be efficiently used. The maximum number of states for the HMMs is equal to the length of the shortest header n-gram.

The times shown in figures 6.4 and 6.5 were calculated in a Pentium IV @

Algorithm 3 Number of States Calculation

$NumStates = 1$
$Fitness = 0$
$TS = TrainingSet$
$CVS = CrossValidationSet$
repeat
  $OldFitness = Fitness$
  $HMM = TrainHMM(TS, NumStates)$
  $Fitness = ParseHMM(HMM, CVS)$
  $NumStates = NumStates + 1$
until $(Fitness - OldFitness) > Threshold$

2.6 GHz, 1GB of RAM, and running Mandrake ©Linux 9.0.

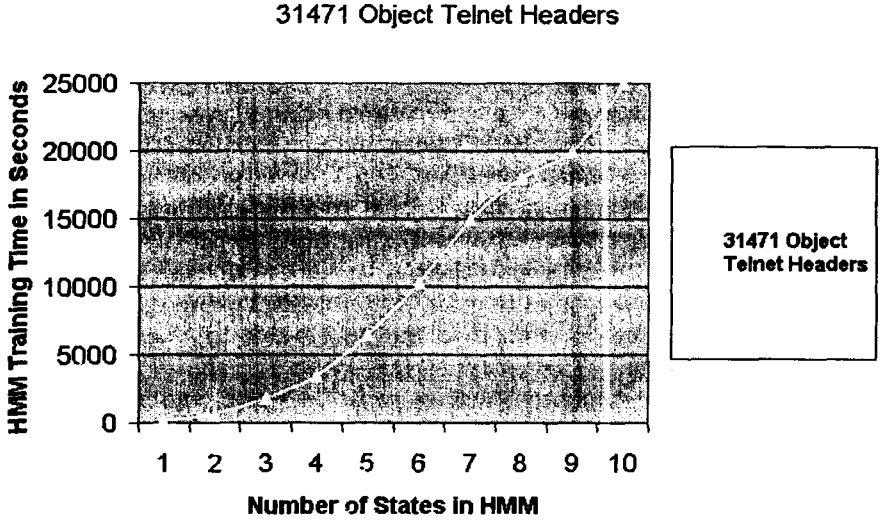


Figure 6.4: Telnet Service HMM Training Times

6.6 Validation Experiments and Data Selection

For validation purposes we incorporated all the data from the training samples, plus the same amount of unseen samples. The size of the validation samples is described in table 6.2.

We can appreciate that by using two thirds of the population as the validation sample we obtain a confidence of 95.5% for `ftp`, and 97.4% for `finger`.

The experiments used for validation are as follows. The test is conducted by parsing sessions from a given service using HMMs of the same service for

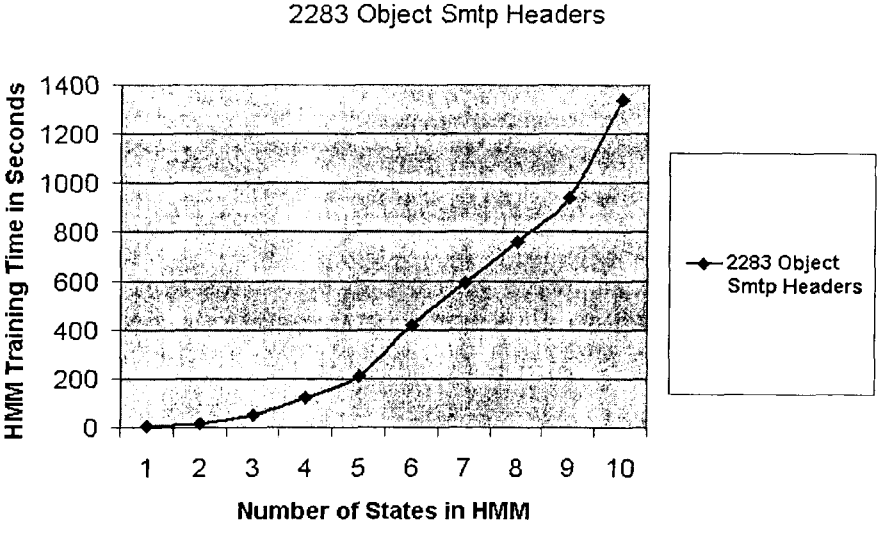


Figure 6.5: Smtp Service HMM Training Times

Table 6.2: Available data and representative sample size for 40 days of log files

	Validation Samples	Confidence of Validation Sample
`telnet`	3284	99%
`smtp`	4084	99%
`ftp`	166	95.5%
`finger`	506	97.4%

hitting ratio and using HMMs of different services for false positive test. Ideally all sessions of a service would be correctly classified by an HMM of the same service and have a hitting ratio of 100%. And all sessions of a service different from the HMM would have a hitting ratio of 0%.

6.7 Experimental Validation

We generated HMMs for each of these services: `telnet`, `smtp`, `ftp`, `finger`. In the case of `telnet` and `ftp` we separated each of them in two HMMs; one for successful login and one for unsuccessful. Then we tested the generated HMMs against headers from the same service and from other services. The results are summarised in table 6.3.

The first column in table 6.3 represents the HMMs for each service. The percentage is the number of correctly classified headers. The classification has a low probability if the header belongs to the HMM or low probability if it does

Table 6.3: % of Correct Service Discrimination

HMMs	telnet headers	smtp headers	ftp headers	finger headers
telnet	100%	1%	1%	0%
smtp	1%	100%	2%	0%
ftp	0%	1%	100%	0%
finger	0%	0%	0%	100%

not belong. When an smtp, ftp, or finger session is started for example from within a telnet session, those sessions are considered as a part of the telnet session.

6.8 Conclusions

By using the service discrimination module, we can scale IDSs by adding detectors for any new service we need to monitor, services can be monitored on demand. As long as the header of a service does not change (the general structure of the header depends on the protocol) such service will be accurately classified independently of configuration values for the service. The detection module also benefits from the service discrimination by reducing the search space for both misuse and anomaly detection. There is no negative impact on intrusion detection, false positives in service selection only imply more parsing for the intrusion detection mechanisms. If there are 200 telnet sessions, 100 smtp sessions and 100 ftp sessions, the telnet IDS will parse 202 sessions. Each service IDS gets to analyse all the sessions corresponding to that service, no session is lost due to the false positive ratio.

From the desired characteristics for an IDS described in §6.1, we conclude that the service discrimination mechanism delivers each characteristic as follows:

Flexibility, the configuration for the service we are monitoring can change, i.e. by using different port numbers. Service selection will not be affected by these changes since we are analysing the behaviour but not specific configuration.

Efficiency, the search space for detection is reduced regardless whether we adopt misuse or anomaly detection, (i.e. only attacks specific to the selected service are considered).

Scalability, in order to monitor a new service, we need only to add a discriminator for that service, and train the misuse and anomaly detection modules accordingly.

Chapter 8

Conclusions

In chapter 1, we presented briefly the research contributions of this dissertation. These were given in the form of three hypotheses that are supported by the evidence presented in this dissertation. Having described the state of the art in intrusion detection, the background theory, our methodology, our techniques, the implementation, the experiments and related work in more detail, we are now in a position to discuss these hypotheses in detail.

Hypothesis 1 By using pre-processing mechanisms we can improve the performance of current intrusion detection methods by operating on compact information. The mechanisms keep key information necessary for a proper intrusion detection.

In chapter 4 we presented an attribute filter based on rough sets. The method selects those attributes that best describe the objects in the BSM log files. The reduced attribute set is equivalent up to information to the original attribute set. Objects can still be differentiated from each other, almost as if all the attributes were used.

In chapter 5 a session folding method based on n-gram models was presented. The method greatly reduces session length and is capable of returning to the original session.

Hypothesis 2 By using an architecture that allows one to plug-in specialised intrusion detectors, we can make intrusion detection scalable.

In chapter 3 we introduced our novel architecture for intrusion detection. In chapter 6 we described our service selection module and the results of using it as a service classifier. The module added scalability is a result of its generalised design. It allows for plug-ins in the form of intrusion detection sensors. These sensors can be trained for either, misuse or anomaly detection. And they are specialised to a given service.

Hypothesis 3 We detect mimicry attacks with the help of hidden Markov models and word networks.

In chapter 7 we presented a method to detect mimicry attacks. Every other method that has aimed at detecting these kinds of attacks have shown to be easy to bypass. The false positive ratio of our technique can be further improved as we will describe in §8.1. We achieved a very decent detection ratio with an acceptable number of false positives. Our approach considers a class of general case mimicry attacks and has good results. The results are more impressive if we consider that no other technique has been able to tackle mimicry attacks, ours is a great improvement in the state of the art

8.1 Future Work

8.1.1 Attribute Reduction, Alternative Applications

Attribute reduction can be improved even further, by using newer reduct extraction methods like *order based genetic algorithms* and *approximate entropy* [Ślęzak and Wróblewski, 2003, Bazan et al., 2003] instead of Johnson's algorithm. These methods claim to extract a minimal decision reduct.

The reduct extraction can also be used to merge different log files into a single log file. The attributes that will appear in the centralised log file can be extracted using rough sets. Once the set of attributes that best discerns between objects is identified, an analysis using all those log files can be done. The analysis can be forensic or real time for intrusion detection.

8.1.2 Session Folding, Alternative Applications

Even though the added number of objects to the vocabulary is small, further reduction can improve performance even more. And more methods whose training time depends on the vocabulary size, like probabilistic context-free grammars (PCFGs) will become an alternative for real time detection.

Our results may help to improve the training times reported in the experiments by Forrest et al. and by Ciao et al. Wagner and Soto have a paper [Wagner and Soto, 2002] describing the disadvantages of using only short sequences as the detection base using HMMs. With the use of the our method equivalent sequences of larger size can be analysed using the same methodology.

8.1.3 Service Selection, Future Work

As future work we need to test the actual benefit of reducing the search space for anomaly detection. Test of the system running without discrimination and with discrimination.

The IDS should be extended to include other services like `ssh`, `sftp`, and `rpc`.

8.1.4 Towards Full Mimicry Attack Detection

Mimicry attack detection needs a lot of tuning. Currently we allow for no-ops in arbitrarily defined positions. We concentrated in a very specific sub-class of the general case, the general case as a whole is a bit more complex. The full general case eliminates two restrictions from our studied class. First, the model can be extended to allow for no-ops in every position of the attack. Second, the no-ops considered should be extended as to include all context dependant no-ops. In our analysis we considered a class of context dependant no-ops. The automatic creation of these no-ops is not a trivial task since a deep knowledge of the inner workings of an attack is needed.

No-ops can not be filtered out because even system calls returning failure which might be considered as no-ops, might be part of an attack's signature in properly positioned, if the position is changed then it becomes a no-op. Our system has a limit in the number of no-ops that can be inserted in each position, if a large number of no-ops is inserted (over 500) then the word network would regard the sequence as an attack with a low probability. This area can also be extended to tolerate a larger number of no-ops.

We propose two approaches to tackle general case mimicry attacks. The first one is to use HMMs but use no-ops as silent symbols. As no-ops are silent, any number of no-ops can be inserted in any position in the attack sequence. Then the no-ops would be consumed (filtered) by the HMM. The other alternative is to use PCFGs. The grammar should include production rules with no-ops, that way the no-ops will be consumed by the parser and the attacks can still be detected. To train the PCFGs, training sessions should be bracketed in a way that no-ops are easily spotted. The training sessions should be synthetically created to assure that the no-ops include context dependant no-ops. The challenge with both approaches is to define which symbols are considered no-ops and how to create context dependant no-ops. A method to automatically create the no-ops can be inspired in the work by [Gorodetski and Kotenko, 2002]. In this work, Gorodetski and Kotenko reproduce an attacker's behaviour using context-free grammars. Such an approach might be extended to synthetically create mimicry attacks.

Word networks can also be used to organise all our misuse database in the same place and only calculate the most likely path of an attack. This can be done by using a word network for each service. Each network is composed of a series of parallel nodes with the same transition probability. Each of this nodes corresponds to an attack subnetwork.

8.1.5 Anomaly Detection Suggested Work

We still need to prove that probabilistically combining the output from a MIDS and an AIDS will improve detection ratio, and at the same time will lower the number of false positives.

We are currently testing anomaly detection using PCFGs to generate normal user profiles. The main problem with this method is the training time. For a

set of 80 different sessions at least a month and a half training period is needed. We still need to test different data representations, however with the use of bracketing, an improvement in training time is obtained.

8.2 Conclusions

We proved that our reduction methods are suitable for intrusion detection and yield very good reduction ratios. Our novel architecture allows for scalability of an IDS and to test different detection methods with the same modular approach.

Detection of mimicry attacks is a recent problem and as such it has a long road ahead. We have but taken the first step towards successful mimicry attack detection. There is still a lot of research left in this area but ours is in a promising direction with very good results.

Bibliography

[Abou-Moustafa et al., 2004] Abou-Moustafa, K., Cheriet, M., and C.Y., S. (2004). On the Structure of Hidden Markov Models. *Pattern Recognition Letters*, 25:923–931.

[Accelrys Inc., 2002] Accelrys Inc. (2002). PROFILESEARCH. http://biobase.dk/gcgmanual/profilesearch.html.

[Alessandri et al., 2001] Alessandri, D., Cachin, C., Dacier, M., Deak, O., Julisch, K., Randell, B., Riordan, J., Tscharner, A., Wespi, A., and Wüest, C. (2001). Towards a Taxonomy of Intrusion Detection Systems and Attacks. Research Report RZ 3366, IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland.

[Amoroso, 1994] Amoroso, E. G. (1994). *Fundamentals of Computer Security Technology*. Prentice-Hall.

[Anderson et al., 1995] Anderson, D., Lunt, T. F., Javitz, H., Tamaru, A., and Valdes, A. (1995). Detecting unusual program behavior using the stastistical component of the next-generation intrusion detection expert system (nides). Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA.

[Arvidson and Carlbark, 2003] Arvidson, M. and Carlbark, M. (2003). Intrusion Detection Systems–Technologies, Weaknesses and Trends. Master's thesis, Linköpings Univesitet.

[Axelsson, 1999] Axelsson, S. (1999). On a Difficulty of Intrusion Detection. In *Recent Advances in Intrusion Detection*.

[Axelsson, 2000] Axelsson, S. (2000). Aspects of the Modelling and Performance of Intrusion Detection. Department of Computer Engineering, Chalmers University of Technology. Thesis for the degree of Licentiate of Engineering.

[Balasubramaniyan et al., 1998] Balasubramaniyan, J., Garcia-Fernandez, J., Isacoff, D., Spafford, E., and Zamboni, D. (1998). An Architecture for Intrusion Detection Using Autonomous Agents. Technical Report 98/05, Purdue University, COAST Laboratory, West Lafayete, Indiana, USA.

[Barbará and Jajodia, 2002] Barbará, D. and Jajodia, S., editors (2002). *Applications of Data Mining in Computer Security.* Advances in Information Security. Kluwer Academic Publishers, first edition.

[Bauer and Koblentz, 1988] Bauer, D. and Koblentz, M. (1988). NDIX–An expert system for real-time network intrusion detection. In *IEEE Computer Networking Symposium*, pages 98–106.

[Bazan et al., 2003] Bazan, J., Skowron, A., Ślęzak, D., and Wróblewski, J. (2003). Searching for the Complex Decision Reducts: The Case Study of the Survival Analysis. In *Proc. of the ISMIS'2003*, volume 2871 of *LNAI*, pages 160 – 168, Maebashi, Japan. Springer-Verlag.

[Bishop, 2003] Bishop, M. (2003). *Computer Security: Art and Science.* Addison-Wesley.

[Boleslaw and Yongqiang, 2004] Boleslaw, S. and Yongqiang, Z. (2004). Recursive Data Mining for Masquerade Detection and Author Identification. In *Proceedings of the 5th IEEE System, Man and Cybernetics Information Assurance Workshop*, pages 424–431, West Point, NY. IEEE.

[Brown, 1999] Brown, M. (1999). *RNA Modeling Using Stochastic Context-Free Grammars.* PhD thesis, University of California, Santa Cruz.

[Brown, 2000] Brown, M. (2000). Small Subunit Ribosomal RNA Modeling Using Stochastic Context-Free Grammars. In *ISMB 2000*, pages 57–66.

[Cisco Systems, 2002] Cisco Systems, I. (2002). Cisco Secure Intrusion Detection System. http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/prodlit/netra_ds.htm.

[Cohen et al., 2003] Cohen, I., Sebe, N., Sun, Y., Lew, M., and Huang, T. (2003). Evaluation of Expression Recognition Techniques. In *International Conference on Image and Video Retrieval (CIVR'03)*, pages 184–195, Urbana, USA.

[Computer Security Products Inc., 2002] Computer Security Products Inc. (2002). Alert-plus. http://www.compsec.com/.

[Computer Security Research, 2002] Computer Security Research (2002). ACME!: Advanced counter-measures environment. http://www.acme-ids.org/acme-ids/.

[Debar et al., 1999] Debar, H., Dacier, M., and Wespi, A. (1999). A revised Taxonomy for Intrusion Detection Systems. Technical report, Zurich Research Laboratory, IBM Research Division.

[Denning and Neumann, 1985] Denning, D. E. and Neumann, P. G. (1985). Requirements and model for IDES-A real-time intrusion detection system. Technical report, Coputer Science Laboratory, SRI International, Menlo Park, CA.

[Denning and Neumman, 1987] Denning, D. E. and Neumman, P. G. (1987). A prototype ides: A real-time intrusion detection expert system. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA.

[Farmer and Venema, 2002] Farmer, D. and Venema, W. (2002). Security administrator tool for analyzing networks. http://www.cerias.purdue.edu/coast/satan.html.

[Félix and Ushio, 2002] Félix, R. and Ushio, T. (2002). Binary Encoding of Discernibility Patterns to Find Minimal Coverings. *International Journal of Software Engineering and Knowledge Engineering*, 12(1):1–18.

[Ferenčik, 1993] Ferenčik, M. (1993). *Handbook of Immunochemistry*. CHAPMAN & HALL, first edition.

[Flynn and McHoes, 1991] Flynn, I. M. and McHoes, A. (1991). *Understanding Operating Systems*. PWS Publishing Company.

[Focus, 2001] Focus, S. (2001). Mscan v1.0. http://online.securityfocus.com/tools/228.

[Forrest et al., 1996] Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996). A Sense of Self for Unix Processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA. IEEE Computer Society Press.

[Free Software Foundation, 2002] Free Software Foundation, I. (2002). Snort ©. http://www.snort.org.

[Freeman and Skapura, 1991] Freeman, J. A. and Skapura, D. M. (1991). *Neural Networks: Algorithms, Applications and Programming Techniques*. Computation and Neural System Series. Addison-Wesley Publishing Company, Inc.

[Godínez et al., 2004a] Godínez, F., Hutter, D., and Monroy, R. (2004a). Attribute Reduction for Effective Intrusion Detection. In Favela, J., Menasalvas, E., and Chávez, E., editors, *Proceedings of the 2004 Atlantic Web Intelligence Conference, AWIC'04*, volume 3034 of *Lecture Notes in Artificial Intelligence*, pages 74–83. Springer-Verlag.

[Godínez et al., 2004b] Godínez, F., Hutter, D., and Monroy, R. (2004b). Service Discrimination and Audit File Reduction for Effective Intrusion Detection. In *Proceedings of the 2004 Workshop on Information Security Applications, WISA'04*, volume 3325 of *Lecture Notes in Computer Science*, pages 101–115. Springer-Verlag.

[Godínez et al., 2005a] Godínez, F., Hutter, D., and Monroy, R. (2005a). Audit File Reduction Using N-Gram Models (Work in Progress). In Patrick, A. and Yung, M., editors, *Proceedings of the Ninth International Conference on Financial Cryptography and Data Security, FC'05*, volume 3570 of *Lecture*

Notes in Computer Science, pages 336–340, Roseau, The Commonwealth Of Dominica. Springer-Verlag.

[Godínez et al., 2005b] Godínez, F., Hutter, D., and Monroy, R. (2005b). On the Role of Information Compaction to Intrusion Detection. In Unger, H., editor, *Proceedings of the Fifth IEEE International Symposium and School on Advance Distributed Systems ISSADS 2005*, volume 3563 of *Lecture Notes in Computer Science*, pages 83–97, Guadalajara, Jalisco, México. Springer-Verlag.

[Gorodetski and Kotenko, 2002] Gorodetski, V. and Kotenko, I. (2002). Attacks Against Computer Network: Formal Grammar-Based Framework and Simulation Tool. In *RAID 2002*, volume 2516 of *LNCS*, pages pp. 219–238, Berlin Heidelberg. Springer-Verlag.

[Habra et al., 1992] Habra, N., Le Charlier, B., Mounji, A., and Mathieu, I. (1992). ASAX: Software architecture and rule-based language for universal audit trail analysis. In *Proceedings of the 2nd European Symposium on Research in Computer Security (ESORICS' 92)*, pages 435–450, Toulouse, France.

[Haines et al., 2001] Haines, J. W., Lippmann, R. P., Fried, D. J., Tran, E., Boswell, S., and Zissman, M. A. (2001). 1999 DARPA Intrusion Detection System Evaluation: Design and Procedures. Technical Report 1062, Lincoln Laboratory, Massachusetts Institute of Technology.

[Heberlein et al., 1990] Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., and Wolber, D. (1990). A network security monitor. In IEEE, editor, *IEEE CS Symposium on Research in Security and Privacy*, pages 296–304, New York, NY.

[Hochberg et al., 1993] Hochberg, J., Jackson, K., Stallings, C., McClary, J., DuBois, D., and Ford, J. (1993). NADIR: An automated system for detecting network intrusion and misuse. *Computers and Security*, pages 235–248. Elsevier Science, New York.

[Hofmeyr and Forrest, 2000] Hofmeyr, S. and Forrest, S. (2000). Architecture for an Artificial Immune System. In *Evolutionary Computation*, volume 7, pages 1289–1296, San Francisco, CA. Morgan-Kaufmann.

[Hofmeyr et al., 1997] Hofmeyr, S., Forrest, S., and Somayaji, A. (1997). Intrusion Detection Using Sequences of System Calls. Technical report, Dept. of Computer Science, University of New Mexico, Albuquerque, NM.

[Holbrook and Reynolds, 1991] Holbrook, P. and Reynolds, J. (1991). Site security handbook.

[Hughey and Krogh, 1996] Hughey, R. and Krogh, A. (1996). Hidden Markov Models for Sequence Analysis: Extension and Analysis of Basic Method. *Comp. Appl. BioSci*, 12(2):95–108.

[Jaakkola and Haussler, 1998] Jaakkola, T. S. and Haussler, D. (1998). Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, volume 11. The MIT Press.

[Jansen et al., 1999] Jansen, W., Mell, P., Karygiannis, T., and Marks, D. (1999). Applying mobile agents to intrusion detection and response. NIST Interim Report (IR) 6416, NIST National Institute of Standards and Technology, Computer Security Division.

[Javitz et al., 1986] Javitz, H. S., Denning, D. E., and Neumann, P. G. (1986). Analytical techniques development for a statistical intrusion detection system (sids) based on accounting records. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA.

[Johnson, 1974] Johnson, D. S. (1974). Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Sciences*, 9:256–278.

[Karlsson, 2001] Karlsson, C. (2001). Methos for Intrusion and Fraud Detection in IP-Based Multimedia Services. Master's thesis, Chalmers University of Technology.

[Kendall, 1998] Kendall, K. (1998). A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems. Master's thesis, Massachusetts Institute of Technology.

[Kim and Bentley, 1999a] Kim, J. and Bentley, P. (1999a). An Artificial Immune Model for Network Intrusion Detection. In *Proceedings of the 7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany. ELITE Foundation.

[Kim and Bentley, 1999b] Kim, J. and Bentley, P. (1999b). The Human Immune System and Network Intrusion Detection. In *Proceedings of the 7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany. ELITE Foundation.

[Klop, 1992] Klop, J. W. (1992). Term Rewriting Systems. In Abramsky, S., Gabbay, D., and Maibaum, T. S., editors, *Handbook of Logic in Computer Science, vol 2*, volume 2, pages 1–116. Clarendon Press, Oxford.

[Knop et al., 2001] Knop, M. W., Schopf, J. M., and Dinda, P. A. (2001). Windows Performance Monitoring and Data Reduction Using WatchTower and Argus. Technical Report Technical Report NWU-CS-01-6, Department of Computer Science, Northwestern University.

[Komorowski et al., 1998] Komorowski, J., Polkowski, L., and Skowron, A. (1998). *Rough-Fuzzy Hybridization – A New Trend in Decision Making*, chapter Rough Sets: A Tutorial, pages 3–98. Springer-Verlag.

[Krogh et al., 1994] Krogh, A., Brown, M., Mian, I. S., Sjölander, K., and Haussler, D. (1994). Hidden Markov Models in Computational Biology: Applications to Protein Modeling. *Journal Molecular Biology*, 235:1501–1531.

[Lane and Brodley, 1999] Lane, T. and Brodley, C. E. (1999). Temporal Sequence Learning and Data Reduction for Anomaly Detection. *ACM Transactions on Information and System Security*, 2(3):295–331.

[Lane and Brodley, 2000] Lane, T. and Brodley, C. E. (2000). Data Reduction Techniques for Instance-Based Learning from Human/Computer Interface Data. In *Proceedings of the 17th International Conference on Machine Learning*, pages 519–526. Morgan Kaufmann.

[Lindqvist and Porras, 1999] Lindqvist, U. and Porras, P. A. (1999). Detecting computer and network misuse through the production-based expert system toolset (p-best). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California. IEEE Computer Society Press, Los Alamitos, California.

[Lindqvist and Porras, 2001] Lindqvist, U. and Porras, P. A. (2001). expert-bsm: A host-based intrusion detection solution for sun solaris. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, pages 240–251, New Orleans, Louisiana. IEEE Computer Society.

[Mamitsuka, 1997] Mamitsuka, H. (1997). Supervised Learning of Hidden Markov Models for Sequence Discrimination. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, pages 202–208. ACM Press.

[Manning and Schütze, 1999] Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* MIT Press, Massachusets Institute of Technology, Cambridge, Massachusets 02142.

[Marceau, 2000] Marceau, C. (2000). Characterizing the Behavior of a Program Using Multiple-Length N-grams. In *Proceedings of the 2000 Workshop on New Security Paradigms*, pages 101–110. ACM Press.

[Marin et al., 2001] Marin, J. A., Ragsdale, D., and Surdu, J. (2001). A Hybrid Approach to Profile Creation and Intrusion Detection. In *Proc. of DARPA Information Survivability Conference and Exposition.* IEEE Computer Society.

[Martínez, 2003] Martínez, R. (2003). Letter Describing How to Calculate the Size of a Representative Sample, With Certain Confidence. Personal Communication.

[Maxion and Townsend, 2002] Maxion, R. and Townsend, T. (2002). Masquerade Detection Using Truncated Command Lines. In *Proceedings of the International Conference on Dependable Systems & Networks*, pages 219–228, Washington, DC. IEEE.

[Maxion and Tan, 2000] Maxion, R. A. and Tan, K. M. C. (2000). Benchmarking Anomaly-Based Detection Systems. In *Proceedings of the 1st International Conference on Dependable Systems and Networks*, pages 623–630, New York, New York, USA. IEEE Computer Society Press.

[Maxion and Tan, 2002] Maxion, R. A. and Tan, K. M. C. (2002). Anomaly Detection in Embedded Systems. *IEEE Transactions on Computers*, 51(2):108–120.

[Mé, 1998] Mé, L. (1998). Gassata, a genetic algorithm as an alternative tool for security audit trails analysis. In *First international workshop on the Recent Advances in Intrusion Detection*, Louvain-la-Neuve, Belgium.

[Mell and McLarnon, 2000] Mell, P. andMarks, D. and McLarnon, M. (2000). A Denial of Service Resistant Intrusion Detection Architecture. *Computer Networks*, 34(4):641–658.

[Mell and McLarnon, 1999] Mell, P. and McLarnon, M. (1999). Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems. In *RAID Recent Advances in Intrusion Detection, Second International Workshop*, West Lafayete, Indiana, USA. online proceedings: http://www.raid-symposium.org/raid99/.

[MimeStar, 2002] MimeStar, I. (2002). Securenet PRO. http://www.mimestar.com/products/index.html.

[Nolazco et al., 2004] Nolazco, J. A., Galván, A., Mandujano, S., Mex, C., Parra, A., Pfeiffer, C., and García, P. (2004). 2004 status report. Personal Communication.

[Øhrn and Komorowski, 1997] Øhrn, A. and Komorowski, J. (1997). ROSETTA: A Rough Set Toolkit for Analysis of Data. In Wong, P., editor, *Proceedings of the Third International Joint Conference on Information Sciences*, volume 3, pages 403–407, Durham, NC, USA. Department of Electrical and Computer Engineering, Duke University.

[Pereira and Riley, 1997] Pereira, F. and Riley, M. (1997). Speech Recognition by Composition of Weighted Finite Automata. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 431–453. MIT press, Cambridge, MA.

[Porras and Neumann, 1997] Porras, P. A. and Neumann, P. G. (1997). Emerald: Event monitoring enabling responses to anomalous live disturbances. In *20th National Information System Security Conference.*

[Proctor, 1994] Proctor, P. (1994). Audit reduction and misuse detection in heterogeneous environments: Framework and applications. In *10th Annual Computer Security Applications Conference*, pages 117–125.

[Qiao et al., 2002] Qiao, Y., Xin, X., Bin, Y., and Ge, S. (2002). Anomaly Intrusion Detection Method Based on HMM. *Electronic Letters*, 38(13):663–664.

[Rabiner, 1989] Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Rencher, 1995] Rencher, A. (1995). *Methods in Multivariate Analysis*. Wiley & Sons, New York.

[SAINT Corporation, 2002] SAINT Corporation (2002). Saint. http://www.saintcorporation.com/products/saint_engine.html.

[Sakakibara et al., 1994] Sakakibara, Y., Brown, M., Hughey, R., Mian, I. S., Sjölander, K., Underwood, R. C., and Haussler, D. (1994). Stochastic Context-Free Grammars for tRNA Modeling. *Nucleic Acids Research*, 22:5112–5120.

[Schonlau et al., 2001] Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., and Vardi, Y. (2001). Computer Intrusion: Detecting Masquerades. *Statistical Science*, 16:1–17. To appear.

[Scott et al., 2003] Scott, C., Joel, B., Boleslaw, S., and Eric, B. (2003). Intrusion Detection: A Bioinformatics Approach. In *Proceeding of the 19th Annual Computer Security Applications Conference*, pages 24–33, Las Vegas, Nevada.

[Ślęzak and Wróblewski, 2003] Ślęzak, D. and Wróblewski, J. (2003). Order Based Genetic Algorithms for the Search of Approximate Entropy Reducts. In *Proc. of the RSFDGrC'2003*, volume 2639 of *LNAI*, pages 308 – 311, Chongqing, China. Springer-Verlag.

[Snapp et al., 1991] Snapp, S., Smaha, S. E., Grance, T., and Teal, D. M. (1991). A system for distributed intrusion detection. In IEEE, editor, *IEEE COMPCON*, pages 170–176, New York, NY.

[Sobirey et al., 1996] Sobirey, M., Richter, B., and Konig, H. (1996). The intrusion detection system aid. In *Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security*, Architecture, and experiences in automated audit analysis, pages 278–290.

[Stainford-Chen et al., 1996] Stainford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D. (1996). Grids – a graph-based intrusion detection system for large networks. In *19th National Information Systems Security Conference*. http://seclab.cs.ucdavis.edu/papers.html.

[SunSHIELD BSM, 2000] SunSHIELD BSM (2000). *SunSHIELD Basic Security Module Guide*. Sun MicroSystems, part number 806-1789-10 edition.

[Swiss Institute of Bioinformatics, 2002] Swiss Institute of Bioinformatics (2002). PROSITE. http://au.expasy.org/prosite/.

[Tan and Maxion, 2002] Tan, K. M. C. and Maxion, R. A. (2002). Why 6?" Defining the Operational Limits of STIDE, an Anomaly-Based Intrusion Detector. In *Proceedings of IEEE Symposium on Security & Privacy*, pages 188–201.

[Tanenbaum, 1992] Tanenbaum, A. S. (1992). *Modern Operating Systems.* Prentice Hall.

[The Computer Security Institute, 2001] The Computer Security Institute, C. (2001). 2001 Computer Crime and Security Survey. http://www.gocsi.com/prelea000321.htm.

[The Computer Security Institute, 2003] The Computer Security Institute, C. (2003). 2003 Computer Crime and Security Survey. http://i.cmpnet.com/gocsi/db_area/pdfs/fbi/FBI2003.pdf.

[Valdes and Skinner, 2000] Valdes, A. and Skinner, K. (2000). Adaptive, model-based monitoring for cyber attack detection. http://www.sdl.sri.com/projects/emerald/adaptbn-paper/adaptbn.html.

[Vigna and Kemmerer, 1999] Vigna, G. and Kemmerer, R. A. (1999). NetSTAT: A Network-Based Intrusion Detection System. *Journal of Computer Security*, 7(1):37–71.

[Viterbo and Øhrn, 2000] Viterbo, S. and Øhrn, A. (2000). Minimal Approximate Hitting Sets and Rule Templates. *International Journal of Approximate Reasoning*, 25(2):123–143.

[Wagner and Soto, 2002] Wagner, D. and Soto, P. (2002). Mimicry Attacks on Host Based Intrusion Detection Systems. In *Proceedings of the Ninth ACM Conference on Computer and Communications Security*, pages 255–265, Washington, DC, USA. ACM.

[Warrender et al., 1999] Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting Intrusions Using System Calls: Alternative Data Models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society Press.

[Wespi et al., 1999] Wespi, A., Dacier, M., and Debar, H. (1999). An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. In Gattiker, U. E., Pedersen, P., and Petersen, K., editors, *Proceeedings of EICAR '99.*

[Yeung and Ding, 2003] Yeung, D. and Ding, Y. (2003). Host-Based Intrusion Detection Using Dynamic and Static Behavioral Models. *Pattern Recognition*, Vol. 36(No. 1):pp. 229–243.

[Young et al., 2002] Young, S., Evermann, G., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. (2002). *The HTK Book for HTK Version 3.2.* Cambridge University Engineering Department.

[Zhang et al., 2004] Zhang, L., Zhang, G., Yu, L., Zhang, J., and Bai, Y. (2004). Intrusion Detection Using Rough Set Classification. *Journal of Zhwjiang University Science*, 5(9):1076–1086.

Appendix A

BSM Log File Format

Reproduction of BSM documentation page downloaded from Sun Microsystems website.[1]

Audit Record Descriptions gives a detailed description of each audit token. The appendix also lists all the audit records generated by Solaris BSM auditing. The definitions are sorted in order of the short descriptions, and a cross-reference table translates event names to event descriptions.

Binary Format Audit records are stored and manipulated in binary form; however, the byte order and size of data is predetermined to simplify compatibility between different machines.

Audit Event Type Each auditable event in the system generates a particular type of audit record. The audit record for each event has certain tokens within the record that describe the event. An audit record does not describe the audit event class to which the event belongs; that mapping is determined by an external table, the **/etc/security/audit_event** file.

Audit Token Types Each token starts with a one-byte token type, followed by one or more data elements in an order determined by the type. The different audit records are distinguished by event type and different sets of tokens within the record. Some tokens, such as the **text** token, contain only a single data element, while others, such as the **process** token, contain several (including the audit user ID, real user ID, and effective user ID).

Order of Audit Tokens

Each audit record begins with a **header** token and ends (optionally) with a **trailer** token. One or more tokens between the header and trailer describe the event. For user-level and kernel events, the tokens describe the process that

[1] http://docs.sun.com/db/doc/806-1789/6jb25l4bc?a=view

performed the event, the objects on which it was performed, and the objects' tokens, such as the owner or mode.

Each user-level and kernel event typically has at least the following tokens:

- **header**
- **subject**
- **return**

Many events also include a `trailer` token, but it is optional.

Human-Readable Audit Record Format

This section shows each audit record format as it appears in the output produced by the **praudit** command. This section also gives a short description of each audit token. For a complete description of each field in each token, see Appendix A, Audit Record Descriptions.

The following token examples show the form that **praudit** produces by default. Examples are also provided of raw (**-r**) and short (**-s**) options. When **praudit** displays an audit token, it begins with the token type, followed by the data from the token. Each data field from the token is separated from other fields by a comma. However, if a field (such as a path name) contains a comma, this cannot be distinguished from a field-separating comma. Use a different field separator or the output will contain commas. The token type is displayed by default as a name, like **header**, or in **-r** format as a decimal number.

The individual tokens are described in the following order:

- "**header** Token"
- "**trailer** Token"
- "**arbitrary** Token"
- "**arg** Token"
- "**attr** Token"
- "**exit** Token"
- "**file** Token"
- "**groups** Token"
- "**in_addr** Token"
- "**ip** Token"
- "**ipc** Token"
- "**ipc_perm** Token"

- "**iport** Token"
- "**opaque** Token"
- "**path** Token"
- "**process** Token"
- "**return** Token"
- "**seq** Token"
- "**socket** Token"
- "**subject** Token"
- "**text** Token"

header Token

Every audit record begins with a **header** token. The **header** token gives information common to all audit records. The fields are:

- A token ID
- The record length in bytes, including the **header** and **trailer** tokens
- An audit record structure version number
- An event ID identifying the type of audit event
- An event ID modifier with descriptive information about the event type
- The time and date the record was created

When displayed by **praudit** in default format, a **header** token looks like the following example from **ioctl**:

header,240,1,ioctl(2),es,Tue Sept 1 16:11:44 1992, + 270000 msec

Using **praudit-s**, the event description (ioctl(2) in the default praudit example above) is replaced with the event name (`AUE_IOCTL`), like this:

header,240,1,AUE_IOCTL,es,Tue Sept 1 16:11:44 1992, + 270000 msec

Using **praudit-r**, all fields are displayed as numbers (that can be decimal, octal, or hex), where **158** is the event number for this event.

20,240,1,158,0003,699754304, + 270000 msec

Notice that **praudit** displays the time to millisecond resolution.

trailer Token

This token marks the end of an audit record and allows backward seeks of the audit trail. The fields are:

- A token ID
- A pad number that marks the end of the record (does not show)
- The total number of audit record characters including the **header** and **trailer** tokens

A **trailer** token is displayed by **praudit** as follows:
trailer,136

arbitrary Token

This token encapsulates data for the audit trail. The item array can contain a number of items. The fields are:

- A token ID
- A suggested format, such as decimal
- A size of encapsulated data, such as int
- A count of the data array items
- An item array

An **arbitrary** token is displayed by **praudit** as follows:
arbitrary,decimal,int,1 42

arg Token

This token contains system call argument information. A 32-bit integer system call argument is allowed in an audit record. The fields are:

- A token ID
- An argument ID of the relevant system call argument
- The argument value
- The length of an optional descriptive text string (does not show)
- An optional text string

An **arg** token is displayed by **praudit** as follows:
argument,1,0x00000000,addr

attr Token

This token contains information from the file vnode. The **attr** token is usually produced during path searches and accompanies a **path** token, but is not included in the event of a path-search error. The fields are:

- A token ID
- The file access mode and type
- The owner user ID
- The owner group ID
- The file system ID
- The inode ID
- The device ID that the file might represent

An **attr** token is displayed by **praudit** as follows:
attribute,100555,root,staff,1805,13871,-4288

exit Token

An **exit** token records the exit status of a program. The fields are:

- A token ID
- A program exit status as passed to the **exit()** system call
- A return value that describes the exit status or indicates a system error number

An **exit** token is displayed by **praudit** as follows:
exit,Error 0,0

file Token

This token is generated by the audit daemon to mark the beginning of a new audit trail file and the end of an old file as the old file becomes deactivated. The audit record containing this token links successive audit files into one audit trail. The fields are:

- A token ID
- A time and date stamp of a file opening or closing
- A byte count of the file name (does not show)
- The file name

A **file** token is displayed by **praudit** as follows:
file,Tue Sep 1 13:32:42 1992, + 79249 msec, /bau-
dit/localhost/files/19920901202558.19920901203241.quisp

groups Token

A **groups** token records the groups entries from a process's credential. The fields are:

- A token ID
- An array of groups entries of size NGROUPS_MAX(16)

A **groups** token is displayed by **praudit** as follows:
group,staff,wheel,daemon,kmem,bin,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1

in_addr Token

An **in_addr** token gives a machine Internet Protocol address. The fields are:

- A token ID
- An Internet address

An **in_addr** token is displayed by **praudit** as follows:
ip addr,129.150.113.7

ip Token

The **ip** token contains a copy of an Internet Protocol header. The fields are:

- A token ID
- A 20-byte copy of an IP header

An **ip** token is displayed by **praudit** as follows:
ip address,0.0.0.0

ipc Token

This token contains the System V IPC message/semaphore/shared-memory handle used by a caller to identify a particular IPC object. The fields are:

- A token ID
- An IPC object type identifier
- The IPC object handle

An **ipc** token is displayed by **praudit** as follows:
IPC,msg,3

ipc_perm Token

An **ipc_perm** token contains a copy of the System V IPC access information. Audit records for shared memory, semaphore, and message IPCs have this token added. The fields are:

- A token ID
- The IPC owner's user ID
- The IPC owner's group ID
- The IPC creator's user ID
- The IPC creator's group ID
- The IPC access modes
- The IPC sequence number
- The IPC key value

An **ipc_perm** token is displayed by **praudit** as follows:
IPC perm,root,wheel,root,wheel,0,0,0x00000000

iport Token

This token contains a TCP (or UDP) address. The fields are:

- A token ID
- A TCP/UDP address

An **iport** token is displayed by **praudit** as follows:
ip port,0xf6d6

opaque Token

The **opaque** token contains unformatted data as a sequence of bytes. The fields are:

- A token ID
- A byte count of the data array
- An array of byte data

An **opaque** token is displayed by **praudit** as follows:
opaque,12,0x4f5041515545204441544100

path Token

A **path** token contains access path information for an object. The fields are:

- A token ID
- A byte count of the path length (does not show)
- An absolute path

A **path** token is displayed by **praudit** as follows:
path,/an/anchored/path/name/to/test/auditwrite/AW_PATH

process Token

The **process** token contains information describing a process. The fields are:

- A token ID
- The user audit ID
- The effective user ID
- The effective group ID
- The real user ID
- The real group ID
- The process ID
- The session ID
- A terminal ID made up of:
  - A device ID
  - A machine ID

A **process** token is displayed by **praudit** as follows:
process,root,root,wheel,root,wheel,0,0,0,0.0.0.0

return Token

A **return** token gives the return status of the system call and the process return value. This token is always returned as part of kernel-generated audit records for system calls. The fields are:

- A token ID
- The system call error status
- The system call return value

A **return** token is displayed by **praudit** as follows:
return,success,0

seq Token

This token is optional and contains an increasing sequence number used for debugging. The token is added to each audit record when the **seq** policy is active. The fields are:

- A token ID
- A 32-bit unsigned long-sequence number

A **seq** token is displayed by **praudit** as follows:
sequence,1292

socket Token

A **socket** token describes an Internet socket. The fields are:

- A token ID
- A socket type field (TCP/UDP/UNIX)
- The local port address
- The local Internet address
- The remote port address
- The remote Internet address

A **socket** token is displayed by **praudit** as follows:
socket,0x0000,0x0000,0.0.0.0,0x0000,0.0.0.0

subject Token

This token describes a subject (process). The fields are:

- A token ID
- The user audit ID
- The effective user ID
- The effective group ID
- The real user ID
- The real group ID
- The process ID
- The session ID
- A terminal ID made up of:

- A device ID
- A machine ID

A **subject** token is displayed by **praudit** as follows:
subject,cjc,cjc,staff,cjc,staff,424,223,0 0 quisp

text Token

A **text** token contains a text string. The fields are:

- A token ID
- The length of the text string (does not show)
- A text string

A **text** token is displayed by **praudit** as follows:
text,aw_test_tokenNext:

Appendix B

ARPA Language Model File Format

Reproduction of man page downloaded from SRI website.[1]

NAME

ngram-format - File format for ARPA backoff N-gram models.

SYNOPSIS

\data
ngram 1=$n1$
ngram 2=$n2$
...
ngram $N = nN$
\1-grams:
p w [bow]
...
\2-grams:
p $w1$ $w2$ [bow]
...
****N**-grams:**
p $w1$ ... wN
...
\end

[1] http://www.speech.sri.com/projects/srilm/manpages/ngram-format.html

DESCRIPTION

The so-called ARPA (or Doug Paul) format for N-gram backoff models starts with a header, introduced by the keyword **\data**, listing the number of N-grams of each length. Following that, N-grams are listed one per line, grouped into sections by length, each section starting with the keyword **\N-gram:**, where N is the length of the N-grams to follow. Each N-gram line starts with the logarithm (base 10) of conditional probability p of that N-gram, followed by the words $w1 \ldots wN$ making up the N-gram. These are optionally followed by the logarithm (base 10) of the backoff weight for the N-gram. The keyword **\end** concludes the model representation.

Backoff weights are required only for those N-grams that form a prefix of longer N-grams in the model. The highest-order N-grams in particular will not need backoff weights (they would be useless).

Since log(0) (minus infinity) has no portable representation, such values are mapped to a large negative number. However, the designated dummy value (-99 in SRILM) is interpreted as log(0) when read back from file into memory.

The correctness of the N-gram counts $n1, n2, \ldots$ in the header is not enforced by SRILM software when reading models (although a warning is printed when an inconsistency is encountered). This allows easy textual insertion or deletion of parameters in a model file. The proper format can be recovered by passsing the model through the command ngram -order N -lm *input* -write-lm *output*

Note that the format is self-delimiting, allowing multiple models to be stored in one file, or to be surrounded by ancillary information. Some extensions of N-gram models in SRILM store additional parameters after a basic N-gram section in the standard format.

SEE ALSO

ngram(1), ngram-count(1), lm-scripts(1), pfsg-scripts(1).

BUGS

The ARPA format does not allow N-grams that have only a backoff weight associated with them, but no conditional probability. This makes the format less general than would otherwise be useful (e.g., to support pruned models, or ones containing a mix of words and classes). The ngram-count(1) tool satisfies this constraint by inserting dummy probabilities where necessary.

For simplicity, an N-gram model containing N-grams up to length N is referred to in the SRILM programs as an N-th order model, although technically it represents a Markov model of order $N - 1$.

BUGS

There is no way to specify words with embedded whitespace.

AUTHOR

The ARPA backoff format was developed by Doug Paul at MIT Lincoln Labs for research sponsored by the U.S. Department of Defense Advanced Research Project Agency (ARPA). Man page by Andreas Stolcke ¡stolcke@speech.sri.com¿.

Appendix C

HTK File Format for HMMs and Word Networks

All the definitions for the HTK language format were taken from [Young et al., 2002]. We will begin by presenting a formal description of the language used to represent an HMM, then we show some of the models we built for our experiments. All the definitions are valid for HTK version 2.0 and up. And the material presented in this text is copyright of Cambridge University Engineering Department. We would like to thank all the people that worked really hard in the HTK project so we were able to use such a valuable tool.

C.1 HMM Definition Language

Syntax for the HTK language format is described using an extended BNF notation in which alternatives are separated by a vertical bar |, parentheses () denote factoring, brackets [ ] denote options, and braces { } denote zero or more repetitions.

All keywords are enclosed in angle brackets and the case of the keyword name is not significant. White space is not significant except within double-quoted strings.

The top level structure of a HMM definition is shown by the following rule.

```
hmmdef =    [ ~h macro ]
            <BeginHMM>
                [ globalOpts ]
                <NumStates> short
                state { state }
                [ regTree ]
                transP
                [ duration ]
```

<EndHMM>

A HMM definition consists of an optional set of global options followed by the <NumStates> keyword whose following argument specifies the number of states in the model inclusive of the non-emitting entry and exit states. The information for each state is then given in turn, followed by the parameters of the transition matrix and the model duration parameters, if any. The name of the HMM is given by the ~h macro. If the HMM is the only definition within a file, the ~h macro name can be omitted and the HMM name is assumed to be the same as the file name.

The global options are common to all HMMs. They can be given separately using a ~o option macro

optmacro = ~o globalOpts

or they can be included in one or more HMM definitions. Global options may be repeated but no definition can change a previous definition. All global options must be defined before any other macro definition is processed. In practise this means that any HMM system which uses parameter tying must have a ~o option macro at the head of the first macro file processed.

The full set of global options is given below. Every HMM set must define the vector size (via <VecSize>, the stream widths (via <StreamInfo> and the observation parameter kind. However, if only the stream widths are given, then the vector size will be inferred. If only the vector size is given, then a single stream of identical width will be assumed. All other options default to null.

```
globalOpts = option { option }
option =     <HmmSetId> string |
             <StreamInfo> short { short } |
             <VecSize> short |
             <InputXform> inputXform |
             covkind |
             durkind |
             parmkind
```

The <HmmSetId> option allows the user to give the macro master files an identifier. The arguments to the <StreamInfo> option are the number of streams (default 1) and then for each stream, the width of that stream. The <VecSize> option gives the total number of elements in each input vector. If both <VecSize> and <StreamInfo> are included then the sum of all the stream widths must equal the input vector size.

The covkind defines the kind of the covariance matrix

```
covkind =    <DiagC> | <InvDiagC> | <FullC> |
             <LLTC> | <XformC>
```

where <InvDiagC> is used internally. <LLTC> and <XformC> are not used in HTK Version 2.0. Setting the covariance kind as a global option forces all

components to have this kind. In particular, it prevents mixing full and diagonal covariances within a HMM set.

The durkind denotes the type of duration model used according to the following rules

durkind = <nullD> | <poissonD> | <gammaD> | <genD>

For anything other than <nullD>, a duration vector must be supplied for the model or each state as described below. Note that no current HTK tool can estimate or use such duration vectors.

The parameter kind is any legal parameter kind including qualified forms

parmkind = <basekind{_D|_A|_T|_E|_N|_Z|_O|_V|_C|_K}>
basekind = <discrete>|<lpc>|<lpcepstra>|<mfcc> | <fbank> | <melspec>| <lprefc>| <user>

where the syntax rule for parmkind is non-standard in that no spaces are allowed between the base kind and any subsequent qualifiers.

Each state of each HMM must have its own section defining the parameters associated with that state

state = <State: Exp > short stateinfo

where the short following <State: Exp > is the state number. State information can be defined in any order. The syntax is as follows

stateinfo = ~s macro |
[mixes] [weights] stream { stream } [duration]
macro = string

A stateinfo definition consists of an optional specification of the number of mixtures, an optional set of stream weights, followed by a block of information for each stream, optionally terminated with a duration vector. Alternatively, ~s macro can be written where macro is the name of a previously defined macro.

The optional mixes in a stateinfo definition specify the number of mixture components (or discrete codebook size) for each stream of that state

mixes = <NumMixes> short {short}

where there should be one short for each stream. If this specification is omitted, it is assumed that all streams have just one mixture component.

The definition of each mixture component consists of a Gaussian pdf optionally preceded by the mixture number and its weight

mixture = [<Mixture> short float] mixpdf

If the <Mixture> part is missing then mixture 1 is assumed and the weight defaults to 1.0.

The tmixpdf option is used only for fully tied mixture sets. Since the mixpdf parts are all macros in a tied mixture system and since they are identical for every stream and state, it is only necessary to know the mixture weights. The tmixpdf syntax allows these to be specified in the following compact form

```
tmixpdf =     <TMix> macro weightList
weightList =  repShort { repShort }
repShort =    short [ * char ]
```

where each short is a mixture component weight scaled so that a weight of 1.0 is represented by the integer 32767. The optional asterisk followed by a char is used to indicate a repeat count. For example, 0*5 is equivalent to 5 zeroes. The Gaussians which make-up the pool of tied-mixtures are defined using ~m macros called macro1, macro2, macro3, etc.

Discrete probability HMMs are defined in a similar way

```
discpdf =     <DProb> weightList
```

The only difference is that the weights in the weightList are scaled log probabilities.

The definition of a Gaussian pdf requires the mean vector to be given and one of the possible forms of covariance

```
mixpdf =      ~m macro | [ rclass ] mean cov [ <GConst> float ]
rclass =      <RClass> short
mean =        ~u macro | <Mean> short vector
cov =         var | inv | xform
var =         ~v macro | <Variance> short vector
inv =         ~i macro |
              (<InvCovar> | <LLTCovar>) short tmatrix
xform =       ~x macro | <Xform> short short matrix
matrix =      float {float}
tmatrix =     matrix
```

In mean and var, the short preceding the vector defines the length of the vector, in inv the short preceding the tmatrix gives the size of this square upper triangular matrix, and in xform the two short's preceding the matrix give the number of rows and columns. The optional <GConst> gives that part of the log probability of a Gaussian that can be precomputed. If it is omitted, then it will be computed during load-in, including it simply saves some time. HTK tools which output HMM definitions always include this field. The optional <RClass> stores the regression base class index that this mixture component belongs to, as specified by the regression class tree (which is also stored in the model set). HTK tools which output HMM definitions always include this field, and if there is no regression class tree then the regression identifier is set to zero.

In addition to defining the output distributions, a state can have a duration probability distribution defined for it. However, no current HTK tool can estimate or use these.

```
duration =    ~d macro | <Duration> short vector
```

Alternatively, as shown by the top level syntax for a hmmdef, duration parameters can be specified for a whole model.

The transition matrix is defined by

```
transP =      ~t macro | <TransP> short matrix
```

where the short in this case should be equal to the number of states in the model.

Finally the input transform is defined by

```
inputXform =  ~j macro | inhead inmatrix
inhead =      <MMFIdMask> string parmkind [<PreQual>]
inmatrix =    <LinXform> <VecSize> short <BlockInfo>
              short short {short} block {block}
block =       <Block> short xform
```

where the short following <VecSize> is the number of dimensions after applying the linear transform and must match the vector size of the HMM definition. The first short after <BlockInfo> is the number of block, this is followed by the number of output dimensions from each of the blocks.

C.2 Word Network Definition Language

Word networks are specified using the HTK *Standard Lattice Format* (SLF). This is a general purpose text format which is used for representing word networks. Since SLF format is text-based, it can be written directly using any text editor. However, this can be rather tedious and HTK provides a tool which allows the application designer to use a higher-level representation. The tool *HParse* allows networks to be generated from a source text containing extended BNF format grammar rules.

Lattices in HTK are used for specifying finite state syntax networks for recognition. The HTK SLF is designed to be extensible and to be able to serve a variety of purposes. However, in order to facilitate the transfer of lattices, it incorporates a core set of common features.

An SLF file can contain zero or more sub-lattices followed by a main lattice. Sub-lattices are used for defining sub-networks prior to their use in subsequent sub-lattices or the main lattice. They are identified by the presence of a `SUBLAT` field and they are terminated by a single period on a line by itself. Sub-lattices offer a convenient way to structure finite state grammar networks. They are never used in the output word lattices generated by a decoder. Some lattice processing operations like lattice pruning or expansion will destroy the sub-lattice structure, i.e. expand all sub-lattice references and generate one unstructured lattice.

A lattice consists of optional header information followed by a sequence of node definitions and a sequence of link (arc) definitions. Nodes and links are numbered and the first definition line must give the total number of each.

Each link represents a word instance occurring between two nodes, however for more compact storage the nodes often hold the word labels since these are frequently common to all words entering a node (the node effectively represents the end of several word instances). This is also used in lattices representing word-level networks where each node is a word end, and each arc is a word transition.

Each node may optionally be labelled with a word hypothesis and with a time. Each link has a start and end node number and may optionally be labelled with a word hypothesis (including the pronunciation variant, acoustic score and segmentation of the word hypothesis) and a language model score.

The lattice must have exactly one start node (no incoming arcs) and one end node (no outgoing arcs). The special word identifier `!NULL` can be used for the start and end node if necessary.

C.2.1 Standar Lattice Format

The format is designed to allow optional information that at its most detailed gives full identity, alignment and score (log likelihood) information at the word and phone level to allow calculation of the alignment and likelihood of an individual hypothesis. However, without scores or times the lattice is just a word graph. The format is designed to be extensible. Further field names can be defined to allow arbitrary information to be added to the lattice without making the resulting file unreadable by others.

The lattices are stored in a text file as a series of fields that form two blocks:

- A header, specifying general information about the lattice.
- The node and link definitions.

Either block may contain comment lines, for which the first character is a '#' and the rest of the line is ignored.

All non-comment lines consist of fields, separated by white space. Fields consist of an alphanumeric field name, followed by a delimiter (the character '=' or '~') and a (possibly "quoted") field value. Single character field names are reserved for fields defined in the specification and single character abbreviations may be used for many of the fields defined below. Field values can be specified either as normal text (e.g. `a=-318 31`) or in a binary representation if the '=' character is replaced by '~'. The binary representation consists of a 4-byte floating point number (IEEE 754) or a 4-byte integer number stored in big-endian byte order by default.

The convention used to define the current field names is that lower case is used for optional fields and upper case is used for required fields. The meaning of field names can be dependent on the context in which they appear.

The header must include a field specifying which utterance was used to generate the lattice and a field specifying the version of the lattice specification used. The header is terminated by a line which defines the number of nodes and links in the lattice.

The node definitions are optional but if included each node definition consists of a single line which specifies the node number followed by optional fields that may (for instance) define the time of the node or the word hypothesis ending at that node.

The link definitions are required and each link definition consists of a single line which specifies the link number as well as the start and end node numbers

that it connects to and optionally other information about the link such as the word identity and language model score. If word identity information is not present in node definitions then it must appear in link definitions.

C.2.2 Syntax

The following rules define the syntax of an SLF lattice. Any unrecognised fields will be ignored and no user defined fields may share the first character with pre-defined field names. The syntax specification below employs the modified BNF notation used in section C.1. For the node and arc field names only the abbreviated names are given and only the text format is documented in the syntax.

```
latticedef = laticehead
             lattice { lattice }

latticehead = "VERSION=" number
              "UTTERANCE=" string
              "SUBLAT=" string
              { "vocab=" string | "hmms=" string |
                "lmname=" string | "wdpenalty=" floatnumber |
                "lmscale=" floatnumber |"acscale=" floatnumber |
                "base=" floatnumber | "tscale=" floatnumber }

lattice = sizespec
          { node }
          { arc }

sizespec = "N=" intnumber "L=" intnumber

node = "I=" intnumber
       { "t=" floatnumber | "W=" string  |
         "s=" string | "L=" string | "v=" intnumber }

arc = "J=" intnumber
      "S=" intnumber
      "E=" intnumber
      { "a=" floatnumber  | "l=" floatnumber  | "a=" floatnumber |
        "r=" floatnumber  | "W=" string | "v=" intnumber |
        "d=" segments }

segments =  ":" segment {segment}
segment =  string [ "," floatnumber [ "," floatnumber ]] ":"
```

C.2.3 Field Types

The currently defined fields are as follows:-

Field	abbr	o|c	Description
Header fields			
VERSION=%s	V	o	Lattice specification adhered to
UTTERANCE=%s	U	o	Utterance identifier
SUBLAT=%s	S	o	Sub-lattice name
acscale=%f		o	Scaling factor for acoustic likelihoods
tscale=%f		o	Scaling factor for times (default 1.0, i.e.\ seconds)
base=%f		o	LogBase for Likelihoods (0.0 not logs, default base e)
lmname=%s		o	Name of Language model
lmscale=%f		o	Scaling factor for language model
wdpenalty=%f		o	Word insertion penalty
Lattice Size fields			
NODES=%d	N	c	Number of nodes in lattice
LINKS=%d	L	c	Number of links in lattice
Node Fields			
I=%d			Node identifier. Starts node information
time=%f	t	o	Time from start of utterance (in seconds)
WORD=%s	W	wc	Word (If lattice labels nodes rather that links)
L=%s		wc	Substitute named sub-lattice for this node
var=%d	v	wo	Pronunciation variant number
s=%s	s	o	Semantic Tag
Link Fields			
J=%d			Link identifier. Starts link information
START=%d	S	c	Start node number (of the link)
END=%d	E	c	End node number (of the link)
WORD=%s	W	wc	Word (If lattice labels links rather that nodes)
var=%d	v	wo	Pronunciation variant number
div=%s	d	wo	Segmentation (modelname, duration, likelihood) triples
acoustic=%f	a	wo	Acoustic likelihood of link
language=%f	l	o	General language model likelihood of link
r=%f	r	o	Pronunciation probability

Note: The word identity (and associated 'w' fields var,div and acoustic) must appear on either the link or the end node.

```
abbr is a possible single character abbreviation for the
field name

o|c indicates whether field is optional or compulsory.
```

C.2.4 Building a Word Network with HParse

Whilst the construction of a word level SLF network file by hand is not difficult, it can be somewhat tedious. In earlier versions of HTK, a high level grammar notation based on extended Backus-Naur Form (EBNF) was used to specify recognition grammars. This *HParse* format was read-in directly by the recogniser and compiled into a finite state recognition network at run-time.

HParse format is still supported but in the form of an *off-line* compilation into an SLF word network which can subsequently be used to drive a recogniser.

A HParse format grammar consists of an extended form of regular expression enclosed within parentheses. Expressions are constructed from sequences of words and the meta-characters

| denotes alternatives

[ ] encloses options

{ } denotes zero or more repetitions

< > denotes one or more repetitions

<< >> denotes context-sensitive loop

The following examples will illustrate the use of all of these except the last which is a special-purpose facility provided for constructing context-sensitive loops as found in for example, context-dependent phone loops and word-pair grammars. It is described in the reference entry for *HParse*.

As a first example, suppose that a simple isolated word single digit recogniserwas required. A suitable syntax would be

```
(
   one | two | three | four | five |
   six | seven | eight | nine | zero
)
```

If this HParse format syntax definition was stored in a file called `digitsyn`, the equivalent SLF word network would be generated in the file `digitnet` by typing

```
HParse digitsyn digitnet
```

The above digit syntax assumes that each input digit is properly end-pointed. This requirement can be removed by adding a silence model before and after the digit

```
(
   sil (one | two | three | four | five |
   six | seven | eight | nine | zero) sil
)
```

The allowable sequence of models now consists of silence followed by a digit followed by silence. If a sequence of digits needed to be recognised then angle brackets can be used to indicate one or more repetitions, the HParse grammar

```
(
   sil < one | two | three | four | five |
   six | seven | eight | nine | zero > sil
)
```

would accomplish this.

HParse grammars can define variables to represent sub-expressions. Variable names start with a dollar symbol and they are given values by definitions of the form

```
$var = expression ;
```

For example, the above connected digit grammar could be rewritten as

```
$digit = one | two | three | four | five |
         six | seven | eight | nine | zero;
(
   sil < $digit > sil
)
```

Here `$digit` is a variable whose value is the expression appearing on the right hand side of the assignment. Whenever the name of a variable appears within an expression, the corresponding expression is substituted. Note however that variables must be defined before use, hence, recursion is prohibited.

As a final refinement of the digit grammar, the start and end silence can be made optional by enclosing them within square brackets thus

```
$digit = one | two | three | four | five |
         six | seven | eight | nine | zero;
(
   [sil] < $digit > [sil]
)
```

HParse format grammars are a convenient way of specifying task grammars for interactive voice interfaces. As a final example, the following defines a simple grammar for the control of a telephone by voice.

```
$digit  = one | two | three | four | five |
          six | seven | eight | nine | zero;
$number = $digit { [pause] $digit};
```

```
$scode  = shortcode $digit $digit;
$telnum = $scode | $number;
$cmd    = dial $telnum |
          enter $scode for $number |
          redial | cancel;
$noise  = lipsmack | breath | background;
( < $cmd | $noise > )
```

The dictionary entries for `pause`, `lipsmack`, `breath` and `background` would reference HMMs trained to model these types of noise and the corresponding output symbols in the dictionary would be null.

Chapter 7

Mimicry Attack Detection

7.1 Problem Description

To detect known attacks, MIDS rely on a classification mechanisms like expert systems or pattern matching. Regardless of the mechanism an MIDS utilises, it needs to recognise the attack from a known attack base. If an attack is substantially different from the signature stored in the attack base, then the detection mechanism becomes less accurate and in some cases useless.

There is a kind of attacks that *mimic* normal traffic to avoid detection by standard intrusion detection mechanisms. These attacks have been of interest for some time now, they are called mimicry attacks. A mimicry attack can be constructed by simply changing a given system call for other system calls that complete the same tasks with tespect to the goal of the attack. Or it can be more complex by inserting system calls that do not change the harmful state of the attack. We call the latter kind of system calls *no-ops*. By harmful state of an attack we mean the achievement of an attack goal whether it is to steal information, destroy information, gaining unprivileged access to a system or causing a denial of service. Two sequences are said to be equivalent up to harmfulness if they achieve the same attack goal, the attacker can not tell the difference between the two sequences in terms of the goal being achieved.

We divide mimicry attacks into two types: base case and general case. Base case mimicry attacks make only use of modified system calls and have a finite number of variations. General case mimicry attacks include the base case ones but also add no-ops, therefore the set of general case mimicry attacks is infinite. We consider an important class of the general case mimicry attacks which includes no-ops built out of system calls returning failure and some context-dependent no-ops. Syntactically it is not possible to use a filter that eliminates no-ops, because many no-ops are context dependent and hence are not identifiable. In a given context a system call can be a no-op while in others it is part of the attack.

For example assume a telnet session where the attacker is trying to execute

a buffer overflow using a vulnerability in program P_1. The attacker injects the buffer overflow into the victim's host and then executes it. The steps to achieve the desired goal are well defined. Now assume the attacker opens a socket connection in the middle of the attack to another machine and then closes the connection. The system calls generated during the socket operation will not affect the buffer overflow in other words the socket system call in this case can be regarded as a no-op. If the attack was not a buffer overflow but uploading information to other site, then a socket operation will be part of the attack. In that case the socket system call is not a no-op.

For a better understanding of mimicry attacks, suppose we have 10 system calls: *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I*, and *J*. Now suppose that the sequence *¡A, C, A, B, C¿* is **attack 1**. If *A* is equivalent up to harmfulness to *E* and *G* ($A \equiv E, A \equiv G$), and *B* is equivalent up to harmfulness to *F* ($B \equiv F$), then the sequence *¡E, C, G, F, C¿* is equivalent up to harmfulness to **attack 1**. Call this new sequence **attack 2** ($attack1 \equiv attack2$). If *H*, *I*, and *J* are no-ops then the sequence $< E, H, C, J, G, F, I, C >$ is also equivalent to **attack 1** and **attack 2**, we will call this **attack 3** ($attack1 \equiv attack2 \equiv attack3$).

We can appreciate that even though **attack 1**, **attack 2**, and **attack 3** are equivalent up to harmfulness, their form is very different. This difference explains the difficulty on detecting mimicry attacks using the signature of the attack.

To solve this problem we propose the use of a "noise" tolerant mechanism that is capable of detecting variations in the signature of an attack. To test for attacks, we first separate sequences of system calls from the log files into sessions. Each session is classified according to the service it belongs to. Our MIDS will parse each of these sessions and compare it to known attacks.

The output of the MIDS is a number that indicates how likely is that a session is an attack. This number is a probability function so it really indicates the similarity between the parsed session and a known attack. The output from the MIDS for each sequence **attack 1**, **attack 2**, and **attack 3** indicates the similarity between the sequence and a given attack. Since the three sequences are equivalent to the same attack the probabilities indicating that they are identical should be very high. We consider similar as any sequence with a likelihood of more than 95%.

The remainder of this chapter is organised as follows: §7.2 explains why we chose HMMs and word networks for mimicry attack detection. §7.4 gives a brief overview of mimicry attacks, both the base case and general case. §7.5 describes two of the attacks used as examples in the chapter. In §7.6 we describe training data for base case mimicry attacks. §7.7 describes our mechanism to test base case mimicry attacks, while §7.8 is a description of the validation data and experiments, and §7.9 presents validation results. §7.10 describes the data used for training in general case mimicry attacks. §7.11 the core of this chapter, introduces our mechanism, a word network based, misuse intrusion detector, while §7.12 describes validation experiments, and §7.13 which reviews results, sustains why it is good at mimicry attack detection. §7.14 further validates our method against mimicry attacks but using folded sessions (see chapter 5). §7.15

contrasts our detection method with others proposed in the literature. Some conclusions drawn from our investigation are discussed in §7.16.

7.2 HMMs and Word Networks for Mimicry Attack Detection

As described in chapter 6, HMMs is a probabilistic classification method that is well suited for sequence classification. When used for misuse detection, HMMs determine how similar an observed session and a known attack is. This is accomplished by training an HMM with a sequence of system calls describing a known attack. When a session is parsed by the trained HMM, the probability of that session being generated by the HMM indicates how similar the attack and the session are.

Other techniques for sequence pattern matching were considered, like stochastic context-free grammars which are more powerful versions of the HMMs, but the training and parsing times are prohibitively larger. Another alternative are Petri nets; however this approach is not well suited to create families of sequences, the transitions in a Petri net are deterministic and therefore variations in an attack might bypass the IDS. Bayesian networks can also be used, but they do not provide a mechanism to test for relations on the order of the sequence elements which is needed for sequence classification.

Other techniques like term rewriting can be used to ease the problem and facilitate even further the detection process by working with a unified set of system calls instead of using all possible variations for each scenario. However system call interchange does not always works straight forward, for example $A \equiv B$ and $C \equiv B$ but $A \not\equiv C$. An example for this is the system call `mmap` that can be used as a substitution for a `read` system call or even a `socket` system call. As in the previous section, there are some system calls that can be no-ops in some circumstances. But the same system calls are very useful to detect an attack under other circumstances, therefore rewriting these system calls as no-ops does not work directly.

In this chapter we show that MIDSs are capable of detecting mimicry attacks. We introduce a host-based MIDS, capable of successfully detecting a great variety of mimicry attacks. The method makes use of a *word network*. A word network is a technique conveniently solves a pattern matching problem by using a chain of smaller, noise-tolerant pattern matchers, thereby making it more tractable and robust. A word network is realised as a finite state machine, where every state is an HMM.

7.3 Word Networks

In speech recognition a word network is a directed graph where each node represents a single HMM that is able to detect a single word or even a phoneme, and the arcs are transition between the nodes [Young et al., 2002,

Pereira and Riley, 1997]. By using word networks, the grammatical structure analysis is independent of the phoneme detection process. Thus HMMs only detect phonemes or even whole words, regardless their position in a phrase. This process allows dealing with long silences or noises between phonemes while the word can still be recognised. Additionally, probabilities can be attached to the arcs making the graph more versatile.

The simplest word network has one pattern matching state and two silent states, one of the silent states is the start point and the other one is the ending point. Silent states are not linked with any HMM as the pattern matching states are. The transition graph is straightforward, not even a recursive transition is needed. This simple network works exactly the same as only using the HMM attached to the pattern matching state. A more complex network has many arcs departing from any given node, thus different paths can be tested. To calculate the probability of a sequence over a word network we probabilistically combine the output from each of its HMM nodes. A weight is assigned to each node transition to calculate the output of the network. The weight can be positive if we want to increment the output of the network or negative if we want to decrement the total output.

7.4 An Approach to Mimicry Attacks

We used the methodology proposed by Wagner and Soto [Wagner and Soto, 2002] to generate equivalent variations of a malicious sequence. These generated sequences are used to train and then test our IDS against mimicry attacks. The ability to create sequences equivalent up to harmfulness is based on the semantic equivalences inherent in the operating system.

Below is a description of possible changes to create mimicry attacks according to Wagner and Soto's methodology.

No op is a system call that has no effect on the state of the attack, by inserting no-ops the effects of the attack do not change. One way to select no-ops is to invoke a system call with invalid arguments. Any system call that takes as a parameter things like:

1. pointer,
2. memory address,
3. file descriptor,
4. `uid`,
5. `pid`,
6. `gid`

can be turned into a no-op by passing invalid arguments to it. Some system calls that can not be used as no-ops for instance `exit`, `pause`,

vhangup, `fork`, `alarm`, and `setsid` since they have a negative impact on the success of the attack as these system calls leave the current session, and following system calls will be executed in a new environment.

System Call Substitution is achieved by replacing successive calls to `read` with a single `read`. Another way is to substitute any call of `read` from an open file by a call to `mmap` followed by a memory access call [Wagner and Soto, 2002]. In an operating system like Solaris, there are new system calls that provide the same functionality as older system calls (or sequences of system calls). An example is the call `sysinfo` which is equivalent to `getdomainname`, `gethostid`, and `gethostname` if we provide appropriate parameters.

System Call Interchange, if one system call does not depend on the results of a previous system call they can be interchanged accordingly.

By dividing a session into segments, we can detect an attack enriched with no-ops, as long as the no-ops are present between the segments and not within them. We divided each attack in segments of size 6. The reason for choosing a size 6 is given by Tan and Maxion [Tan and Maxion, 2002]. They proved that a subsequence size of 6 is the minimal length in which at least one foreign subsequence exists. A foreign subsequence is a subsequence not found in the training data. All of the subsequences of size five and below exist in the training data. These demonstration was conducted on anomaly detection data, and proved that in order to detect anomalies, a sequence of at least six system calls should be analysed. The rationale for our research is that a mimicry attack can be regarded as an anomaly if compared against an attack base.

Consider Σ as the vocabulary of system calls x, $x_i \in \Sigma$ and $0 \leq i \leq n$ where n is the size of the vocabulary. The sets of n-grams with occurrence frequency ≥ 1 are defined as Σ^2 for bi-grams, Σ^3 for tri-grams, etc. And the n-grams for all anomaly subsequences are defined as Σ^2_A for bi-grams, Σ^3_A for tri-grams, etc. All subsequences of size two in Σ^3_A can be found in Σ^2. Or in other words, $\Sigma^2_A \subset \Sigma^2$, $\Sigma^3_A \subset \Sigma^3$, $\Sigma^4_A \subset \Sigma^4$, $\Sigma^5_A \subset \Sigma^5$, $\Sigma^6_A \not\subset \Sigma^6$. This was proved for the log files used by Forrest which are composed of sequences of system calls; we hope that the same results holds for the DARPA repository.

7.5 Eject and FFB Attacks

During this chapter we will use as examples two specific attacks for the Solaris operating system. The attacks can be found in the BSM log files of the DARPA repository. The two attack we will use are *eject* and *ffb*.

7.5.1 Eject Attack

The Eject attack exploits a buffer overflow in the `eject` utility distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject

button or removable media devices that are managed by Volume Management use the `eject` utility. Due to insufficient bounds checking on arguments in the volume management library, `libvolmgt.so.1`, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on the system under attack.

The *eject* attack can be executed by following these steps: i) inject the exploit script to the victim's host; ii) compile the exploit script; iii) execute the compiled exploit script; and iv) use the root console. If the exploit script is already in the victim's host and if it has been compiled, then the *eject* attack can also be executed as follows: i) execute the compiled exploit script; and ii) use the root console. Both versions achieve the same results using similar means (eject vulnerability), therefore the two versions of the attack are equivalent up to harmfulness. We will refer to the long version of the attack as *eject 1* and to the short version as *eject 2*.

7.5.2 FFb Attack

The ffb attack exploits a buffer overflow in the `ffbconfig` utility distributed with Solaris 2.5. The `ffbconfig` utility configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWffbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the ffbconfig program.

The attack follows an execution path similar to the *eject* attack. We also separate the *ffb* attack in two version: i) *ffb 1* injects, compiles and executes the exploit script; and ii) *ffb 2* executes the compiled exploit script.

7.6 Base Case Mimicry Attacks–Data Selection

To generate base case mimicry attacks, we used a table of equivalent system calls, a portion of this table is presented in table 7.2. The number of variations for some of the attacks from our attack base are shown in table 7.1. The number of possible attacks is given as the product of all the possible variations for each segment. We considered an average of 3 possible substitutions in case of `read`, `mmap` or `exec` system calls. Then the number of segments subject to modification is equal to the number of attack segments that include a modifiable system call. The maximum number of attacks is the number of possible values for a system call multiplied by the number of modifiable system calls.

From table 7.1 we can see that the possible number of attacks that can be generated by only modifying some system calls is staggering. If we were to detect all variations for an attacks using a simple matching mechanism, the time to search the entire database would be prohibitively large, let alone storage space.

The population for each segment is at most 3^6 (considering three variations for every system call in the segment). If we use equation (5.1) (rewritten below)

Table 7.1: Number of Variations for Each Attack

Attack Name	# of Segments	# of Segments Subject to Modification	Maximum # of Attacks that can be generated
Eject 1	106	72	3^{175}
Eject 2	38	22	3^{26}
FFB 1	32	17	3^{24}
FFB 2	44	15	3^{43}

to calculate the sample size n, assume a population size N and a desired error tolerance e, with a 2% error tolerance ($e = 0.02$), then we need 564 variations of the segments for a representative sample of the entire segment population of $N = 3^6$.

$$n = \frac{N}{1 + Ne^2}$$

By contrast, the entire attack population for the *eject 1* attack is $N = 3^{175}$, for a 2% error margin, we have a representative sample $n = 2500$. For a 0.5% error margin, a representative sample is 40,000 elements. For a population with over 20,000 elements, the size of the representative sample is always similar: it only variates with the desired error margin. Equation (7.1) is best suited for large populations. To train a single HMM for an entire attack, a sample size of 40,000 training sequences would be enough. If the attack is divided in segments of size 6, the sample size for each segment is smaller.

For example, for the first *eject* attack in table 7.1, each segment has an average of 14.43 variations. The total number of variations is 3^{175}, generated from 72 segments, so each segment has an average of $3^{175/72} = 3^{2.43} = 14.43$ variations. Using the same approach, with a sample size of 40,000 training sequences, we would have an average of 1.158 variations for each segment ($log_3(40000) = 9.6454$, and $3^{9.6454/72} = 158$). This number of variations is not enough to train an HMM and allow for generalisation.

By using another rule of thumb to select the training sample size we selected a third of the available population, that is 4.81 variations for each segment, to train each segment's HMM. With this sample size we allow for a better generalisation of the HMMs.

7.7 Base Case Mimicry Attacks–Detection Methodology

We followed a three step approach to train our IDS: i) we generated approximately 4.81 variations for each segment; ii) we trained one HMM for each segment using the generated variations; and iii) we generated the word network that combines all the segments.

Table 7.2: System Call Equivalences

Original System-Call		Equivalent System-Call(s)
memcntl	↔	mctl
read	↔	pread, readv, mmap
write	↔	pwrite, writev
open_read	↔	read
open_write	↔	write
execv	↔	execve, execvp, execl, execle, execlp
exit	↔	_exit
acl	↔	facl
chdir	↔	fchdir
chmod	↔	fchmod
chown	↔	fchown, lchown
stat	↔	fstat
brk	↔	sbrk

In order to generate attack variations, we considered only "one to one" substitutions (one system call is substituted by a single system call). In table 7.2 we show some of the equivalences used to generate base case mimicry attacks.

To train an HMM for a segment, we gather a selected number of variations for that segment and use them as a family. We call family a set of sequences that describe the same object, in this case a family is the set of sequences that describe an attack segment. Each HMM is saved as a separate file using the following naming convention: "name of the attack it detects" concatenated with a number i. The number i represents the position of the segment in the attack sequence. With the use of HMMs instead of a simple matching mechanism, the number of HMMs needed to describe the attack is equal to the number of segments of the attack it describes. For example, the `eject 1` attack, only needs 106 HMMs. Training time for 106 HMMs trained with sequences of size 6 and using up to 3^5 different sequences is about a minute. Total training time for an attack of n segments is in the worst case n minutes.

The topology for each segment's HMM is left-to-right HMMs (see §6.3). Each state in the HMM corresponds to a system call occurring in a sequence of size 6. The same transitions can be represented with an *ergodic* HMM with two states, where one state emits an output symbol when the parsed system call is part of the segment, and the other state emits a different output symbol when the system call is not part of the segment. Our experiments show that a left-to-right HMM is more precise in classifying our sequences. We tested both topologies and with a left-to-right topology the rate of false positives is lower (by 1%) and the detection rate is also higher (by 0.5%). An additional advantage of using a left-to-right topology is training and parsing time because this topology has less connections than the ergodic topology.

The generation of word networks for base case mimicry attacks is straightforward. We put the pieces of the attack together by linking the HMMs for every segment of the attack. Then by parsing the word network with an unseen sequence, we will know the probability that the sequence was generated by the word network.

7.8 Base Case Mimicry Attacks–Validation Experiments

All false positive tests were conducted using complete non-attack sessions of different services. We tested against 800 `telnet`, 1000 `smtp`, 50 `ftp` and 150 `finger` sessions. All the sessions were randomly picked from non-attack sessions of the BSM DARPA repository.

These non-attack sessions were combined with attacks generated with Wagner and Soto's methodology. We used 2500 variations of each attach as our validation sample. The attacks were generated using the first and second thirds of each segment variations. With a sample size of 2500 we have a representative sample with a 97.5% confidence.

7.9 Validation of Base Case Mimicry Attacks Word Network

Since the number of possible variations for each segment is finite, we used equation (5.1) to calculate the number of attacks needed for proper generalisation of the segment detectors. So the validation set for base case mimicry attacks is included in the training set of the HMMs. We used 100 variations of each attack to test detection rate. For false positive rates we tested each detector against 50 other attacks and against all normal sessions described above.

On average, detection rate is 98%, and the false positive rate is about 10%. By reducing the detection threshold the false positive rate also lowers. Initially we were using a 90% similarity measure. For a sequence of system calls, to be considered as an attack, the similarity between the sequence of interest and the training sequences should be at least of 90% (this is our similarity threshold), the similarly measure is given by the HMM. By increasing the similarity threshold to 95% the false positives reduced from 10% to 3% on the average. But the detection rate also lowered to 93%. Results summarised for each attack are shown in table 7.3.

It is worth mentioning that false positive detections arose out of sessions belonging to the same service. If an HMM is trained to detect a `telnet` attack then only `telnet` sessions will throw false positives. This can be explained with the work presented in chapter 6; when an HMM is trained to detect a `telnet` attack it is also trained to distinguish `telnet` sessions from sessions of other services. The reason for the high false positive is that the harmful elements

Table 7.3: Detection and false positive rate with different similarity measures. Similarity is indicated with an S and false positive with FP.

Attack Name	Detection % S=90%	FP % S=90%	Detection % S=95%	FP % S=95%
Eject 1	97%	9%	92%	2%
Eject 2	96%	8%	90%	3%
FFB 1	98%	10%	93%	4%
FFB 2	99%	8%	95%	1%
Loadmodule	97%	7%	91%	1%
Format 1	98%	9%	92%	4%
Format 2	96%	7%	89%	3%
Ftp-Write	99%	11%	96%	5%
warezclient	99%	12%	95%	5%
Satan	98%	9%	90%	2%
ipsweep	97%	10%	91%	3%

of an attack usually comprehend a small portion of the whole attack session; therefore the rest of the session used to train the HMM contains segments that can be found in non-attack sessions.

Thus the IDS will detect a normal session as an attack because it has similar segments. The order of these segments is important and a normal session and an attack have similar segments in the same order when they belong to the same service. The reason for this similitudes is that many system calls in a session are automatically generated by the service server. This will cause the HMM to detect a normal session as an attack.

7.10 General Case Mimicry Attacks–Data Selection

Word networks can be used for misuse intrusion detection by allowing no-ops to be treated as silences or noise. The rationale behind this is that an attack can be still detected even if it is divided in small segments and an HMM is used to detect each segment. Each segment's HMM has a probability associated with it. The joint probability of all the segment's HMMs in the word network indicates the probability that the sessions is an attack.

The transition between segments will indicate the occurrence of part of an attack. If the word network followed a path from beginning to end that resulted in a high probability for each of HMM corresponding to attack segments then these transitions will define how similar are the session and the attack.

Detecting general case mimicry attacks is more complicated than detecting base case mimicry attacks; in addition to interchanged system calls, the attacks are enriched with no-ops. Usually a system call returning a failure can be regarded as a no-op since it does not affect the state of the system, but there are

system calls that return success and can be turned into a no-op (like the socket example). Moreover, as shown in table 7.4, the same system call can be both a no-op and a useful system call for intrusion detection in the same sequence. In our example, we selected a sequence of four system calls from the eject attack with two `stat` system calls. The first `stat` system call is the execution of part of the *eject* attack (specifically address jump in the buffer overflow). The second is simply a result of a temporary file not found and can be considered a no-op since it does not affect the execution of the script exploit, or the script compilation or injection. Thus by filtering out system calls returning failure we risk to modify the signature of the attack. A filtering mechanism that tries to differentiate such system calls would need the arguments of the system calls which leaves us with another problem, if we consider as two different objects two executions of the same system call with different arguments, then the size of our vocabulary will grow as large as the number of possible arguments. A `readdir` system call with a directory name as its argument would produce a number of objects equal or greater than the number of valid directories in a system. So, to keep the number of objects tractable we consider only the name of the system call, the return value and in case of `exec` system calls the name of the program being executed. Wagner and Soto for example, state that none of the IDSs they analysed consider the return value of the system calls. By considering the return value of a system call we can identify some no-ops since some system calls that return failure can be considered no-ops. There are other system calls that can be inserted as no-ops like a *chdir* with argument ".". This system call will probably not modify the harmful state of an attack and will generate a non-failure system call trace. The very nature of what makes a no-op a no-op is what makes it so difficult to identify, and as a consequence, filter out.

Table 7.4: A Sequence of 4 system calls where system calls 1 and 2 return failure but are not no-ops and number 4 returns failure and is a no-op

System calls returning failure that are not no-ops (a buffer overflow attack)
1:open(2)␣-␣read|/etc/openwin|failure:␣No␣such␣file␣or␣directory
2:stat(2)|/export/home/alie/ÁÁÁÁÁ-n|failure:␣No␣such␣file␣or␣directory
3:execve(2)|ksh|success
System call returning failure that is a no-op
4:stat(2)|/tmp/115553|failure:␣No␣such␣file␣or␣directory

We considered sequences of size two to train no-op HMMs. Size two is the smallest sequence size which can be used in an HMM. This way an HMM with two states can revisit either state for a sequence of size three. If a two state HMM can parse sequences of size two and three successfully then with the use of word networks we can parse no-op sequences of any size just by iterating over no-op nodes.

Different nodes in a word network are used to test for occurrences of desired patterns or repetitions of unwanted patterns (noise). The no-ops used in

our experiments were mostly generated from system calls returning failure and some not returning failure. The number no-ops we used is 210. These no-ops can be safely inserted in an attack without modifying its harmfulness. But as demonstrated on table 7.4 we can not just filter them out.

In our experiments we consider a smaller, yet representative, class of general case mimicry attacks. In order to insert no-ops from system calls not returning failure, a complete understanding of the attack is needed. We need to test the effect of every inserted no-op and then decide if it is really a no-op. This decision depends on the position of the system call, and the state of the attack at insertion time. If the harmfulness of the attack is not affected then the system call then it is a no-op. Thus no-op creation is not a trivial problem.

The no-ops based on system calls returning failure conform to a set of approximately 210^2 bi-grams. The set of bi-grams is the result of combining all no-ops. Some of these bi-grams have to be removed from the set, specially those that have an occurrence in the attack segments. The removal is to avoid confusion when parsing in parallel an attack segment node, and a no-op node. After removing the bi-grams present in the attacks from this bi-gram set, the remaining elements are used to train no-op HMMs.

7.11 General Case Mimicry Attacks–Detection Methodology

We now have the building blocks for our word network. We have an HMM for each attack segment and also HMMs for a total of 210 possible no-ops. Each training sequence is a no-op followed by another no-op. So every HMM was trained with a total of 210 sequences (the same no-op can be used twice). We choose sequences of size two since this is the smallest usable for an HMM. Smaller sequences would not include transition probabilities, but only symbol emission probabilities (only 1 state).

Since *eject 2* is a shorter version of *eject 1* we can use the same word network to detect the both. *eject 2* attack consists of the execution of the exploit script which is 226 system calls long. *eject 1* consists of exploit script injection, exploit script compilation, and exploit execution; in this case the whole attack is 632 system calls long. Our word network must be able to follow both path of execution. Either it begins by exploit script injection, or it begins by exploit script execution.

Resulting probabilities from the HMM of each segment are weighted with positive values; whereas results from no-op nodes will be weighted with a small negative value as a penalisation. Then we calculate the joint probability of all the HMMs in the execution path of the word network. This joint probability indicates the similarity between the session of interest and the attack described by the word network.

In table 7.5 we show a piece of a grammar that generates a word network for an *eject* attack. The notation depicted in the table is the same used in HTK,

Table 7.5: A grammar corresponding to the eject attack word network.

$no-op	=	$no\text{-}op_1 \mid no\text{-}op_2 \mid \ldots \mid no\text{-}op_m$;
$code_insertion	=	$eject\text{-}s_1$ {$no-op} $eject\text{-}s_2$ {$no-op} ... $eject\text{-}s_i$;
$code_execution	=	$eject\text{-}s_j$ {$no-op} $eject\text{-}s_{j+1}$ {$no-op} ... $eject\text{-}s_n$;
([$code_insertion {$no-op}] $code_execution)		

it is not BNF notation (for a more detailed description of HTK notation refer to appendix C). In HTK format everything starting with $ is a variable, we created many variables to simplify the word network. Anything enclosed in { } means zero or more repetitions. Anything enclosed in [ ] is optional, zero or 1 repetition. $no\text{-}op_i$, and $eject\text{-}s_j$, ($1 \leq i \leq M$ and $1 \leq j \leq N$) correspond to the names of HMMs that identify such sequences.

Figure 7.1 is a graphical representation of the word network for the *eject* attack. Each node in figure 7.1 represents a word network. The first node is parsed depending on the session being *eject 1* or *eject 2* . The word network for exploit script injection and exploit script compilation is shown in figure 7.2. The word network corresponding exploit script execution is shown in figure 7.3. Each of these word networks use only one node for no-ops, this node is a word network representing all possible word no-ops; this word network is shown in figure 7.4.

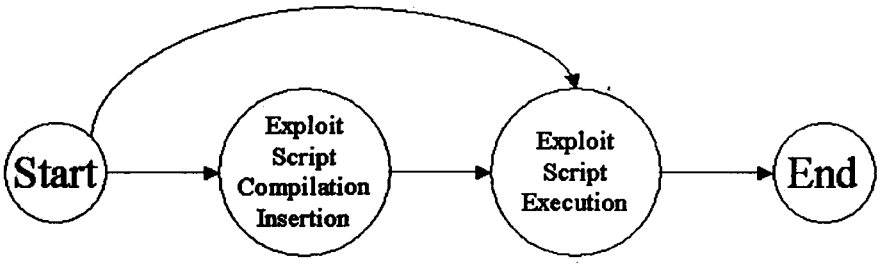


Figure 7.1: Word network for eject attack

As we demonstrated in chapter 5, it is possible to find repetitive subsequences. The same applies for attack segments. We found repetitive sequences of size 6 among the attacks. If we build an HMM for each available sequence of size 6, then we only need to build the word networks using HMMs from unique sequences of size 6 to detect an attack. Therefore the number of HMMs needed to describe an attack is equal or lesser than the total number of segments of the attack.

Since training data is sparse, training time is very reasonable. For a word network with 110 HMMs with up to 3^6 training sequences of 6 system calls for each HMM, and each HMM with 6 states, training time was about 25 minutes.

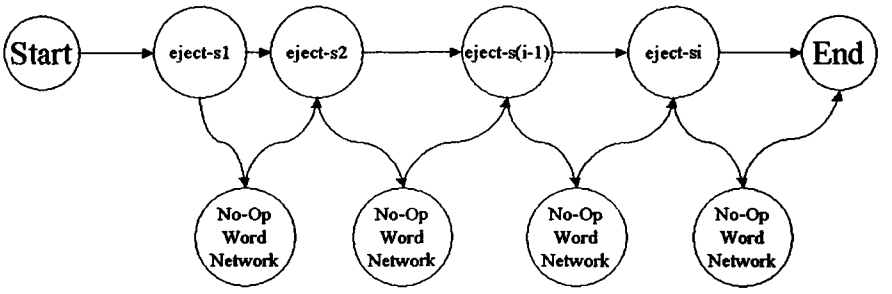


Figure 7.2: Word network for eject exploit script injection and compilation

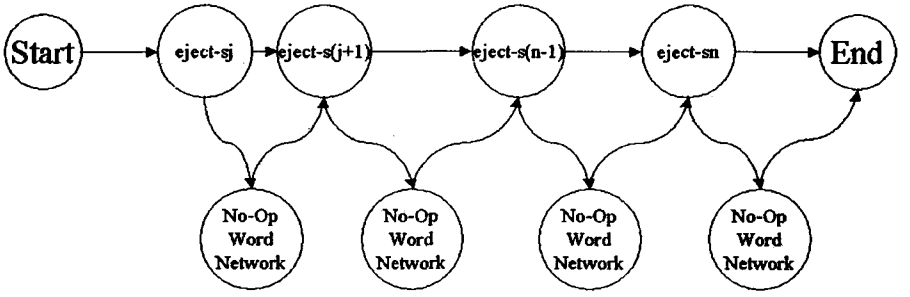


Figure 7.3: Word network for eject exploit script execution

We also tried the training with 12 states (two for each element in the sequence) for each HMM and the time it took to train was about 55 minutes. All the tests were made on a PIV HT @ 2.6 GHz with 1 GB of dual channel RAM @ 400MHz running Linux Mandrake 9.0.

7.12 General Case Mimicry Attacks–Validation Experiments

Since the set of possible attacks that can be generated by inserting no-ops is infinite, we limit ourselves to a subclass of general case mimicry attacks that contain at most 10 different randomly selected no-ops. For each variation of an attack we created 10 more variations by inserting the no-ops. This subclass is still important because it includes some context-dependant no-ops, and no-ops derived from system calls returning failure. In both cases a normalisation is not possible. With a data base of 2,500 variations for each attack, we obtain 25,000 different attacks for general case mimicry attacks. The attack segments were generated using the first and third sets of variation of segments (as defined

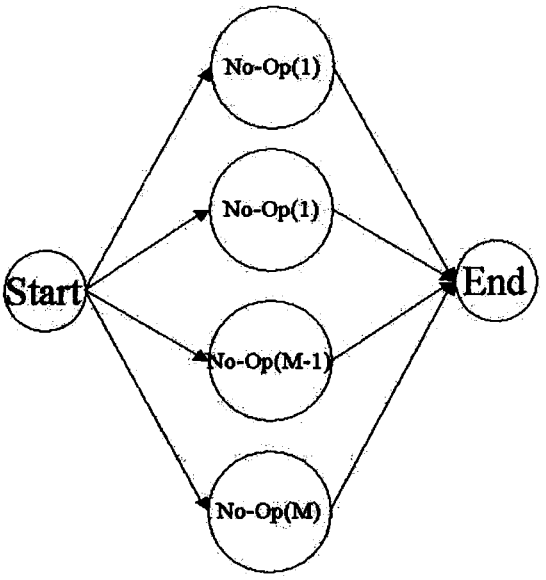


Figure 7.4: Word network for no-ops

in §7.6). 25,000 different attacks are a representative sample of the possible attacks with a 99.2% confidence. As in base case mimicry attacks validation, normal sessions were used to test for false positives.

The experiments consist of parsing each generated attack with each of the generated word networks. If the probability that a session was generated by an attack word network falls below a threshold, the session is considered as a non-attack session. Otherwise the session is considered as an attack.

7.13 Validation of General Case Mimicry Attacks Subclass

By using word networks as described in table 7.5 we extend mimicry attack detection to a subclass of general case mimicry attacks as defined in §7.10. The detection accuracy for this subclass is about 92%. The false positive detection rate is still high; about 4% of the sequences were wrongfully labelled as attacks. The increment on the false positive rate over modified attacks without inserted no-ops was to be expected. The noise that the no-ops insert to the model confuses the detector. In table 7.6 we summarise our results.

Table 7.6: Detection and false positive rate for general case mimicry attacks

Attack Name	Detection Rate	False Positive Rate
Eject 1	93%	4%
Eject 2	91%	2%
FFB 1	93%	2.8%
FFB 2	94%	3.2%
Loadmodule	91%	3%
Format 1	92%	2.5%
Format 2	89%	2%
Ftp-Write	93%	4%
warezclient	92%	3%
Satan	90%	4%
ipsweep	91%	5%

7.14 Reduction Impact on Intrusion Detection

This section aims to show that using n-gram reduction allows us to use HMMs with larger sequences than the ones proposed in previous works, such as [Yeung and Ding, 2003, Qiao et al., 2002, Warrender et al., 1999]. HMMs take a large amount of time for training. Wagner and Soto [Wagner and Soto, 2002], describe the disadvantages of using only short sequences as a detection base for HMMs. We used entire sessions containing the attacks for both, our training and testing data. We used a single word network for each attack.

We used 20 instances of each attack to train the HMMs. The tests were conducted against entire sessions of different services, in this case we used folded sessions. Again, we tested against 800 `telnet`, 1000 `smtp`, 50 `ftp` and 150 `finger` sessions.

7.14.1 Detection Results

By using reduction techniques as described in chapters 4 and 5 we obtained a 98% detection ratio and 23 false positives out of 200 detected attacks. With a higher similarity measure of 95%, false positives lowered from 23 to 14 and the detection ratio also lowered but only by 1%. We tested the same attacks in both scenarios. The difference in false positives was found in short attacks as `eject`. Most of the false positives were normal sessions labelled as one of these short attacks. Nevertheless, the higher detection ratio is due to the additional detection of variations of these same short attacks. We have successfully detected a significant subclass of general case mimicry attacks which is a great breakthrough since no other approach was able to do anything similar.

For all our experiments we used the "Hidden Markov Model Toolkit (HTK)". The software allows for large HMMs to be used and it also has the ability to use word networks. The software and its documentation

can be found at `http://htk.eng.cam.ac.uk/`. The HTK file format for HMMs and word networks is shown is appendix C and in our website at `http://webdia.cem.itesm.mx/ac/raulm/ids`.

7.15 Related Work

7.15.1 HMMs-Based Detection Methods

More recently, misuse intrusion detection has been complemented with anomaly intrusion detection. Following this approach, a user activity is compared against a profile of a user's normal behaviour; any disagreement is considered an anomaly, and thereby as an intrusion.

An anomaly IDS (AIDS) is usually implemented using a Hidden Markov Model (HMM). Warrender, Forrest and Pearlmutter's seminal paper reports on an HMM based AIDS with 96.6% of accuracy [Warrender et al., 1999]. Their method uses a sliding window, of size 6, with which they take an observation, a phrase, and then compare it with a profile of known, ordinary behaviour. The size of the window is the depth of the grammar; the smaller the size of the window, the more limited is the intrusion detector. Other researchers, for example [Qiao et al., 2002, Yeung and Ding, 2003, Tan and Maxion, 2002], have also explored the use of HMMs to intrusion detection, improving only slightly Warrender et al's results. More related to our work are approaches of Tan and Maxion [Tan and Maxion, 2002], and Wagner and Soto [Wagner and Soto, 2002]. The former have demonstrated that for an anomaly to be detected a minimal depth of the grammar equal to 6 should be used. The latter, have argued against this result, pinpointing the disadvantages of using a small grammar depth for intrusion detection. In particular, Wagner and Soto have demonstrated that using a sliding window of size 6 is insufficient to detect lots of mimicry attacks, therefore dismissing Warrender et al.'s approach. We used an approach of size 6 just as Tan and Maxion propose as our minimal detection block. Then we combined these these block with the use of word networks expanding the depth of the grammar by a size six for each segment of the attack. We provided evidence in this chapter that by using the proper encoding and detection methods, a MIDS is capable of dealing with mimicry attacks.

7.15.2 Mimicry Attack Detection Methods

On their seminal work, [Schonlau et al., 2001] analyse the performance of various mimicry detection methods, or as they call them *masquerade attacks*. They analyse six distinct detection methods: *Uniqueness*, *Bayes 1-Step Markov*, *Hybrid Multi-Step Markov*, *Compression*, *IPAM* and *Sequence-Match*. All these are anomaly detection methods, and use sets of user commands to build normal behaviour profiles. For a given number of commands a score is calculated and if the score is below certain threshold then the commands are considered as a mimicry attack. Below is a brief description of each method.

Uniqueness bases its scoring system on unpopular and uniquely used commands. It assumes that commands of these kind are the ones that most likely represent a mimicry attack. A mimicry attack will disguise itself as a normal sequence. This is the reason the approach yields a detection rate of only 40% and a good false positive ratio of 1.4%.

Bayes 1-Step Markov is similar to uniqueness but tests for transition probabilities between commands. It builds a profile with known probabilities and tests input for deviations on these probabilities. Again this approach tests for mimicry attacks as if they were anomalies. Detection ratio for this approach is 69.3% and false positive ratio is 6.7%.

Hybrid Multi Step Markov relies on a high order Markov Chain. The length of the command history is 10. The transition probabilities are calculated with the help of a *mixture transition distribution*. Only the most frequent commands are given attention, the rest is labelled as *other*, the most frequent commands cover 99% of the user's training data. If there are too many commands labelled as *other* in the analysed sequence, an independence model is proposed. Instead of calculating the occurrence probability, each command is treated as an independent event and the probability of a sequence is calculated by multiplying each command's probability. The approach to use for detection is selected automatically. Detection ratio is 49.3%, and false positive ratio is 3.2%

Compression is based on a reversible mapping that uses fewer bytes (much in the sense of our session folding method). The rationale behind this method is that data from a normal user will compress more readily than that of a mimicry attack. This approach, as we demonstrated in chapter 7, improves detection ratio but is not well suited for detecting mimicry attacks as a stand alone measure. Detection ratio is 34.2%, and the false positive ratio is 5%.

IPAM (Incremental Probabilistic Action Modelling) is based on a 1-step command transition (just as Bayes 1-Step Markov). The approach is similar to a bi-gram model. From the training data the probability of the next command is estimated. These probabilities are dynamically updated with an arbitrary value α ($0 < \alpha \leq 1$). On arrival of a new command, $1 - \alpha$ is added to the transition probability from the previous command to the most recent command. All other transition probabilities on the previous command are multiplied by α. A value of α of 0.9 is suggested. The four commands with higher transition probabilities are compared against the current command. If the command is present in the predictions it is labelled as *good*. The fraction of *good* transitions is used as the measure for anomalies. If it falls below a threshold an alarm is raised. Detection ratio is 36.8%, and false positive ratio is 3.7%

Sequence-Match computes a similarity measure between the 10 more recent commands and a user profile. The similarity measure is the count of

matches between the commands and the user profile of 10-grams. Detection ratio is 41.1%, and false positive ratio is 2.7%.

Another method, proposed by [Maxion and Townsend, 2002], uses naïve Bayes to estimate the probability that a command c can be issued by user u. This method builds a profile for a user, so-called self, from a set of training data. The self of other users is then taken as the user's non-self. This method shows a detection ratio of 61.5% and a false positive ratio of 1.5%.

An alternative way of approaching mimicry detection is proposed by [Scott et al., 2003]. The method is based on a widely used technique in the comparison of genetic material such as DNA, RNA, and other protein sequences. The method aims at detecting how well two sequences align one another and thus how similar they are. If enough no-ops are inserted in an attack sequence the alignment will test positive as if it belonged to a normal sequence. The technique assigns a small penalty for gaps occurring between normal user commands. The gaps are non-matching objects in the sequence. This allows to deal with inserted no-ops, but with a certain number of no-ops, the alignment score will be too low. Detection ratio for this approach is 75.8%, and false positive ratio is 7.7%.

A recursive data mining approach is proposed by [Boleslaw and Yongqiang, 2004]. It recursively extracts repetitive sequences, and replaces them by a new symbol until no dominant patterns are left. After the substitution, different features are extracted, such as the number of distinct patterns, the number of dominant patterns, the number of distinct dominant patterns, the length reduction or the number of distinct symbols. Then, a support vector machine is trained using the user patterns as negative examples and other user patterns as positive examples. This approach can also be used for misuse detection by using attack patterns for positive samples and user patterns for negative patterns. The authors report a detection ratio of 68% and a false positive ratio of 9%. However with a mimicry attack the patterns can resemble user patterns and therefore be wrongfully classified as a normal pattern.

These approaches all have the same limitation: they are all susceptible to overlook no-ops inserted along normal user commands. These approaches are good for detecting anomalies but have a poor performance on mimicry attack detection. By contrast, our approach is able to detect a wider class of mimicry attacks thanks to the use of word networks, which provide a way to sequence decomposition. Our approach detects many variations in the attack segments by means of the HMMs and connect all these segments with a word network which tolerates no-ops and still detects the attack as such.

7.16 Conclusions

We have successfully tested our IDS against a large number of the mimicry attacks that can be generated from the BSM DARPA repository. In our experiments, our mechanism has shown a 93% of effectivity for base case mimicry

attacks and 92% for general case mimicry attack detection. Thus, a false negative occurs at a rate of 7% and 8 % respectively. By contrast, the rate of false positive occurrences is a bit smaller: only 3% for base case mimicry attacks, and 4% for general case mimicry attacks.

Using the same technique in reduced log files yields interesting results. False positive ratio is only 1% higher than the one obtained with non reduced log files. Detection ratio is better using reduced log files.

The use of word networks proved to be very useful to detect attacks with inserted no-ops. As future work we propose to build word networks using full language models and to characterise usual behaviour in attacks such as a buffer overflows. A full language model would incorporate probabilities for node transitions in the word network and could be used to reduce the false positive rate.

Another area of improvement is to develop a method to test for no-ops inside the attack segments. Word networks might also be used to model normal behaviour for an anomaly IDS.

Our approach is an attempt at detecting a very elusive kind of attacks. Other techniques as term rewriting can be used in conjunction with our approach to test in order to alleviate the search space even further.