Characterization of Cyber Attacks Through Variable Length Markov Models

by

Daniel Schnetzer Fava

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Assistant Professor, Department of Computer Engineering Dr. Shanchieh J. Yang
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
July 2007

Approved By:

Dr. Shanchieh J. Yang
Assistant Professor, Department of Computer Engineering
Primary Adviser

Dr. Juan Carlos Cockburn
Associate Professor, Department of Computer Engineering

Dr. Leonid Reznik
Professor, Department of Computer Science

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Characterization of Cyber Attacks Through Variable Length Markov Models

I, Daniel Schnetzer Fava, hereby grant permission to the Wallace Memorial Library reporduce my thesis in whole or part.

____

Daniel Schnetzer Fava

____

Date

Dedication

Para o meu melhor amigo. Pelo seu grande exemplo como ser humano. Pelo seu coração, pela sua paciência, pela sua sabedoria e companheirismo. Por ter me proporcionado tantas oportunidades. Por ter ido muito além do seu papel como pai.

Acknowledgments

I want to express my gratitude to Dr. Shanchieh Jay Yang for his innumerable technical and non-technical advice, and most of all, for his friendship. I would like to thank Dr. Juan Cockburn and Dr. Leon Reznik for their inputs during our discussions, and for serving as members of the committee. I would also like to thank friends and colleagues from the Computer Engineering Department, especially Meng Jiang and Eric Bohannon for the conversations in which we shared our incipient and diverse insights about research, and Alexandru Samachisa for his feedback and for proofreading the thesis draft. I would also like to thank Dr. Andreas Savakis, the head of the Computer Engineering Department, for granting me an extension to work on this thesis during the Summer of 2007. I must also mention the student advisor and staff assistants: Ms. Anne DiFelice; Mrs. Pam Steinkirchner; and Mrs. Kathy Stefanik; laboratory manager and system administrator, Mr. Richard Tolleson and Mr. Paul Mezzanini.

Abstract

The increase in bandwidth, the emergence of wireless technologies, and the spread of the Internet throughout the world have created new forms of communication with effects on areas such as business, entertainment, and education. This pervasion of computer networks into human activity has amplified the importance of cyber security.

Network security relies heavily on Intrusion Detection Systems (IDS), whose objective is to detect malicious network traffic and computer usage. IDS data can be correlated into cyber attack tracks, which consist of ordered collections of alerts triggered during a single multi-stage attack. The objective of this research is to enhance the current knowledge of attack behavior by developing a model that captures the sequential properties of attack tracks. Two sequence characterization models are discussed: Variable Length Markov Models (VLMMs), which are a type of finite-context models, and Hidden Markov Models (HMMs), which are also known as finite-state models. A VLMM is implemented based on attack sequences $s = \{x_1, x_2, ...x_n\}$ where $x_i \in \Omega$ and Ω is a set of possible values of one or more fields in an alert message. This work shows how the proposed model can be used to predict future attack actions (x_{j+1}) belonging to a newly observed and unfolding attack sequence $s = \{x_1, x_2, ..., x_j\}$. It also presents a metric that measures the variability in attack actions based on information entropy and a method for classifying attack tracks as sophisticated or simple based on average log-loss. In addition, insights into the analysis of attack target machines are discussed.

Contents

List of Figures

List of Tables

List of Equations

Glossary

A

Apache Apache is a widely adopted Web server, p. 43.

API Application Programming Interface is a set of routines, protocols, and tools for building software applications, p. 38.

C

CDF Cumulative Distribution Function, p. viii.

D

DARPA Defense Advanced Research Projects Agency, p. 4.

DMC Dynamic Markov Chain, p. 19.

DoS Denial of service attack, p. 20.

Dragon Dragon is a distributed Intrusion Detection System (IDS), p. 43.

F

FTP File Transfer Protocol, p. 61.

H

HMM Hidden Markov Model, p. v.

I

IDS Intrusion Detections System, p. v.

IIS Internet Information Services is a set of Internet-based services for servers using Microsoft Windows, p. 43.

N

no-repetition Refers to a data set of sequences where consecutive characters do not repeat ($x_i \neq x_{i+1}$), p. vii.

O

OSI Open System Interconnection, p. 2.

R

repetition Refers to a data set of sequences where consecutive characters may repeat ($x_i = x_{i+1}$), p. 49.

S

SAX Simple API for XML is a serial access parser API for XML, p. 38.

Snort Snort is a protocol analyzer and Intrusion Detection System (IDS), p. 31.

T

top-1 Refers to the alert that is assigned the highest probability of happening given a context, p. 50.

top-2 Refers to the two alerts that are assigned the highest probability of happening given a context, p. 50.

top-3 Refers to the three alerts that are assigned the highest probability of happening given a context, p. 50.

V

VLMM Variable Length Markov Model, p. v.

VMware VMware is a proprietary virtualization software, p. 42.

X

XML Extensible Markup Language is a general-purpose markup language used to encapsulate information, p. 38.

Chapter 1

Introduction

Analog data has systematically been converted to digital format so that it can be cataloged, searched and reached by a broader audience than ever before. At the same time that data is being digitized, a wealth of information is created directly in digital format and made available to a wide public everyday.

More than ever before, the power of accessing and interpreting information has become a crucial asset. This is illustrated by the development of new communication mediums with higher bandwidth and lower latency, by the emergence of wireless technologies, and by the spread of the Internet throughout the world.

In the last 27 years, about 1 billion computers were manufactured, and market researchers expect that the next billion will be deployed in the next five years [67]. Meanwhile the number of documents on the Web has increased dramatically, from about 550 million in 2001 [7] to more than 11.5 billion in 2005 [23], so has the number of Internet users — from 40 million to 100 million in the US, and from 10 million to 45 million in Asia between 1997 and 2000 [52].

The growth of information in digital format and the pervasion of computer networks into human activity have amplified the importance of managing data access and the channels through which data flows. According to the United States Code (U.S.C.), which is a compilation and codification of the general and permanent United States federal law, information security is the process of protecting data from unauthorized access, use, disclosure, destruction, modification, or disruption [1]. Information and cyber security are multifaceted. They entail the provision of user accounts to password protect data, the encryption of communication mediums, the imposition of network access rules through firewalls, etc.

However, when all preventive measures fail, one would like to be warned about possible cyber security breaches. This is the idea behind Intrusion Detection Systems (IDS), which are sensors that look for traces of malicious activities in the cyber domain.

IDSs are usually based on pattern recognition or anomaly detection techniques. The pattern recognition approach identifies and stores signatures of known malicious activities. An alert is then raised when an observed event matches a signature in the pattern database. This approach is efficient when detecting known intrusions, but it often fails to detect novel ones.

On the other hand, anomaly detection works by establishing a profile for what is considered to be the system's normal activity. IDSs based on this approach raise an alert when there is a discrepancy between observed activities and the norm profile. Anomaly detection can often identify both novel and known attacks; however, it tends to produce a greater than desirable number of false positives.

IDSs are also categorized as network-based or host-based. Network-based IDSs monitor communication at the layers of the Open System Interconnection (OSI) model. Host-based IDSs, on the other hand, monitor host machines; for example, their services' and programs' performance, the state of their files and file system, their users' command pattern, etc.

As the size of networks and the number of IDSs deployed grew, network analysts and IT professionals faced an increasingly overwhelming number of alerts to be scrutinized. Therefore, it became necessary to create correlation engines to generate comprehensive reports from raw alert data. Previous works in the area of cyber security have emphasized the creation of IDSs and the development of these correlation engines.

Attack tracks are one possible byproduct of running a correlation engine on raw IDS alerts. These tracks consist of ordered collections of alerts belonging to a single multi-stage attack. This thesis proposed a model for attack tracks and investigates attack behavior in light of the proposed model. Since the ordering of attack actions within an attack is of great interest, two main approaches to sequence characterization are presented in Chapter 2. These approaches, called finite-context or Markov Models, and finite-state or Hidden Markov Models (HMM), have been used in many contexts, including the cyber security domain. Chapter 2 contrasts how these models have previously been used against what is proposed in this thesis work.

Chapter 3 discusses the information contained in different IDS alert fields and shows how sequences can be created from attack tracks. As it will be shown, different aspects of an attack track can be emphasized depending on the alert fields used to represent it. Chapter 3 also details how the model is constructed from several representative attack sequences. Once the model has been created, it can predict possible future attack actions based on a history of observed attack events. Section 3.2.1 shows how a future alert (x_{j+1}) is inferred from a newly observed and unfolding attack sequence $s = \{x_1, x_2, ..., x_j\}$. This chapter also presents a metric that measures the variability in attack actions based on information entropy and a method for classifying attack tracks as sophisticated or simple based on average log-loss.

Results and observations are given in Chapter 4. Chapter 5 summarizes the work performed and suggests extensions to this research. This thesis takes one step towards comprehending cyber attack behavior. The insights derived here can be beneficial to systems that assess the impact of and react to cyber security breaches. By adding to the body of knowledge related to cyber attacks, we hope to contribute to the solution of some of the issues described in the beginning of this chapter.

Chapter 2

Background and related work

2.1 Cyber attack data-sets

The purpose of this thesis is to study cyber attacks with the intent of enhancing the current knowledge about attack behavior. Finding a suitable data-set for this work is not a simple matter. An ideal data-set for the proposed work would contain cyber attack tracks. As discussed in Chapter 1, attack tracks are sequences of alerts generated by Intrusion Detection Systems (IDS) that get correlated by software. However, most of the research in cyber security has focused on identifying anomalies and clustering alert messages; therefore, most of the data-sets available are composed of uncorrelated alert with little or no ground truth in terms of attack tracks. These include sets from the MIT Lincoln Lab [31] [43], KDD Cup 99 [13], Defcon [10], and the Cyber Panel Correlation Technology Validation [24], which are briefly described next.

In 1998, the Defense Advanced Research Projects Agency (DARPA) sponsored MIT Lincoln Laboratory to produce the first attack data-set open to the research public. It was composed of several types of attacks, including port scans, remote compromises, local privilege escalation attempts and denial-of- service. They were carried out on a testbed network with relevant audit events being collected from network traffic and OS records. The success of the 1998 data inspired the creation of new sets in 1999 and 2000, which incorporated Windows NT attacks and multistep scenarios. The KDD Cup 99 data is similar to the MIT sets. It was created by the Third International Knowledge Discovery and Data Mining

Tools Competition in conjunction with the Fifth International Conference on Knowledge Discovery and Data Mining. These data-sets were created with intrusion detection in mind; therefore, they are composed of uncorrelated alerts mixed with normal background traffic. Since this thesis expects correlated attack tracks, the MIT and the KDD data-sets cannot be used as inputs.

Defcon sponsors a yearly hacking competition called Capture The Flag. Network traffic is recorded during the competition, and the data is made available. This set has a particularly high concentration of attack tracks when compared to background noise; however, the data is also uncorrelated and is not suitable for the research work proposed here.

A 2002 data-set created by the Cyber Panel Correlation Technology Validation (CPCTV) effort was encountered during the writing for this thesis. The objective of the data-set was to test correlation engines that perform high-level attack analysis. The data was produced by several runs of 14 different attacks divided into four classes — denial-of-service, data theft, Web defacement, and backdoor installation — along with simulated background traffic. Even though it was not used during this thesis work, this data set may be suitable for cyber attack characterization because it was built with high-level attack analysis in mind. However, as with the data-sets described previously, the CPCTV data would also have to be correlated into attack tracks before it could be used as input to the model proposed in this thesis.

Differently from the data-sets described previously, an ideal set for this thesis work would be composed of ordered collections of alerts belonging to a single multi-stage attack. The objective then is to capture the sequential properties of an attack track and to understand how different actions are put together by an attacker.

2.2 Sequence modeling

After a suitable attack set is found, tracks are modeled as finite sequences s with alert fields being mapped into an alphabet Ω:

$$s = \{x_1, x_2, ...x_n\},\ x_i \in \Omega \tag{2.1}$$

where Ω is composed of symbols representing unique attack actions.

The idea of modeling attacks as finite sequences of symbols is similar to how the DNA is represented as strings composed of permutations of the four fundamental basis A, G, T, C. The sequential property of the DNA has been studied with great extent; therefore, by representing attack actions as strings one can take advantage of the existing tools created for the analysis of sequences.

For example, computation biologists have looked into information theory and text compression when trying to determine the entropy of DNA sequences. “Since the alphabet of DNA and mRNA have four symbols, if these sequences were totally random, it would take two bits per symbol to represent [them]. However, only a small fraction of DNA sequences result in a viable organism; therefore there are constraints on those sequences” [37]. By compressing a sequence, one arrives at an upper bound for its entropy. “Common compression algorithms, such as Huffman, and various Lempel-Ziv based algorithms fail to compress DNA sequences at all, and low order arithmetic encoding algorithms only compress to around 1.96 bits per base” [37]. Vast efforts have been placed to characterize DNA sequences; for example, [37] proposes new approaches to estimate DNA entropy, and [22] studies local and global sequence alignment and methods for comparing and measuring similarities between DNA sequences.

If attack tracks are interpreted as sequences, the research on cyber security can be leveraged by borrowing from the theory developed for DNA analysis. At the same time, studying the entropy of cyber attack tracks will help us understand the variability in attack behavior. Studying measurements of similarity between attack sequences may allow clustering according to behavior.

Two main models can be applied to the characterization of sequences:

- Finite-context model, also known as Markov Models

- Finite-state models, also known as Hidden Markov Models or HMMs

Finite-context models assign a probability to a symbol based on the context in which it appears. On the other hand, finite-state models are more complex; they are composed of an observable part called "events," and a hidden part called "states." A state stores information about the past since it reflects changes in the system from the start to the present moment. A transition indicates a state change and is described by a condition that needs to be fulfilled in order to enable the transition. Events are observed with different probability distribution depending on the state of the system. These two models are discussed in Sections 2.3 and 2.4.

Different from the DNA, whose structure is well captured as sequences of four fundamental basis, there is not a well defined representation for cyber attacks. Therefore, before finite-context and finite-state models can be applied in the cyber security domain, one needs to convert attack tracks into sequences. One of the thesis objectives is to study this conversion. Chapters 3 and 4 discuss the trade-offs between different alphabet choices when mapping attack tracks to character sequences.

2.3 Finite-context models or Markov chains

Finite-context models predict a future outcome based on previously observed events.

2.3.1 First order finite-context models

In a first order finite-context model, the next state depends on the immediate previous state. Therefore, the transition probability is defined as:

$$P_{i,j} = Pr[\text{system in state j at t} + 1|\text{system in state i at t}] \tag{2.2}$$

A first order finite-context model for a system with n distinct states can be represented by the following transition probability matrix:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,s} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,s} \\ \cdots & \cdots & \cdots & \cdots \\ p_{s,1} & p_{s,2} & \cdots & p_{s,s} \end{bmatrix} \tag{2.3}$$

where $\sum_{j=1}^{s} p_{i,j} = 1, \forall\ i$ and s is the total number of states.

The initial probability distribution is represented by the vector Q:

$$Q = [q_1, q_2, ..., q_s] \tag{2.4}$$

where q_i is $Pr[\text{system is in state i at time } 0]$.

The probability that a sequence of states $X_1, X_2..., X_t$ will occur is given by the product of the probability for the initial state q_{x_1} and the transition probabilities p_{x_{t-1},x_t}:

$$P\{X_1, X_2, ..., X_t\} = q_{x_1} \prod_{t=2}^{T} p_{x_{t-1},x_t} \tag{2.5}$$

The initial probability distribution and the transition probability matrix can be created from observed data. Let N be the total number of observations, N_i be the total number of observations belonging to state i, and $N_{i,j}$ be the total number of transitions from state i to j; then:

$$q_i = \frac{N_i}{N} \tag{2.6}$$

$$p_{i,j} = \frac{N_{i,j}}{N_i} \tag{2.7}$$

The training data used to create initial probabilities and transition probabilities may not contain a certain state i or a transition from i to j, thus assigning them the probabilities of zero. This phenomena is known as the *zero frequency problem* [6]. Two methods for smoothing probabilities can be employed when all possible outcomes should receive non-zero probability values.

The first approach described in [14] adds a small proportion of the total number of states ($N_i + \epsilon \times N$) to the state count for i. The total number of states becomes N plus the additional $N \times \epsilon$ added to each state. Since there are s unique states, the total number of states becomes $N + N \times \epsilon \times s$ and the new state probabilities become:

$$q_i = \frac{N_i + \epsilon \times N}{N(1 + \epsilon \times s)} \tag{2.8}$$

Similarly, the transition probability can be smoothed by adding $\epsilon \times N_j$ to the transition count from i to j and increasing the total count proportionally:

$$p_{i,j} = \frac{N_{i,j} + \epsilon \times N_j}{N_i(1 + \epsilon \times s \times N_j)} \tag{2.9}$$

where ϵ is some small $\epsilon > 0$.

The second approach to smoothing probabilities is called *adaptive probability estimation*. In this case, each state or state transition receives an equal probability of occurrence at $t = 0$. These initial probabilities are updated as states are observed [45].

For example, assume a six state system. The initial state probability distribution is $q_i = 1/6 \; \forall \; i$ at $t = 0$. Then assume that the following sequence is observed:

$$q_1, q_1, q_1, q_5, q_5, q_3, q_1, q_4, q_1, q_6$$

At $t = 1$, 2 and 3, the numerator for q_1 is incremented along with the denominator for all q. At time $t = 4$ and 5, the numerator for q_5 is incremented and so on. At the end, all probabilities will be non-zero as shown in Table 2.1.

2.3.2 Higher order finite-context models

Different from the first order finite-context model shown previously, higher order models relate an event with its previous n observables. In an n^{th} order finite-context model, future events are dependent on n previous states as given by the following definition:

$$P\{X_{t+1} = x_{t+1} | X_{t-n+1} = x_{t-n+1}, ..., X_t = x_t\} \tag{2.10}$$

		Probability for states					
Time	Observed state	q_1	q_2	q_3	q_4	q_5	q_6
0	–	1/6	1/6	1/6	1/6	1/6	1/6
1	q_1	$\boxed{2/7}$	1/7	1/7	1/7	1/7	1/7
2	q_1	$\boxed{3/8}$	1/8	1/8	1/8	1/8	1/8
3	q_1	$\boxed{4/9}$	1/9	1/9	1/9	1/9	1/9
4	q_5	4/10	1/10	1/10	1/10	$\boxed{2/10}$	1/10
5	q_5	4/11	1/11	1/11	1/11	$\boxed{3/11}$	1/11
6	q_3	4/12	1/12	$\boxed{2/12}$	1/12	3/12	1/12
7	q_1	$\boxed{5/13}$	1/13	2/13	1/13	3/13	1/13
8	q_4	5/14	1/14	2/14	$\boxed{2/14}$	3/14	1/14
9	q_1	$\boxed{6/15}$	1/15	2/15	2/15	3/15	1/15
10	q_6	6/16	1/16	2/16	2/16	3/16	$\boxed{2/16}$
11	–	6/16	1/16	2/16	2/16	3/16	2/16

Table 2.1: Adaptive probability estimation table

When predicting x_{t+1}, $\{x_{t-n+1}, ..., x_t\}$ is called an *n-order context*.

Shannon proposes an interesting application of higher order models in order to model characters in the English language. The frequencies of alphabet letters are not uniform but range from about 13% for 'E' to 0.1% for 'Q' and 'Z,' the frequencies of pairs of letters are clearly not uniform either: 'Q' is always followed by 'U,' and the most frequent pair is 'TH' with a frequency of 3.7%. Proceeding in this way, higher order conditional probabilities can be estimated.

As proposed by Shannon, the minus-one order approximation is simply a sequence of random characters with equal probabilities. In the zero order approximation, each character occurs with the same frequency as it occurs in English. In the first order, each letter depends on its previous neighbor. In the second, each character depends on the two previous letters and so forth [44]. Table 2.2 was created based on this information. It shows that the higher the order, the closer it resembles English.

For the minus-one order approximation of English, the entropy of a character (H_{-1}) is at its maximum—all 26 letters, space and period have equal probability of occurrence. Entropy monotonically decreases at higher orders since more complex models are able to

Order	Approximation
-1	tsn rdjpdzfigypmawaknodjwxltaqmucoxweoefi jyfpxsdawbircocrlut sfptroy efbubxiebbb gbiocwfbsgi fafqqalf
0	yos tmhota i n sshntletnt anootnrnoPtoeeIc kwe hehq ith t. oeitai s oclcinryctaet. risonalven atia worgumfi. wleos enn
1	e obutant tnwe o Mar thionas pr is withious wid watout iofo inityrs ivasto tie w tapertibisthe te tsphan wiestime a
2	Thea se thook, somly. Let ther of mory. A romensmand codun shate sed be lat throphignis de thand sion le wentem macturt
3	The generated job providual better trand the displayed code, abovery upondults well the coderst in thestical i do hock bothe merg. (Instates cons eration. Never any of puble and to theory. Evential callegand to elast benerated in with pies as

Table 2.2: Shannon's approximations of English

capture more information about the structure and the regularity in the English language:

$$H_{-1} = \sum (1/28 \times \log_2(1/28)) \tag{2.11}$$

$$H_n = \sum_{x_1, x_2, ..., x_n} p(x_1, x_2, ..., x_n) \times \log_2(p(x_n | x_1, x_2, ..., x_{n-1})) \tag{2.12}$$

There are $28^4 = 614,656$ four letter combinations in an alphabet composed of 26 letters, space, and period. The least frequent yet valid four letter combination may happen with chances much smaller than once in every $614,656$ characters. Thus, in order to create a probability table for the fourth order model, one would need a training data much larger than 28^4 characters. In fact, the higher the order of approximation, the larger the training data must be. For example, it has been estimated that the largest library in the United States—the Library of Congress—would barely contain enough data necessary to create a 9^{th} order approximation of English [44]. The size of the alphabet also imposes constraints on the training data size. If the alphabet is, for example, 200 characters (200 valid states), then a fourth order approximation would require a training data much larger than 200^4.

2.3.3 Variable Length Markov Models (VLMM) and blending

Finite-context models of different orders can be blended into what is called a Variable Length Markov Model (VLMM). As discussed in Section 2.3.2, larger data-sets are needed for the creation of higher order Markov Models. One immediate consequence of the blending performed by the VLMM is that these requirements are relaxed as lower order models are emphasized in the absence of data for the creation of accurate higher order models.

Given that several order Markov Models exist, and given an unfolding sequence of events, a match for the n most recent observables is searched for at the n-order finite-context model. The $n - 1$ most recent events are matched against the $n - 1$ order model. This process is repeated until a minus-one order model is reached. All these results are blended into the VLMM through the weighted sum:

$$p(\phi) = \sum_{o=-1}^{m} w_o \times p_o(\phi) \tag{2.13}$$

where $p_o(\phi)$ is the probability of character (or event) ϕ happening at a certain context according to a model of order o. When blending, the emphasis on different order models is performed through the weights w_o.

Weights among different contexts of the same order

A simple VLMM could attribute fixed weights to each model order. Assuming that the system runs online; that is, it continuously computes probabilities as sequences are observed, then an improvement to the fixed weight approach is to attribute greater weights to higher order models after many sequence events have been observed. This would satisfy higher order model's need for more statistics.

A more sophisticated way of attributing higher weights after enough observations have been made is to take context frequencies into account. A long context that has high frequency count should be attributed greater weight than an equally long context that has low frequency. The low frequency count indicates that insufficient statistics have been collected for that context.

Escape probabilities

The previous discussion described how weights should be attributed to different contexts of the same order. A discussion on how weights should be attributed to a given context and its suffixes (sub-contexts) follows.

Smaller contexts are a subset of larger ones; therefore, they necessarily have a larger frequency count in the data. For example, given the sequence below:

$$\boxed{\text{XXXXXX}}\boxed{\text{OO}}A\boxed{\text{OO}}A\boxed{\text{OO}}B\boxed{\text{OO}}C\boxed{\text{OO}}D\boxed{\text{XXXXXX}}\boxed{\text{OO}}$$

A long context is represented as $\boxed{\text{XXXXXX}}\boxed{\text{OO}}$, and it is followed by A. There are also occurrences of the subset context $\boxed{\text{OO}}$ being followed by A, B, C, and D. The contrast in frequency count of characters following $\boxed{\text{OO}}$ and $\boxed{\text{XXXXXX}}\boxed{\text{OO}}$ shows that the longer context may be too restrictive because it only accounts for A and is oblivious to characters B, C, and D.

Escape probabilities work by "[allocating] some code space in each model to the possibility that a lower-order model should be used to predict the next character. The motivation for this is to allow for the coding of novel characters in a particular model by providing access to lower-order models" [6]. This notion is captured by Equation 2.14.

$$\begin{aligned} w_m &= (1 - e_m) \\ w_n &= (1 - e_n) \times \prod_{i=n+1}^{m} e_i, \; -1 \leq n < m \end{aligned} \tag{2.14}$$

Three different methods for computing escape probabilities, called A, B, and C, are discussed next.

Method A

This escape probability allocates one additional count over and above the number of times the context has been seen to the possibility that a lower order context should be taken into account:

$$e_n = \frac{1}{C_n + 1} \tag{2.15}$$

Method B

Here, the escape probability is proportional to q_n, which is the number of different characters that have occurred in some context of order n.

$$e_n = \frac{q_n}{C_n} \tag{2.16}$$

Method C

Similar to method B, the escape probability still increases with the number of different characters in the context but needs to be a little smaller to allow for the extra code space allocated to the characters:

$$e_n = \frac{q_n}{C_n + q_n} \tag{2.17}$$

2.4 Finite-state and Hidden Markov Models (HMM)

When describing finite-context models, ‘states,’ ‘events,’ ‘characters,’ and ‘symbols’ are often used interchangeably. However, in the case of the finite-state models, many different events can occur given a particular state. In an HMM, states are hidden; that is, they cannot be observed directly. Events, on the other hand, are the observable part of the model. The relationship between states and events is that events happen with different probability distributions depending on the state of the system.

2.4.1 Motivation for finite-state models

The motivation for finite-state models comes from the fact that some systems are poorly captured by finite-contexts. For example, consider the four possible bases, A, C, G, and T, along a string of DNA. Each of the 64 combinations of base triplets can be used to determine how amino acids are going to be chained in a protein. "The probability of one of the bases occurring is strongly influenced by its position within the triplet—first, second, or third. [One] way to find out the position is to count back to the start of the DNA sequence and decide at which position the cycle started. Knowing the previous base or even a number of previous bases does not help much in finding out what the current triplet alignment is" [6]. Therefore, a n-order finite context model that looks back at n basis would not accurately capture the structure of the DNA.

In other words, a finite context-model predicts an event based on the past n observations. It can be shown that systems where "counting" occurs cannot be captured by n-tuples. The generative rule in Equation 2.18 is an example of a system with counting.

$$(\mathrm{a} * \mathrm{b}\ \mathrm{a} * \mathrm{b}\ \mathrm{c}) * \tag{2.18}$$

'a,' 'b,' and 'c' are observables and '*' means repeated an arbitrary number of times (zero or more). A possible string generated with this rule is:

a b a b c a a a b a a a a a b c a a b a b a c a a a a b a a b c

As noticeable from the example string, event 'c' is generated after the second, fourth, sixth, eighth... 'b' has happened, while events 'b' are separated by arbitrarily long sequences of events 'a'. A finite-context model will always be incomplete since arbitrarily long contexts are needed to capture the structure of this sequence. However, the simple three state finite-context model shown in Figure 2.1 is able to capture Rule 2.18.

It is interesting to observe the entropy of the different order optimal finite-state models for Rule 2.18. Figure 2.2 shows the entropy of the first, second, and third up to 100^{th} order finite-state model when an optimal configuration is trained given an 100 character sequence

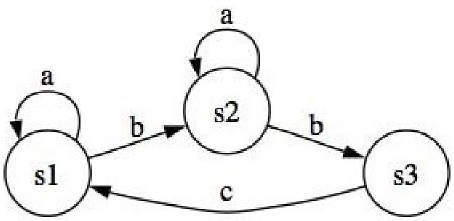

Figure 2.1: Three-state finite-context model for rule (a*b a* b c)*.

generated with Rule 2.18. One interesting characteristic of this graph is the abrupt decrease in entropy from the second to the third order. This abrupt decrease is caused by the fact that the third order model shown in Figure 2.1 accurately captures the rule that generates the sequence. Additional increases in order will cause a decrease in entropy at the expense of generality: the higher order models are over fitting to the particularities of the training sequence. A model with the same number of states as the number of observed events will have entropy zero; however, it has no prediction value since it merely duplicates the training sequence.

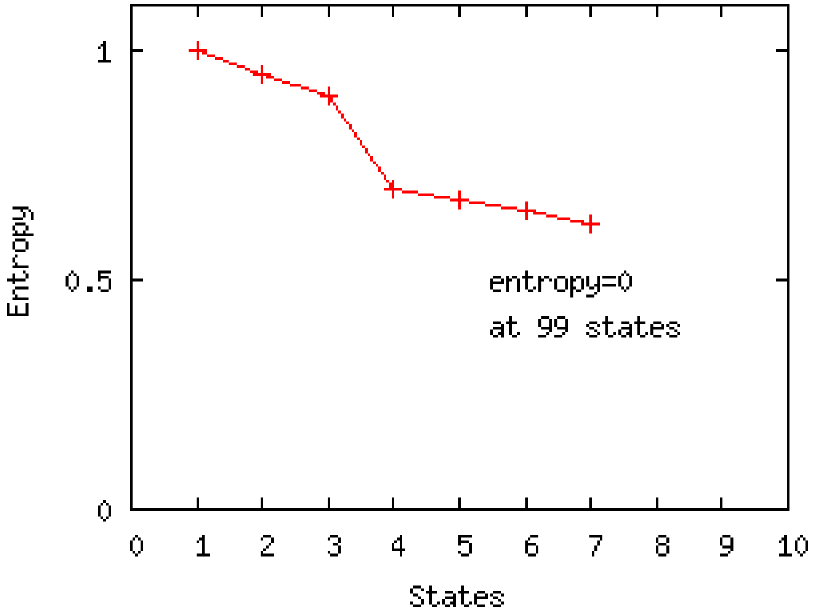

Figure 2.2: Entropy versus number of states — finite-context model.

2.4.2 Building an HMM

A finite-state model is characterized by the following [50]:

1. N, the number of states in the model. Individual states are denoted as $S = \{S_1, S_2, ..., S_N\}$, and the state at time t as q_t.

2. M, the number of distinct events per state. M is also known as the discrete alphabet size; that is, the number of distinct observables or the physical output of the system being modeled. Individual events are denoted as $V = \{v_1, v_2, ..., v_M\}$.

3. The state transition probability distribution $A = \{a_{ij}\}$ where

$$a_{ij} = Pr[q_{t+1} = S_j | q_t = S_i],\ 1 \leq i,\ j \leq N \tag{2.19}$$

   States may be connected in different topologies. An ergodic model is a special case where any state can reach any other state in a single step. In this case, $a_{ij} > 0$ for all i, j.

4. The observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where

$$b_j(k) = Pr[v_k \text{ at } t | q_t = S_j],\ 1 \leq j \leq N,\ 1 \leq k \leq M \tag{2.20}$$

   that is, $b_j(k)$ means the probability of observing event v_k when the system is at state S_j.

5. The initial state distribution $\pi = \{\pi_i\}$ where

$$\pi_i = Pr[q_1 = S_i],\ 1 \leq i \leq N \tag{2.21}$$

A finite state model can be used to generate a sequence $O = O_1\ O_2\ ...\ O_T$, where each observation O_t is one of the symbols from A, and B is the number of observations in the sequence. The generator can be constructed as follows [50]:

1. Choose an initial state $q_1 = S_i$ according to the initial state distribution π.
2. Set $t = 1$.
3. Choose $O_t = v_k$ according to the symbol probability distribution in state S_i, i.e., $b_i(k)$.
4. Transit to a new state $q_{t+1} = S_j$ according to the state transition probability distribution for state S_i, i.e., a_{ij}.
5. Set $t = t + 1$; return to step 3 if $t < T$; otherwise, terminate the procedure.

It can be seen from the above discussion that a complete description of a finite state model requires two model parameters (N and M), the specification of observation symbols, and the specification of the three probability measures A, B, and π. For convenience, a finite-state model can be referred to given the compact notation in Equation 2.22 [50].

$$\lambda = (A, B, \pi) \tag{2.22}$$

When building an HMM, one needs to account for many design parameters. For example, one needs to adjust $\lambda = (A, B, \pi)$ in order to maximize the probability of an observation sequence O given the model. The task of choosing these parameters can be interpreted as a question of "scoring how well a given model matches a given observation sequence. For example, if [one is] trying to choose among several competing models, the solution to [this question will determine] the model which best matches the observation" [50].

Another question that springs from the use of finite-state models is, given an observation sequence $O = O_1\ O_2\ ...\ O_T$; that is, given a sequence of events, what is an appropriate state transition sequence $Q = q_1\ q_2\ ...\ q_T$ that helps explain the observations. In all but degenerate models, for which certain events have zero probability of occurrence at certain states, a myriad of state transitions may have to be considered. Also, there is not a single 'correct' state sequence. Instead, one needs to define a criteria for optimality according to the context of the problem and to find the state transition that best meets the criteria.

Dynamic Markov Chain

Dynamic Markov Chains (DMC) provide a methodology for building state-based models. The technique starts from a simple initial model configuration. Then, this configuration's states get cloned as sequence events are observed. The newly incorporated states and transitions are created in order to adapt the model to the sequence.

Figure 2.3 illustrates the cloning operation. "A fragment of a finite-state model is shown in which state t is the target state for cloning. From it, emanate two transitions, one for symbol 0 and the other for 1, leading to states labeled x and y" [6]. Here, events are assumed to be 0s or 1s; however, the treatment can be extended to accommodate for multiple symbols. "There may well be several transitions into t. Three are illustrated, from states u, v, and w, and each will be labeled with 0 or 1 (although, in fact, no labels are shown)" [6] .

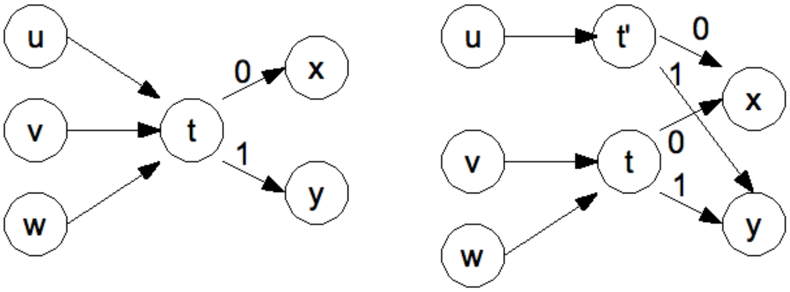

Figure 2.3: DMC state cloning.

Assume that the transition from state u has a large frequency count. Then state t is cloned to form an additional state t' because of the high frequency of the $u \rightarrow t$ transition. "The point of the exercise is that in the original model, whenever state t is entered, some contextual information is lost. Thereafter, it is not known whether t was entered from the $u \rightarrow t$ transition or along some other one, $v \rightarrow t$ or $w \rightarrow t$. But it is quite possible that the best prediction for the next symbol, 0 or 1, is influenced by the previous state. The simplest way to determine whether significant correlation exists is to clone state t as shown. The operation of cloning is analogous to increasing the context length for some particular prediction in a blended context model." Despite its interesting approach, Bell also shows that the DMC does not create a true finite-state model. That is, the model created by the DMC can be reduced to finite-context one. However, additional methods for the construction of HMMs have been proposed [56].

2.5 Cyber security and sequence modeling

When trying to model events, one may look at event frequency, event properties related to time (time interval between the arrival of two consecutive events or duration of individual events), or at the ordering or sequential properties of events.

Event frequency is probably the simplest of the approaches. In the context of cyber security, statistical tests, such as chi-square, have been used in conjunction with duration and time interval information when identifying anomalies indicative of security breaches. For example, [18] [46] [48] and [55] proposed to flag Denial of Service (DoS) attacks by applying statistical tests to packet fields.

Instead of focusing on event frequency or at time interval between events, this thesis leverages the ordering property of attack tracks; and this section investigates how the cyber security research community has applied models capable of capturing event ordering.

One application of sequence modeling to cyber security domain is related to program behavior. A program can be interpreted as a black box that emits observables in terms of system calls, which are used by applications to transfer CPU control to the operating system. Researchers have studied the use of system calls with the objective of detecting vulnerability exploitations such as buffer overflow attacks. The expectation is that a program that is being exploited will produce a sequence of system calls that differs from the ones generated during regular usage.

The University of New Mexico has reported several results in the area of system call modeling. For example, in 1996, Forrest, Hofmeyr, and Longstaff investigated the automatic construction of a database of short sequences of system calls generated when a program is executed under regular usage. They performed anomaly detection based on this database [20]. In 1997, Kosoresow and Hofmeyr used Deterministic Finite Automata to model program behavior [36]. In 1999, Warrender, Forrest, and Pearlmutter compared the performance of different models for regular behavior, one of the models being an HMM [64]. These researchers observed that sequences of system calls executed by a program are locally consistent during normal operation and that some unusual short sequences are generated when a security hole is exploited in the program.

In 1997, Lee and Stolfo combined the idea of automatic generation of program profiles with rule based specification of program behavior [41]. They used RIPPER, a rule learning program, in order to create classifiers for sequences of system calls. In order to illustrate the accuracy of their approach, they also created rules to predict the system call following a sequence of size n. Their prediction scheme was not intended to work in real-time. Much to the contrary, an entire training process was required to create prediction rules for a specific sequence size.

In 2001, Ye, et al. compared different frequency based approaches against a Markov chain finite-context model when performing anomaly detection on sequences of system calls [68]. Finally, other efforts to modeling program behavior can be found at [34] [35].

Finite-context models have also been used to model user shell commands. Ju and Vardi used a Markov chain model to profile UNIX command sequences with the objective of identifying user 'signature behavior' [29]. "To increase the model-flexibility, a 'higher-order' Markov structure is assumed, which takes into account the last few commands (rather than the last single command) in order to determine the next command. The underlying rational behind this approach is that, although the command sequence of any specific user is random, it generally follows a probabilistic patter which might be captured by a sufficiently rich Markov model." Probabilities for newly observed commands are derived based on the command history and can be used to determine if an attacker has hijacked an authentic user account.

Similar approaches have been used in order to find anomalies in system log messages.

Under regular conditions, a server is assumed to produce a certain log pattern that can be approximated as a finite-context model. When the system is under attack, the log message pattern is likely to change. Computing the probability of a sequence of log messages given the model generated under regular conditions will indicate whether the system is performing anomalously [69].

All the approaches described previously were implemented with anomaly detection in mind. Therefore, they assume a data-set composed of regular and abnormal behavior. This research is fundamentally different since it intends to characterize abnormal behavior. Therefore, it assumes a data-set composed solely of attack sequences captured by IDSs.

Perhaps Qin and Lee's approach to alert correlation is the work that bears closest resemblance with this thesis. In [49], Qin and Lee propose an alert correlation system that is also capable of performing attack prediction. The system performs low and high level correlation. The low level correlation is composed of two steps. The first is a Bayesian-based mechanism that correlates attack steps based on security states of computers and networks. The second approach to low level correlation employs statistical analysis in order to identify temporal and statistical patterns that may not have obvious or direct relationships in terms of security and performance measures. The results of low level correlation are "isolated alert sets." High level correlation is then applied on these sets in order to identify attack sequences across them and then to predict possible future attack steps. The high level correlation is based on plan recognition. Several possible attack plans need to be created by domain experts. Qin and Lee recognize that the system is unable to correlate isolated scenarios and to make an inference on the future attacks if an attacker's strategy does not fall within the defined attack plans created from expert knowledge. The design challenge is in creating sufficiently many attack plans. These plans need to be general enough in order to capture a myriad of attack behavior, but specific enough in order to provide insights about an attacker's goals. Different from Qin and Lee's approach, the model proposed in this thesis does not rely on expert knowledge.

2.6 Predictors and sequences

Sequence behavior analysis is intrinsically related to prediction. In order to accurately predict, one must have captured the intricacies in sequence behavior. Predictors infer future behavior based on past observations and have been used in many areas such as weather forecasting, investment, and defense.

In [33], Kieffer discusses the relationship between prediction and information theory. He details the difference between a deterministic and a universal predictor. When the data model for the sequence being analyzed is known, one may employ a deterministic predictor which infers the next event (X_i') as a function of the previous events.

$$X_i' = f_i(X_1, X_2, ...X_{i-1}) \tag{2.23}$$

By definition, an optimal predictor is a predictor that minimizes $P\{X_i' \neq X_i\}$ where X_i is the probability distribution for the known model. Thus, the optimal predictor chooses the most likely event for X_i given its known distribution:

$$X_i' \equiv max\ P\{X_i = a|X_1, X_2, ...X_{i-1}\} = f_i(X_1, X_2, ...X_{i-1}) \tag{2.24}$$

for $a \in \Omega$ where Ω is a finite alphabet. As noticeable from the definition, the optimal predictor for a known data model is a deterministic predictor.

Suppose that the model M of the data sequence $\{X_i\}$ is unknown but lies in some known class μ of stationary models. A predictor infers X_i to be X_i' with an average prediction error of:

$$n^{-1} \sum_{i=1}^{n} P\{X_i' \neq X_i\} \tag{2.25}$$

A universal predictor is an optimal predictor whose average prediction error converges to the minimum prediction error, $\Delta(M)$, independently of the specific model $M \in \mu$:

$$n^{-1} \sum_{i=1}^{n} P\{X_i' \neq X_i\} \rightarrow \Delta(M) \tag{2.26}$$

as $n \rightarrow \infty$. In other words, when the number of observed outcomes goes to infinity, a universal predictor (which does not require any a priori information about a given data) performs as well as the best deterministic predictor for the given data model. Since all universal predictors will converge to the minimal prediction error, their performance is evaluated based on their speed of convergence given in big-O notation [33].

One must also take into consideration the size of the model. For example, effort has been placed on determining the minimum amount of data needed to capture source probabilities. Gathering more and more representative sequences may yield to better prediction; however, in practice there is an obvious difficulty in managing the exponentially growing number of items that need to be stored [51]. In [56], Cosma and Kristina Shalizi discuss the concept of *minimum sufficient statistics*; that is, "the most compact summary of the data which retains all predictively-relevant information."

As discussed previously, two main approaches are used for modeling and prediction. One is based on Variable Length Markov Models (VLMM) which is also a type of finite-context models, and another is based on Hidden Markov Models (HMM), also known as finite-state models. Since the introduction of the concept of *context* by Rissanen in [51], many researchers have explored VLMMs [26] [8] [32] [54] [60] [66] [5]. HMMs have also received much attention [27] [56] [57] [58] [61].

The objective of this work is to determine how an attacker's previous actions impact his or her future behavior. One of the goals is to determine how strong is the correlation between an action and its predecessors. For this reason, the work is based on finite-context models, more specifically on a Variable Length Markov Model (VLMM).

One approach devised in 1992 by Ehrenfeucht and Mycielski became the base for other finite-context methods [16]. Ehrenfeucht and Mycielski defined the concept of maximal suffix (the largest repeating substring) as the smallest l such that $x_l, x_{l+1}, ...x_n = x_{l-i}, x_{l-i+1}, ...x_{n-i}$ for some $1 \leq i \leq n$. By definition, the maximal suffix has size $D \equiv n - l + 1$. Then they created a pattern matching predictor that took the smallest i (the most recent occurrence), say $i = I$, for which the longest match is found and set the prediction for x_{n+1} to be x_{n-I+1}. In other words, the next occurrence is estimated to repeat the character that followed the maximum suffix.

Jacquet, Szpankowski, and Apostol modified Ehrenfeucht's and Mycielski's predictor by searching for a fraction of the maximal suffix in the original sequence [26]. The fraction of the maximal suffix was set to a size $\lceil \alpha D_n \rceil$ such that the fractional suffix occurs more than twice in the original string. Each of such occurrences was called a marker. A majority vote among the K symbols that immediately follow the markers was used to predict the next K sequence symbols. They proved that this scheme yields to a universal predictor.

Begleiter, El-Yaniv, and Yona compare the performance of different Variable Length Markov Models, including the algorithm similar to the one proposed in [26], when performing predictions on sequences composed of English texts, proteins, and music pieces [5].

The context predictor implemented in this thesis is based on the work of Jacquet, Szpankowski, and Apostol and on the algorithm provided by Begleiter, El-Yaniv, and Yona. Similar to their approach, this thesis utilizes a suffix tree as data structure to hold all contexts of a sequence. As discussed in Chapter 3, the proposed Variable Length Markov Model (VLMM) implementation looks for the longest match of the observed sequence into the model and combines information from all matches that are shorter than the longest.

Chapter 3

Framework for cyber attack track characterization

This chapter discusses a framework for cyber attack characterization. Section 3.1 gives an overview of the inputs to the framework. Sections 3.2 discusses the implementation of the Variable Length Markov Model (VLMM). Sections 3.2.1, 3.2.2 and 3.2.3 discuss how the framework can be used to predict future attack actions, propose a metrics for variability in attack actions based on information entropy, and propose a method for classifying attack tracks as sophisticated or simple based on average log-loss. Section 3.3 talks about implementation choices such as programming languages, paradigms and tools.

3.1 Input data-set

This thesis objective is to investigate how information contained in attack tracks can be used to leverage the current understanding of cyber attack behavior. The first step was to find a suitable data-set containing correlated alert messages. Since most of the research in cyber security to this point concentrates on the development of IDSs and of correlation engines, most of the data-sets in existence today are composed of raw alerts of normal and malicious activities with no ground truth in terms of attack tracks.

The desired data-set for this thesis work is composed of ordered collections of alert messages. As discussed in Chapter 2, this excludes the possibility of using common sets such as the 1998, 1999 and 2000 MIT Lincoln Laboratory [31] [43], the KDD Cup 99 [13] and the Capture the Flag [10] sets.

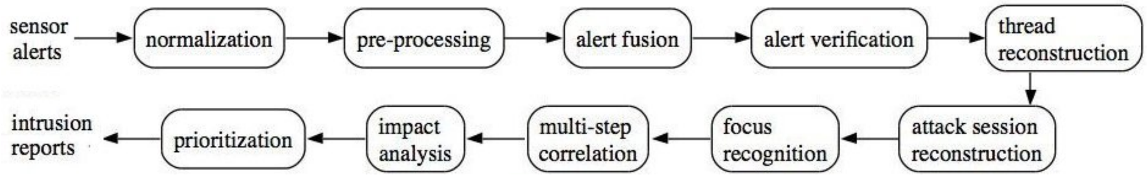

Figure 3.1: Steps into correlation

Yet another issue is that correlation engines vary greatly in implementation; therefore, not every correlated set is suitable to this thesis research. Valeur et al. acknowledge that numerous correlation engines have been proposed and that there is no consensus on what the process should entail exactly. They suggest the approach depicted in Figure 3.1 and detailed next:

- **Normalization and preprocessing**: alerts can be produced at many levels — network packets, operating system calls, log messages from applications — normalization and preprocessing create a common set of fields across different alert levels.

- **Fusion**: combines duplicated alerts generated by the same attack action.

- **Verification**: given an alert and a model of the network, it determines whether the attack succeeded or failed.

- **Thread reconstruction**: at a low level, this step combines several alerts belonging to a single attacker's actions. Additional processing is performed by multistep reconstruction.

- **Attack session reconstruction**: at this stage, network- and host-based alerts are combined.

- **Focus recognition**: this step identifies Denial of Service (DoS) attacks and port scannings by determining if a host is either the source or the target in many alert messages.

- **Multistep correlation**: this step performs high level correlation. For example, it lumps scanning, intrusion and privilege escalation into a single attack track.
- **Impact analysis and Prioritization**: computes the potential harm of correlated attacks and prioritizes them accordingly.

This work expects the output of an engine that can perform normalization, preprocessing, fusion, thread reconstruction, attack session reconstruction, focus recognition and multistep correlation. Verification, impact analysis and prioritization are not being used in this work. Therefore, the proposed cyber attack characterization framework detailed in this thesis expects a data-set with the following characteristics:

1. Data is represented in the form of attack tracks.
2. An attack track is a sequence of alerts corresponding to a single multistep attack.
3. Tracks are sequences, therefore there is no forking into parallel subtracks or into any other topology.
4. Alerts belonging to a track are presented in the same order as the actions that have triggered them.
5. Alerts from different sensors are consistent among each other. In other words, normalization and pre-processing steps are performed during correlation.
6. Fusion is also part of alert correlation so that duplicate alerts generated from a single action are lumped together.
7. Unsuccessful attack attempts don't necessarily need to be marked, and they should not be removed because they also carry information about attack behavior.
8. There are sufficiently many alerts from a statistical point of view.
9. Alerts are generated from representative attack scenarios.

There are many possible ways of perceiving a single attack action [28]. This work considers IDS alerts as direct representations of an attack step. Given the existence of a suitable data-set, IDS alert fields that are relevant to modeling attack behavior are selected. Fields containing descriptions of the attack and the IP of the target machine are of major relevance. After this selection, attack tracks are mapped to sequences of characters with each character representing an alert. The set of unique characters that can happen in a sequence is referred to as the *alphabet*, and is depicted as the capital Greek letter omega (Ω).

The alphabet size is dependent on the number of alert fields considered and the variety of possible values that these fields can take. For example, the cyber community usually classifies alerts given the following categories: scanning (reconnaissance); footprinting (reconnaissance); intrusion root; intrusion user; backdoor, trojan, virus, worm or similar attacks; DoS; data exfiltration. A possible alphabet based to represent these categories is shown in Table 3.1. A hypothetical multi-stage attack composed of scanning actions, followed by an intrusion with user privileges, followed by data exfiltration would be represented by the sequence 'ADG.' Therefore, the byproduct of translating attack tracks into sequences is a set S consisting of several attack sequences $s \in S$ where $s = \{x_1, x_2, ...x_n\}$ and $x_i \in \Omega \; \forall \; i$.

Char	Category
A	Scanning (reconnaissance)
B	Footprinting (reconnaissance)
C	Intrusion Root
D	Intrusion User
E	Backdoor, Trojan, Virus or Worm attack
F	Denial of Service
G	Data Exfiltration

Table 3.1: Alert to character mapping example

If more than one field is to be considered for the creation of the alphabet, then characters are made to represent combinations of the values between the desired fields. Also, since finite alphabets are assumed, continuous fields must be discretized before they can be used.

3.2 VLMM suffix tree based implementation

The framework was implemented as a Markov Model given the intuition that an attacker's actions are strongly correlated with his or hers recent behavior. An o^{th} order Markov Model captures this notion very succinctly through the following definition:

$$P_o\{X_{n+1}|x_{n-o+1}, ..., x_n\},$$

where the o previous observations from x_{n-o+1} to x_n are also called *contexts* — therefore, this scheme is also referred to as *context-based* modeling.

As discussed in Chapter 2.3, a first order Markov Model can be represented by a square transition probability matrix in which entry $p_{i,j}$ holds the probability of a transition from state i to j. A second order Markov Model can be represented with a rectangular matrix in which $p_{ij,k}$ holds the probability of a transition into state k given that states i and j have been observed. Similarly, higher order Markov Models can be obtained.

This thesis implements a Variable Length Markov Model (VLMM) in which several order Markov Models are blended. In other words, the probability of an event being observed is derived by blending probabilities from n, $n - 1$, $n - 2$, including probabilities from a first up to zero and minus-one order Markov Models.

Instead of using several matrices for each model order, the VLMM is implemented with a suffix tree. The suffix tree is a very good choice of data structure since a context of size n and its succeeding symbol x_{n+1} can be found in $O(n)$ time. In addition, the suffix tree naturally holds all sub-contexts (suffixes) of a sequence; therefore, it holds models of order n, $n - 1$... up to a zero and minus one order.

For example, Figure 3.2 shows the suffix tree for the sequence:

$$+FGGFGF* \tag{3.1}$$

where '+' and '*' mark the start and the end of sequence and 'F' and 'G' correspond to Snort alerts 'WEB-IIS nsiislog.dll access' and 'WEB-MISC Invalid HTTP Version String,' respectively. All suffixes of Sequence 4.1 — $F*$, GF*, $FGF*$, $GFGF*$, $GGFGF*$, $FGGFGF*$ and $+FGGFGF*$ — and their frequencies are present in the tree. These suffixes are the contexts in which Markov Models are built. For example, according to the data-set, the context 'F' was followed by a 'G' two times and the context 'FG' was followed by '$GFGF*$' once.

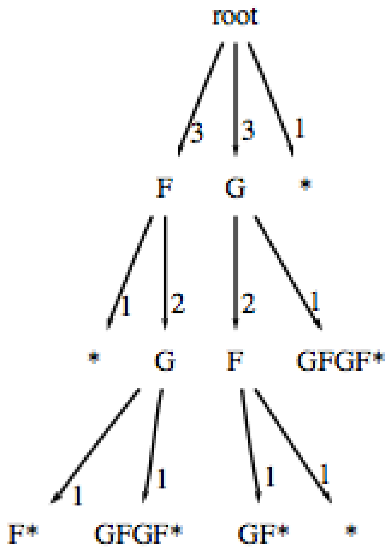

Figure 3.2: Suffix tree for the finite sequence '$+FGGFGF*$'.

Once built from a data-set containing representative attack tracks, the tree can be mined for information. [1] As shown next, Section 3.2.1 describes how the model can be used to predict future alerts given an unfolding sequence of attack actions. Section 3.2.2 proposes a method for measuring the uncertainty of a prediction using information entropy, which also represents the variability of possible future attack actions. Section 3.2.3 shows how

[1] The research objective is not to develop a system that can be placed in production; therefore, the model size is not a concern. Instead of creating methods for limiting the model's size, the tree is allowed to grow unbounded given training set data.

average log-loss can be used to classify attack tracks as sophisticated or simple.

3.2.1 Prediction

Let $s = \{x_1, x_2, ..., x_n\}$ be an unfolding sequence of attack actions. The objective is to predict the next action (x_{n+1}) given s and given a data-set containing representative attack tracks.

The approach taken is the following:

- Let H be a set containing all suffixes of representative attack tracks. In the case of this research, H is contained in a suffix tree.
- Let l be the size of the longest suffix of s such that s_l is in H.
- Let H' be the set of all suffixes of s_l. Then $H' \subset H$.
- For all $s_i \in H'$, $i \leq l$, compute the following discrete probability distribution: $P_i\{X_{n+1}|x_{n-i+1}, ..., x_n\}$.
- A Variable Length Markov Model (VLMM) blends P_i for all $i \leq l$ creating a $P\{X_{n+1}\}$.
- A prediction is set to $x_{n+1} = \arg\max_{\sigma \in \Omega} P\{X_{n+1} = \sigma\}$.

For example, consider using the suffix tree shown in Figure 3.2 to predict the next alert following Sequence 3.2: [2]

$$+FFGF \tag{3.2}$$

The longest suffix of Sequence 3.2 in the suffix tree is 'FGF' and it is followed once by '$*$.' Therefore, the third order prediction is given by: $P_3(*|FGF) = 1$. The second

[2] This example illustrates the derivation of probabilities from the tree branches; however, this is not a realistic case since the tree in Figure 3.2 is built from only one attack track given by Sequence 4.1. In reality, a useful tree would have been built from several representative attack tracks.

longest suffix is 'GF' and it is followed by 'G' and '$*$'. Therefore, the second order prediction is given by: $P_2(G|GF) = 1/2$ and $P_2(*|GF) = 1/2$. First order predictions are $P_1(G|F) = 2/3$ and $P_1(*|F) = 1/3$. The zero and minus one order predictions are $P_0(F) = P_0(G) = 3/6$ and $P_{-1}(F) = P_{-1}(F) = 1/2$. Probabilities for a 0^{th} order model are the normalized counts of all possible characters in the alphabet. The minus one order is defined as $P_{-1}(x) = 1/|\Omega|$ for all $x \in \Omega$, where $|\Omega|$ is the alphabet size.

$P\{X_{n+1}\}$ is created by blending these several finite-context predictions. Let $P_n(x)$ be the probability of a character $x \in \Omega$ happening according to the n^{th} order model. The blended probability is the weighted sum of $P_n(x)$:

$$P(x) = P\{X_{n+1} = x\} = \sum_{n=-1}^{l} w_n \times P_n(x)$$

where l is the size for the longest suffix of s that can be found in the suffix tree.

As discussed in Section 2.3, weights should be assigned proportionally to context frequencies. *Escape probabilities* are used to compute weights so that the relevance of a context is proportional to its frequency. Escape probabilities capture the notion that a longer context may be too restrictive in its prediction, and that lower orders should also be taken into account. This is illustrated in Figure 3.3.

In the example of Sequence 3.2, the 4^{th}, 3^{rd}, 2^{nd}, 1^{st}, 0 and -1 order contexts have escape probabilities 1, $1/2$, $1/3$, $1/4$ and $1/8$ and 0 respectively as represented by the vertical arrows of Figure 3.3. Equation 2.15 was employed when computing these escape probabilities. In this example, the weights for the 4^{th}, 3^{rd}, 2^{nd}, 1^{st}, 0 and -1 order contexts are 0, $1/2$, $1/3$, $1/8$, $7/192$ and $1/192$ respectively. They are computed using Equation 2.14.

Once the weights have been derived, blending takes place as shown in Table 3.2. The probability for 'F' is derived by summing the multiplication of the probabilities of 'F' at each context order by the context's escape probability. In this case, $P(F) = 3/7 \times 7/192 + 1/3 \times 1/192$. The probability for '$G$' is computed as $P(G) = 1/2 \times 1/3 + 2/3 \times 1/8 + 3/7 \times 7/192 + 1/3 \times 1/192$. The same procedure is applied for '$*$'.

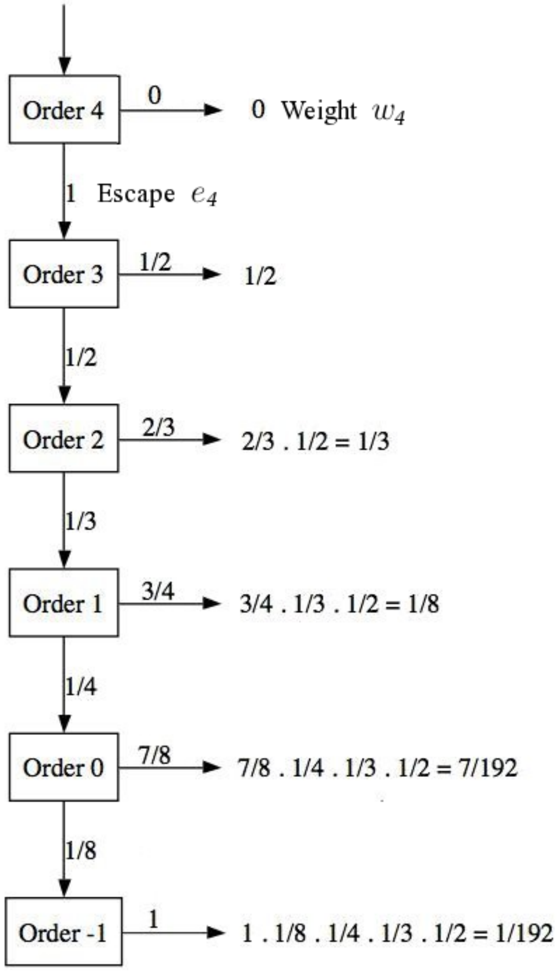

Figure 3.3: Calculation of escape probabilities – method A, sequence $FFGF$

This discussion is captured by the pseudo-code in Figure 3.4 which computes the probability that a character c will follow a given context. An auxiliary method described in Figure 3.5 is used to compute the escape probabilities that yield to weights w_n.

The performance of the predictor is measured by a *prediction rate* (%). The prediction rate is calculated as the percentage of correct predictions over the total number of predictions made using the suffix tree model. As it will be described in Chapter 4, the data-set randomly split in half, with one half as training set and the other half as testing set. Once the suffix tree was created given the training set, prediction rates for sequences in the testing

			Predictions							
order	context	counts	F		G		*		w_o	e_o
4	$FFGF$	0	-	0	-	0	-	0	0	-
3	FGF	1	0	0	0	0	1	1	1/2	1/2
2	GF	2	0	0	1/2	1	1/2	1	1/3	1/3
1	F	3	0	0	2/3	2	1/3	1	1/8	1/4
0	-	7	3/7	3	3/7	3	1/7	1	7/192	1/8
-1	-	-	1/3	-	1/3	-	1/3	-	1/192	0
Blended probabilities			5/288		77/288		206/288			

Table 3.2: Calculation of blended probabilities – escape method A, sequence $FFGF$

set were computed given different choices of alphabets.

3.2.2 Entropy

Given a VLMM built from representative attack data, the following discrete probability distribution functions (DPDFs) can be computed:

$$\begin{aligned}
&P\{X_1 && \} \\
&P\{X_2 && \mid \quad X_1 = x_1\} \\
&P\{X_3 && \mid \quad X_1 = x_1, X_2 = x_2\} \\
&\quad\ldots \\
&P_n\{X_n && \mid \quad X_1 = x_1, X_2 = x_2, ..., X_{n-1} = x_{n-1}\}
\end{aligned} \tag{3.3}$$

The DPDFs are then used to compute the entropy of each symbol in a sequence $s = \{x_1, x_2, ..., x_n\}$. Let H_i be the entropy of the i^{th} symbol:

$$\begin{aligned}
H_i = -\sum_{X_i \in \Omega} & P\{X_i \mid X_1 = x_1, X_2 = x_2, ..., X_{i-1} = x_{i-1}\} \\
\times\ & \log_2 P\{X_i \mid X_1 = x_1, X_2 = x_2, ..., X_{i-1} = x_{i-1}\}
\end{aligned} \tag{3.4}$$

```
string $context$
character $c$
accumulated_escape = 1

for i from 1 to $context$.length do
   sub_context = $context$[ i, $context$.length ]
   append_context = sub_context & $c$
   nominator = number of matches for append_context in suffix tree
   denominator = number of matches for sub_context in suffix tree
   probability = nominator / denominator
   escape_prob = compute_escape
   prediction = prediction + probability $\times$ (1 - escape_prob ) $\times$ accumulated_escape
   accumulated_escape = accumulated_escape $\times$ escape_prob
end for


escape_prob = compute_escape
nominator = number of matches for $c$ in suffix tree
denominator = total number of observed characters
probability = nominator / denominator
prediction = prediction + probability $\times$ (1 - escape_prob ) $\times$ accumulated_escape
accumulated_escape = accumulated_escape $\times$ escape_prob

probability = 1 / alphabet_size
prediction = prediction + probability $\times$ accumulated_escape

return prediction
```

Figure 3.4: Prediction pseudocode

The larger the value for H_i, the larger the uncertainty associated with X_i. In other words, H_i tells us the predictability of the i^{th} symbol in the sequence.

As it will be shown in Chapter 4, the approach was to measure H_i for all symbols in the testing data and then to make a prediction for that symbol given the model. It is expected that mispredicted symbols will, on average, have a higher uncertainty (H_i) than correctly predicted ones.

```
string $context$

numerator = number of characters following $context$ in the suffix tree
denominator = number of matches for $context$ in the suffix tree
if ( escape_method == A ) then
  escape_prob = 1 / ( 1 + denominator )
else if ( escape_method == B ) then
  escape_prob = numerator / ( denominator )
else
  escape_prob = numerator / ( numerator + denominator )
end if

return escape_prob
```

Figure 3.5: Escape probabilities pseudocode

3.2.3 Average log-loss

Average log-loss is a metric that can be used to compare the performance of models when predicting a given sequence. The greater the probability assigned by a predictor to a sequence of symbols, the smaller its average log-loss.

For example, let x_j be a character in a finite alphabet Ω. Let P be a predictor that assigns a probability $P(x_j)$. Then, given a testing sequence $s = \{x_1, x_2, ...x_n\}$ $x_i \in \Omega \; \forall \; i$, the average log-loss is defined as:

$$l(P, s) = -\frac{1}{n} \sum_{i=1}^{n} \log P(x_i | x_1, ..., x_{i-1}) \tag{3.5}$$

Suppose a predictor is able to tell the future. In other words, it assigns 100% probability to the characters that will happen next, and zero to all other characters:

$$P(x_j | x_1, x_2, ..., x_{i-1}) = \begin{cases} 1 & \text{if } x_j = x_i \\ 0 & \text{otherwise} \end{cases}$$

From the definition its noticeable that the average log-loss for a perfect predictor is zero: $l(P, s) = 0$.

Assume now two predictors $\hat{P}$ and $\dot{P}$. By computing $l(\hat{P}, s)$ and $l(\dot{P}, s)$, its possible to compare how well the sequence s is captured by both predictors. Two predictors may correctly predict a sequence; however, the one that assigns greater probabilities to the occurring characters will have a lower average log-loss.

Average log-loss also measures how well a sequence is captured by a predictor, or how unusual the sequence is to a given predictor. This thesis employs average log-loss when classifying attack sequences as simple and sophisticated. As it will be shown in Chapter 4, the average log-loss of each sequence in the test set is computed. Then the set is split according to a threshold value for the average-log loss. Sequences whose average log-loss is above the threshold are placed in the sophisticated set, and placed in the simple set otherwise. When comparing the prediction rate of each set, its expected that the higher log-loss sequences (the sophisticated ones) will have lower prediction rates.

3.3 Choices of programming languages and tools

The framework was implemented in Java and Python. The input to the framework is an XML file containing attack tracks. The processing chain starts with the conversion of tracks from XML into sequences of characters. An alphabet file with the mapping between characters and their related alert fields is also created. This is shown in the top box in Figure 3.6. In addition, a filtering process which will be described in Chapter 4, is also performed at this level.

Python was chosen to implement the preprocessing and filtering algorithms because of its regular expression support and because of its implementation of Simple API for XML (SAX), which provides an event-driven API for reading data from large XML documents.

The most important classes here are:

- Alert.py
- Track.py

- ParseLog.py
- Conversion.py

‘ParseLog.py’ parses the XML creating Track and Alert objects. ‘Conversion.py’ transforms tracks into text files containing sequences that are fed into the VLMM.

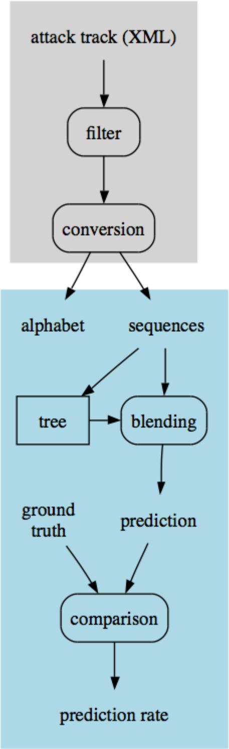

Figure 3.6: Framework processing chain

The lower box in Figure 3.6 shows the tree being constructed from sequences. It also shows the blending process from which predictions for an unfolding attack are made. The prediction is contrasted against ground truth and a prediction rate is computed. It is also possible to train the suffix tree with attack sequences as they are observed. In addition, a graphical representation of the suffix tree can be outputted in DOT format, which is a plain text graph description language that can be rendered using Graph Visualization Software (Graphviz).

The suffix tree and the prediction algorithm were implemented in Java. The most important classes are:

- suffix.java
- vmm.algs.SuffixTreePredictor.java
- vmm.algs.st.SuffixTree.java
- vmm.algs.st.SuffixTreeNode.java
- vmm.pred

The class 'suffix.java' is the main driver. It takes command line arguments and creates instances from other classes. 'SuffixTreePredictor.java' contains the predictor, which works by extracting information from 'SuffixTree.java' and its nodes, 'SuffixTreeNode.java'. The package 'vmm.pred' contains interfaces and iterators. In addition, the following classes are used to test the suffix tree and to derive results:

- TestBench.java
- TestEvent.java
- TestEventListener.java
- EntropyListener.java
- LogLossListener.java

'TestBench.java' computes the prediction rates of sequences in the testing-set based on the suffix tree constructed with sequences from the training-set. An observer/observable paradigm in which listeners register themselves with a producer of test events ('TestEvent.java') was implemented in order to simplify the collection of results.

‘TestBench’ is the producer that calls its observers when an alert is predicted. Each observer is responsible for keeping metrics that are relevant to itself. Observers (‘EntropyListener.java’ and ‘LogLossListener.java’) implement an interface described by ‘TestEventListener.java.’

The suffix tree algorithm was motivated by [5]. The program also borrows from other open source pieces of code such as Trove’s implementation of *getopt* for command line parsing, and JRuby’s implementation of *glob* for directory and file interfacing. The complete code is provided in the attached CD-ROM.

Chapter 4

Results

4.1 Experimental setup

This section presents Skaion-2, which is the data-set used for this thesis work. Skaion-2 is more adequate to the research of attack tracks and attack behavior than the commonly used data-sets described in Section 2.1. However, Skaion-2 has a few shortcomings and it doesn't always align with the guidelines from Chapter 3.1. These inadequacies in the data and the workarounds developed are presented here.

4.1.1 Skaion-2 data-set

The Skaion-2 data-set was chosen as the input to this thesis framework. The set was crafted by Skaion Corporation through scripted multi-stage attacks performed on a VMware network. Different from the data-sets described in Section 2.1, Skaion-2 contains alerts related to malicious activities only. In other words, background noise traffic has been removed. In addition, its alerts are grouped into attack tracks as opposed to being uncorrelated.

The data-set is presented in XML format. Each attack track is enclosed around <Track> tags which are, in turn, composed of many alerts belonging to a single multi-state attack.

```
<?xml version="1.0" encoding="UTF-8"?>
<Scenario>
```

```
  <ID>Test</ID>
  <Description />
  <Results>
    <Track>
      <ID>1149003966567</ID>
      <Description>
        Multistage_with_Background_Recon_Cyber_Attack_Model
      </Description>
      <Certainty>0.54545456</Certainty>
      <Alert>
        ...
      </Alert>
    </Track>
    <Track>
      ...
    </Track>
    ...
  </Results>
</Scenario>
```

Tracks contain alerts from sensors such as Snort [53] and Dragon [3], and from IIS [12] and Apache [19] Web servers. However, alert fields from different sources are not necessarily consistent among each other, which indicates that Skaion-2 could have benefited from a better normalization process. In order to circumvent this shortcoming, this research focuses solely on Snort alerts because it has the richest set of alert fields and because it accounts for the majority of the data-set.

Below is an example of Snort alert as presented in the data-set:

```
<Alert>
  <Snort>
    <DateTime>2005-10-04 14:34:47.503911</DateTime>
    <Description>ICMP PING NMAP</Description>
    <Protocol>ICMP</Protocol>
    <Source_IP>192.168.222.4</Source_IP>
    <Dest_IP>100.20.0.0</Dest_IP>
    <GId>1</GId>
    <SId>469</SId>
    <SId_rev>3</SId_rev>
    <Classification>Attempted Information Leak</Classification>
    <Priority>2</Priority>
  </Snort>
  <Sensor_ID>snort232</Sensor_ID>
  <Category>Recon_Scanning</Category>
</Alert>
```

The first part of this thesis work was to determine how different alert fields relate to behavior. For example Snort alerts contain description and category fields. We believe that this information reflects the hacker's tendency in choosing different reconnaissance and exploitation methods and that it allows mapping to system and network vulnerabilities.

Skaion-2 contains the following Snort alert descriptions. These alerts belong to nine categories of attacks that were devised by the creators of the data-set.

- **Category**: Intrusion_Other

  Descriptions:

  - (http_inspect) BARE BYTE UNICODE ENCODING
  - (http_inspect) IIS UNICODE CODEPOINT ENCODING
  - (http_inspect) OVERSIZE REQUEST-URI DIRECTORY
  - (http_inspect) U ENCODING
  - (http_inspect) OVERSIZE CHUNK ENCODING

- **Category**: Recon_Scanning

  Descriptions:

  - SCAN SOCKS Proxy attempt

  - ICMP PING NMAP
  - ICMP L3retriever Ping

- **Category**: Recon_Footprinting

  Descriptions:

  - SCAN nmap TCP
  - SCAN nmap XMAS

- **Category**: Intrusion_Root

  Descriptions:

  - EXPLOIT ICQ SRV_MULTI/SRV_META_USER overflow attempt
  - EXPLOIT ICQ SRV_MULTI/SRV_META_USER first name overflow attempt
  - FTP format string attempt
  - FTP wu-ftp bad file completion attempt
  - FTP USER overflow attempt
  - NETBIOS SMB_DS OpenKey unicode overflow attempt
  - NETBIOS SMB IPC$ unicode share access
  - NETBIOS SMB-DS Session Setup AndX request unicode username overflow attempt
  - NETBIOS SMB-DS IPC$ unicode share access
  - NETBIOS SMB Session Setup NTMLSSP unicode asn1 overflow attempt
  - NETBIOS SMB OpenKey unicode overflow attempt
  - NETBIOS SMB-DS Session Setup NTMLSSP unicode asn1 overflow attempt
  - SCAN Squid Proxy attempt
  - SCAN Proxy Port 8080 attempt

  - SMTP MAIL FROM overflow attempt
  - SMTP Content-Transfer-Encoding overflow attempt
  - WEB-IIS nsiislog.dll access
  - WEB-IIS w3who.dll buffer overflow attempt
  - WEB-MISC Chunked-Encoding transfer attempt
  - WEB-MISC PCT Client_Hello overflow attempt
  - WEB-MISC Invalid HTTP Version String
  - WEB-MISC cat%20 access

- **Category**: Intrusion_User

  Descriptions:

  - ICMP Large ICMP Packet
  - WEB-IIS ISAPI .printer access
  - WEB-IIS ISAPI .idq access
  - WEB-IIS ISAPI .idq attempt
  - WEB-IIS encoding access
  - WEB-MISC /etc/passwd
  - WEB-MISC nc.exe attempt

- **Category**: Goal_Backdoor_Trojan_Virus_Worm

  Descriptions:

  - MISC OpenSSL Worm traffic
  - WEB-MISC bad HTTP/1.1 request, Potentially worm attack

- **Category**:Goal_Data Exfiltration

  Descriptions:

  - ATTACK-RESPONSES directory listing
  - FTP format string attempt
  - (portscan) TCP Portscan
  - (portscan) TCP Portsweep

- **Category**:Goal_Dos

  Descriptions:

  - WEB-IIS .htr access
  - WEB-MISC apache directory disclosure attempt

- **Misc_Other**

  Descriptions:

  - WEB-MISC http directory traversal

The nine category fields provide information at a coarser level of granularity than the 47 descriptions. For example, 'ICMP L3retriever Ping' is much more specific than 'Recon_Footprinting.' However, some alerts happen too infrequently, especially at the description level, and the data-set doesn't contain statistically significant information about them. Therefore, the trade-off in using category versus description fields is investigated in terms of level of granularity and of information reliability.

Destination IP fields contain the IP address of the attack target machine. This information is relevant to the analysis of attack behavior; therefore, it is incorporated into the characterization framework. Unfortunately, the observations regarding attack target machines are limited since the network topology used when creating the Skaion-2 data-set is not publicly available.

In summary, three choices of alphabets Ω, relevant during the creation of sequences from attack tracks, are considered:

- Snort alert description fields, which contain a detailed account of the attack action.

- Category fields, which contain a high level account of the attack.
- Category and Destination IP fields combined.

Conversion of attack tracks into sequences

As shown previously, three choices of alphabets are considered: Snort alert description fields, which contain a detailed account of the attack action; category fields, which contain a high level account of the attack; and category and Destination IP fields combined, from which insights about the path of an attacker into the network is derived from the choices of target machines.

In the case of description, alert messages belonging to an attack track are converted into a sequence of characters with each character mapping to a Snort alert description. Similarly, the category field may also be used to capture information at a coarser level of granularity.

In this case of combining the target machine's IP with its alert category, two alerts belonging to the same track are mapped to characters depending on whether the destination IP of the most recent alert is different from the destination IP of its predecessor. For example, consider the following attack track:

```
<Track>
  <Alert>
  <Category>Recon_Scanning</Category>
  <Snort>
    <Dest_IP>100.20.0.0</Dest_IP>
  </Snort>
  </Alert>
  <Alert>
  <Category>Recon_Scanning</Category>
  <Snort>
    <Dest_IP>100.20.0.1</Dest_IP>
  </Snort>
  </Alert>
    <Alert>
  <Category>Intrusion_User</Category>
  <Snort>
    <Dest_IP>100.20.0.1</Dest_IP>
  </Snort>
  </Alert>
</Track>
```

This track is translated to AaG where A and a correspond to the category Recon_Scanning, with the lowercase letter representing a change in IP from the previous alert. The G represents Intrusion_User, and it is upper case because its destination IP is the same as the previous alert's. The lower/upper case convention to mark different/same target IP is kept throughout this thesis work.

Filtering the data-set

Skaion-2 contains $1,331$ sequences and a total of $7,117$ Snort alerts. A preliminary study of the data reveals that attack tracks are composed of several alerts corresponding to a single attack action. In other words, duplicate alerts, which result from one attack action triggering multiple IDSs or being reported several times by the same IDS, are not filtered. Duplicate alerts are handled by a pre-filtering process that removes alerts that have the same fields (signature, source IP, target IP, etc.) as the preceding ones if they fall within $\Delta t = 1$ second from each other — the Δ value was chosen arbitrarily. If the resulting attack track has size one, the track is also removed from the data-set. In [63], Valeur et al. propose a similar approach to alert fusion; however, they use a two second window and they process alerts in realtime as they arrive into their proposed correlation engine.

Once the Skaion-2 data-set is filtered, the set is left with $1,113$ sequences and $4,723$ alerts. At this stage, the data may still contain repeated consecutive alerts if they happen at a time interval greater than 1 second. This data is referred to as the *repetition* set. A second filtering process that removes consecutive alerts of the same type, regardless of the time interval between them, is also investigated. This step yields to what is called the *no-repetition* data-set. The objective of creating a no-repetition set is to investigate the transition in attack actions. That is, investigating how different actions trigger different types of alerts rather than looking at how many times a certain alert happens consecutively.

Once repetition has been removed, the total number of attack tracks decreases to a little over 100 and the total number of alerts drops to about $1,200$ depending on the choice of alphabet. This significant reduction happens because many attack tracks are composed of a single reconnaissance action. Since there is no variety within these tracks, their sequences are reduced to a single alert that gets removed from the data-set. Therefore, another shortcoming in Skaion-2 is that, despite its initial large size, it does not seem to be composed of many representative attack tracks. However, this data-set is the only one composed of correlated attack tracks that was available at the time of this work.

4.1.2 Reporting of results

The attack tracks in Skaion-2 are randomly split in half, with one half being used for training, and the other half for testing. The suffix tree, which is the basis for the VLMM, is constructed from the training set as shown in Section 3.2. On the other hand, the testing set was used to derive three main metrics: *prediction rate* and *entropy* of sequence characters, and *average log-loss* of sequences.

Section 3.2.1 discusses how the VLMM can be used to predict future attack actions based on an unfolding attack sequence. The *prediction rate* (%) is a performance measurement calculated as the percentage of correctly predicted attacks over the total number of predictions made using the VLMM. This metric indicates how adequately the model captures the probabilities of certain attack actions given an attacker's recent history. Ten independent runs with randomly selected training and testing sets were averaged for the results presented in Section 4.2. These results are prediction rates given with respect to the top-1, top-2 and top-3 predicted alerts. The term *top-1* refers to the attack that is assigned the highest probability of occurrence given the VLMM and given the attacker's recent history. The term *top-2* and *top-3* refer to the first and second; and to the first, second and third attack that are predicted with highest probability.

Results are also given in terms of entropy. As explained in Section 3.2.2, variability in attack actions can be measured from the entropy of a sequence symbol given its context. Therefore, a correlation between entropy and prediction rate is reported in Section 4.3.

Average log-loss is a measurement of how unusual a sequence is with respect to a given model. Section 4.4 shows how the average log-loss of test sequences is computed and is used to classify attacks as simple and sophisticated.

4.2 Prediction

4.2.1 VLMM versus 0^{th}, 1^{st}, 2^{nd}, and 3^{rd} order predictors

The relationship between an attack action and the attacker's recent behavior can be inferred from the correlation of neighboring attacks within a sequence. Attack actions were predicted based on 0^{th}, 1^{st}, 2^{nd} and 3^{rd} order finite-context models. The performance of each model for the no-repetition data-set [1] is measured in terms of prediction rate and is presented in Figure 4.1.

In this figure, points marked with a dash ('-') represent the prediction rate when considering the symbol that is attributed the highest probability of occurrence (also referred to as top-1). Points marked with 'x' represent the top-2 prediction rate, while points marked with '*' represent top-3 prediction rate. For example, when top-3 is considered, the correct prediction is among the three highest probability symbols for about 25% of the time in the case of a 0^{th} order predictor, 50% for a 1^{st}, 47% for 2^{nd}, 45% for 3^{rd}, and 50% for the VLMM.

Notice that each character in an attack sequence created from Snort alert description

[1] The reason for choosing the no-repetition data-set will be given in Section 4.2.2.

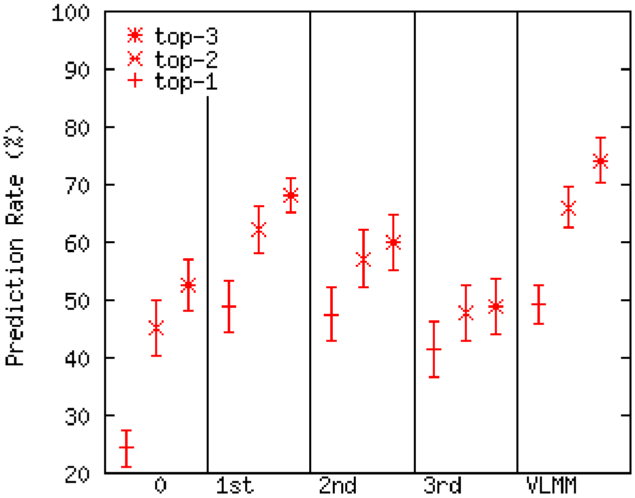

Figure 4.1: Prediction rate using 0^{th}, 1^{st}, 2^{nd}, 3^{rd} order models and VLMM on an alphabet with no repetition based on Snort alert descriptions.

fields can assume 47 different values according to the Skaion-2 data-set. Therefore, a predictor that predicts all possible characters with equal probability would have a $1/47 = 2.1\%$ chance of correctly predicting a sequence character. This figure is significantly smaller than the 25% prediction rate for the 0^{th} order, top-1 prediction.

The 0^{th} order model, which predicts based on frequency counts of previously observed attacks, yields the lowest performance among the predictors shown in Figure 4.1. This shows the importance of having a non-empty context. In other words, consecutive attack actions are indeed correlated, and higher order Markov Models are able to capture such correlation.

The 1^{st} order model, which predicts an action based on its immediate previous one, performs better than 0^{th}, 2^{nd}, 3^{rd} and higher orders. This indicates that the next exploit has a strong correlation with the immediate previous action. One possible explanation is the following. As discussed in Section 3.2.1, prediction is performed by looking for matching contexts for a sequence in the tree. The character that immediately follows a matching context is recorded. These recorded characters are the ones considered during prediction. Longer contexts happen less frequently; therefore, they have fewer matches in the tree. For this reason, the set of recorded characters following a long context tends to be less diverse — contain fewer unique characters — than when a shorter context is taken into account. This leads to higher order models being too restrictive in their prediction, which explains the decreasing prediction rate from 1^{st} to 2^{nd} order, 2^{nd} to 3^{rd}, and so on.

Figure 4.2 also shows the performance of the VLMM. By cleverly combining the results from orders 0^{th}, 1^{st}, etc, the VLMM performs better than any individual model. The end result is a prediction rate of about 75% when top-3 — the highest three predicted symbols — are taken into account. This accuracy is impressive, especially when compared to 6.4%, which is how well a random predictor that assigns equal probabilities to all characters would preform $(3/47 = 6.4\%)$.

Three different types of blending schemes (A, B and C) used by the VLMM are contrasted in Figure 4.2. This figure was also obtained by running the VLMM predictor on sequences with no repetition created based on Snort alert descriptions fields. As noticeable, no difference in performance between the blending schemes is observed.

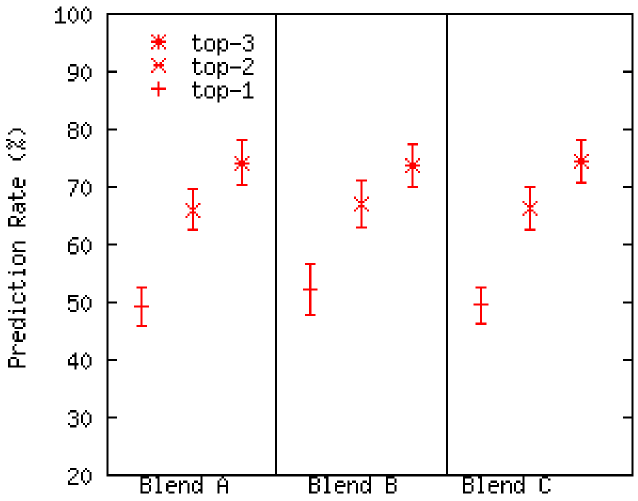

Figure 4.2: Prediction rate using different blending schemes using an alphabet with no repetition based on Snort alert descriptions.

4.2.2 Repetition versus no-repetition data-sets

Experiments were run in order to contrast the prediction rate of a suffix tree constructed from sequences that contain repeated alerts ($\exists\ i \mid x_i = x_{i+1}$) against a tree constructed from sequences that do not contain repetition. The prediction rates were calculated for data-sets with and without repetition created based on description, category and category plus destination IP fields. Notice that the prediction rate of alphabets with repetition, shown in Figure 4.3, is much higher than for alphabets without repetition, shown in Figure 4.4. This happens because sequences such as $s = \{AA...A\}$, where A corresponds to an 'ICMP PING NMAP' reconnaissance/scanning action, are only present in data-sets that allow repetition. These extremely simple sequences contribute to the high prediction rate observed in Figure 4.3.

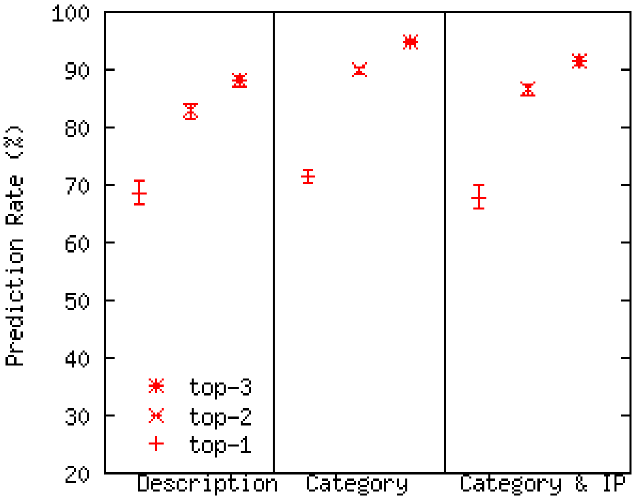

Figure 4.3: Prediction rate for sequences with repetition.

However, note that an important part of characterizing attack behavior is accurately determining the transitions in attack action. For example, if a prediction is to be interpreted by an analyst, he or she will be interested in knowing which new attack action is likely to happen next, as opposed to being informed that the current attack type will happen again. Therefore, in order to make a fair comparison, one must contrast the prediction rate

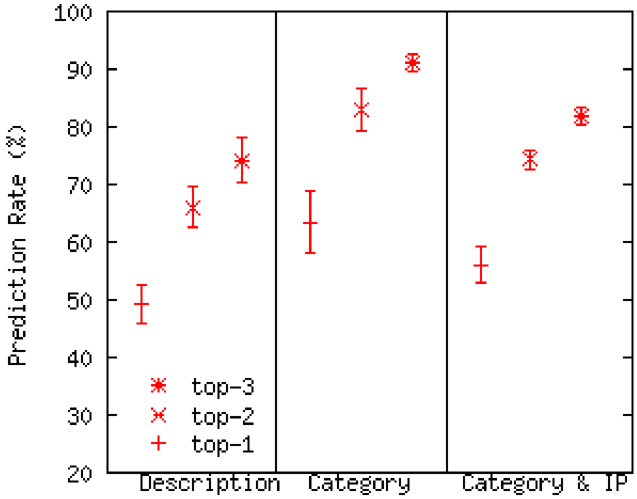

Figure 4.4: Prediction rate for sequences without repetition.

of sequences without repetition against the prediction rate of transitions within sequences with repetition — where a transition is defined as $x_i \neq x_{i+1}$.

The prediction rates for transitions within sequences with repetition are given in Figure 4.5. By contrasting Figure 4.4 with Figure 4.5, one observes that the prediction rate of transitions within sequences with repetition is lower than the prediction rate of sequences without repetition. This confirms the intuition that the high prediction rates exhibited in Figure 4.3 are primarily due to accurately predicting simple sequences — recall the $s = \{AA...A\}$ example given previously.

In addition to the benefit of capturing attack transitions, the no-repetition data-set helps reduce the size of the suffix tree. In the case of alphabets constructed from Snort alert description fields, characters A and B observe the greatest reduction in occurrence when repetitions are removed from the data. Character A represents 'ICMP PING NMAP,' which is a type of reconnaissance scanning action. This character is very likely to appear consecutively in a sequence because many attackers sequentially scan a network looking for possible vulnerabilities. Character B represents a scan for the Squid proxy server and web cache [65]. Vulnerabilities on Squid can be used to gain root access or can be used to cause a denial of service. When exploiting Squid with the intention of causing a denial of service,

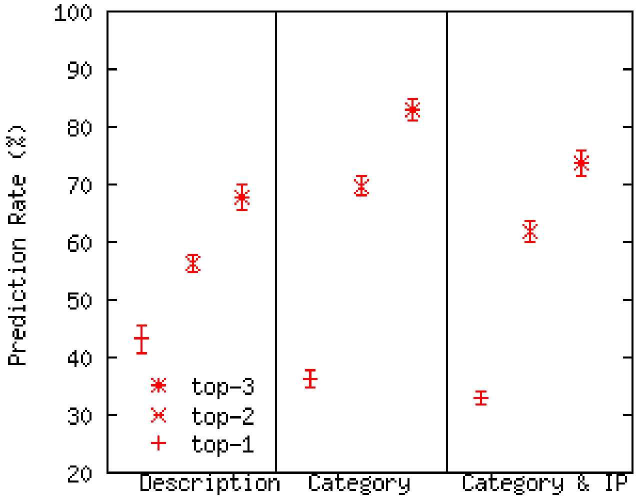

Figure 4.5: Prediction rate of transitions in sequences with repetition.

several requests are sent repeatedly. This creates sequences with innumerable consecutive B characters. Therefore, when repetitions are removed, the occurrence of A and B are greatly reduced from $2,068$ to 139, and from 73 to 3 respectively. These new values are 6.72% and 4.11% of their original frequency count. [2]

Figures 4.6 and 4.7 plot the model size as a function of the data-set size. These figures show the number of nodes in the suffix tree versus the number of characters used to create the tree. Figure 4.6 plots curves for the sizes of two trees, one created from sequences without repetition and another created from sequences with repetition. Sequences with repetition form a much larger data-set and yield to a significantly larger suffix tree. This explains why the dotted curve representing data-sets of sequences with repetition goes to almost $5,000$, while the solid curve, which illustrates no-repetition data-sets, stops at around $1,500$ sequence events.

Figure 4.7 shows that smaller alphabets yield to smaller size trees. For example, a tree built from 600 events based on the category field yields to about $2,000$ nodes. On the other hand, the same amount of description events yield to more than $5,000$ nodes.

[2] Sequences composed of a single type of scan action or a single type of denial of service attack action are removed from the no-repetition data-set.

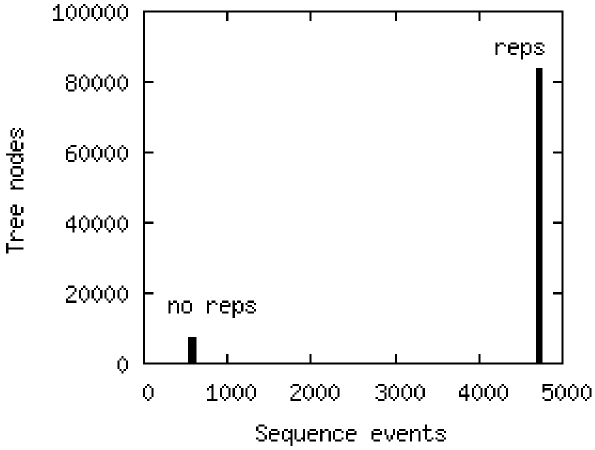

Figure 4.6: Model growth: number of nodes in the suffix tree by number of alerts for non-repetition and repetition data-sets

4.2.3 Attack frequency count and prediction rate

The Skaion-2 data-set is dominated by few characters that tend to repeatedly appear in the same sequence even after it has been processed by the $\Delta t = 1$ second filter. The set also contains many scattered characters that have low frequency counts. Therefore, the frequency of characters representing the description field has a very large variance. In the case of sequences with repetition, the mean character occurrence is 100 ± 338, the median character occurrence is 10, and the nine most occurring characters $\{A,F,J,I,H,W,K,a,B\}$ are responsible for 93% of the alerts as illustrated in Table 4.1. The remaining 37 characters are responsible for only 7% of the data. This analysis is based on Snort alert descriptions; however, similar observations can also be made for alphabets based on category and category plus destination IP.

Even after removing repetitions, the discrepancy in frequency count between high and low frequency characters remains large. The no-repetition data-set's nine most occurring characters are shown on Table 4.2. They account for 85.6% of the data.

Often a low frequency character and its succeeding character are mispredicted since

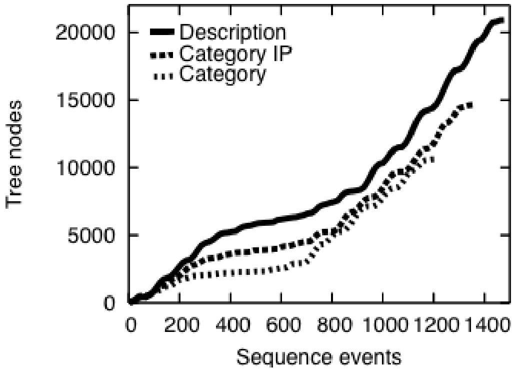

Figure 4.7: Model growth: number of nodes in the suffix tree by number of alerts for three different alphabet choices

there are not enough occurrences for the algorithm to base its prediction. When an infrequent character is the most recently observed symbol, the longest context match into the suffix tree is generally small or non-existent; therefore, the prediction is dominated by a zero order model. Figure 4.8 depicts this phenomena. It shows the suffix tree's predicted probability for characters on Sequence 4.1.

$$+BDBDBDBDBDEDBDBDB* \tag{4.1}$$

'B' represents the category 'Intrusion_Root category', 'D' represents 'Intrusion_Other' and 'E' 'Goal_Dos.'

The example illustrated by Sequence 4.1 and Figure 4.8 can be explained by the fact that sequences with no-repetition created based on alert category are often composed of alternating 'B's and 'D's. Therefore, 'E' (Goal_Dos) is not expected to appear on Sequence 4.1. In this case, the suffix tree model assigns 'E' a relatively low probability. The longest match for the '$BDBDBDBDBDE$' context in the suffix tree is 'E'; therefore, the prediction for the subsequent 'D' is heavily influenced by blending a 0^{th} and a 1^{st} order models that do not contribute with the prediction of the subsequent characters. This explains the dip or depression following 'E' in Figure 4.8.

One would expect high frequency characters to be better predicted by the VLMM.

Char	Freq	Category	Snort Description
A	2,068	Recon_Scanning	ICMP PING NMAP
J	1,057	Intrusion_Root	WEB-MISC Invalid HTTP Version String
F	417	Intrusion_Other	(http_inspect) BARE BYTE UNICODE ENCODING
I	274	Intrusion_Root	NETBIOS SMB-DS IPC$ unicode share access
H	240	Recon_Scanning	ICMP L3retriever Ping
W	99	Goal_Dos	WEB-MISC apache directory disclosure attempt
K	98	Intrusion_Other	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY
a	75	Intrusion_Root	NETBIOS SMB IPC$ unicode share access
B	73	Intrusion_Root	SCAN Squid Proxy attempt
Total	4,401		

Table 4.1: High frequency attack descriptions on sequences with repetitions

Table 4.3 shows that this expectation is true but with a few exceptions. This table shows the average prediction rate and the average frequency of occurrence for characters in the testing set created from Snort alert description fields. The table is sorted with the highest prediction rate characters being presented first. Notice that the characters $\{J, F, H, A, I, K\}$ are also the highest occurring ones, while character D is an exception.

Exception to the correlation between prediction rate and frequency count

Two exceptions to the correlation between prediction rate and frequency count have been observed. The first exception involves patterns that are infrequent but that form *signatures* of high prediction rates. The second exception is related to a phenomenon called *overshadowing* in which high frequency characters yield to low prediction rates.

In the case of description fields, the subsequence 'CD' represents a "MISC OpenSSL Worm traffic" attack being followed by "WEB-MISC bad HTTP/1.1 request, Potentially worm attack." When observing the data-set, one notices that 'D' follows 'C' for all occurrences of 'C'. This indicates a signature for the behavior of the worm attack present in

Char	Freq	Category	Snort Description
J	384	Intrusion_Root	WEB-MISC Invalid HTTP Version String
F	325	Intrusion_Other	(http_inspect) BARE BYTE UNICODE ENCODING
A	139	Recon_Scanning	ICMP PING NMAP
H	127	Recon_Scanning	ICMP L3retriever Ping
I	100	Intrusion_Root	NETBIOS SMB-DS IPC$ unicode share access
K	90	Intrusion_Other	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY
W	60	Goal_Dos	WEB-MISC apache directory disclosure attempt
i	25	Intrusion_Other	(http_inspect) IIS UNICODE CODEPOINT ENCODING
G	20	Intrusion_Root	WEB-IIS nsiislog.dll access
Total	$1,270$		

Table 4.2: High frequency attack descriptions on sequences without repetitions

Skaion-2. The consequence of this signature is that a first order model with a context of 'C' yields a prediction of 'D;' therefore, 'D' is perfectly predicted even though it is a low occurring character.

In addition to signatures, a process that was labeled as overshadowing has also been observed. In this case, a high frequency character is poorly predicted because it shares similar contexts with other high frequency characters that are favored during prediction. Overshadowing is mostly present when only the top-1 prediction is taken into account. For example, from Tables 4.2 and 4.3 it is noticeable that 'K' is among the high frequency characters; however, 'K' shares similar contexts with higher frequency characters 'J' and 'F.' In other words, these three characters often alternate within sequences. Therefore, when taking the top-1 prediction, 'J' and 'F' overshadow 'K,' making its prediction rate drastically drop to 5.9% from 71.5% for a top-3 prediction.

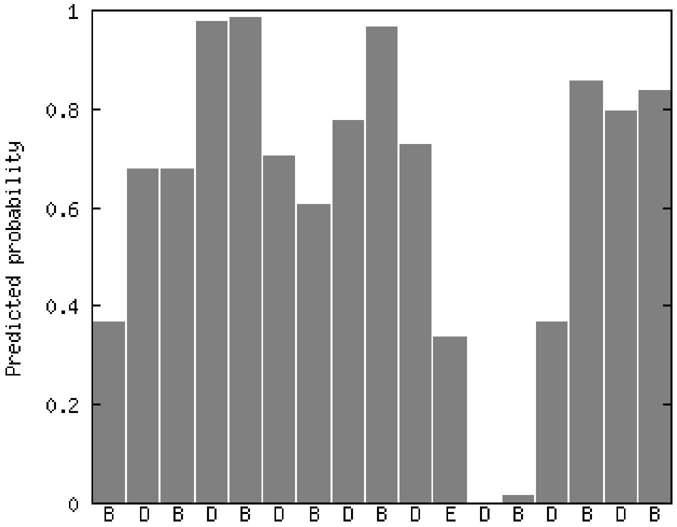

Figure 4.8: Per character probability for Sequence 4.1

4.2.4 Alphabet's level of granularity

Higher levels of granularity allow for more information to be derived from the model. For example, WU-FTPD [2] is a widely deployed software package used to provide File Transfer Protocol (FTP) services on UNIX and Linux systems. There are vulnerabilities in certain versions of WU-FTPD that expose a system to a remote root compromise by anyone with access to the FTP service. A suffix tree constructed from category fields would report an exploitation on WU-FTP simply as "Root_Intrusion" attempt. On the other hand, a suffix tree constructed from description fields would generate the alert message "FTP wu-ftp bad file completion attempt," which is much more informative.

From section 4.2.2 we have that data-sets with no-repetition are better at capturing attacker's courses of actions, and that they require less resources than sets with repetitions. Given these conclusions, this section compares the top-1, top-2 and top-3 prediction rates of no-repetition sets created across the three different proposed levels of granularity (description, category and category with IP fields). Figure 4.4 shows the impact of the alphabet size on the prediction rate. The worst prediction rate happens for the largest alphabet size, which is created based on the 47 Snort alert description fields. The alphabet is of size 17

Char	Avg. Pred. Rate	% Freq. Count
D	100.0%	0.8%
J	94.8%	26.0%
F	91.2%	22.0%
H	85.5%	8.6%
A	78.1%	9.4%
I	74.0%	6.8%
K	71.5%	6.1%

Table 4.3: Highest prediction rate attack descriptions on sequences without repetitions

when category and IP are used, and it yields to the second best prediction rate. The best prediction rate comes from using category only, in which case, the alphabet is the smallest with only 9 unique characters.

Therefore, there is a trade-off between the level of granularity and prediction certainty (or accuracy). The finer the granularity, the more data is needed in order for special cases to be captured with statistical significance.

Blurring may be a good compromise between the level of granularity and the availability of data. Through blurring, one predicts at a coarser level of granularity in the places where there is a lack of training data, or in the cases where un-preceded sequences are observed. This idea is suggested as a possible improvement for future works.

4.2.5 Destination IP

This section demonstrates how behavior can be extracted from the destination IP of attack actions. There is a total of 936 distinct destination IPs and $4,723$ alerts over the $1,113$ sequences belonging to the Skaion-2 data-set of sequences with repetitions. Figure 4.9 shows a histogram of the number of machines per number of attacks in logarithmic scale. For example, 875 out of 936 machines in the data-set are targeted twice. Therefore, the median number of alerts per machine is two. From the graph, one can see that about a dozen of the machines were attacked once, three, four and six times. The average number of alerts per machine is 5.0 ± 62.6. The large standard deviation is due to the fact that the two most frequently targeted machines were victims of 814 and $1,735$ attack actions. This discussion indicates that some targets are preferred by attackers or are more easily accessible than others.

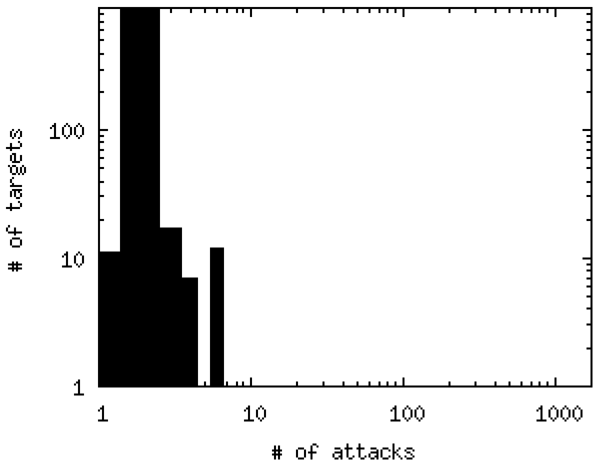

Figure 4.9: Number of targets by number of attacks

Most machines were only victims of two scanning actions belonging to the same attack track. Very few machines were targeted by two or more types of alerts. The machines that were targeted by the greatest number of attack actions are shown in Table 4.4. This table also illustrates how certain machines are targeted by specific types of actions. For example, machine 100.10.20.4 suffered most of the attacks and was the only target of denial of service. Another example is machine 100.20.200.15, which was the only one that experienced reconnaissance actions of type footprinting. The types of services and the processes running on a machine are probable factors that can explain why certain servers are targeted more than others, and why certain attacks are focused on certain machines.

Table 4.5 shows the tendency of an attack action to happen at the same or at a different target machine. This table plots the frequency count for the alphabet created from category and destination IP fields. As explained in Chapter 4.1.1, capital letters represent an alert whose destination IP is the same as its preceding alert's. Lowercase letters, on the other

		Destination IP 100.xxx.xxx.xxx				
Char	Alert category	1.10.101	20.200.15	10.20.20	10.20.3	10.20.4
A	Recon_Scanning	-	-	66	438	2
B	Intrusion_Root	9	14	-	372	$1,075$
	Goal_Backdoor					
C	Trojan_Virus_Worm	6	8	-	-	-
D	Intrusion_Other	9	12	-	-	517
E	Goal_Dos	-	-	-	-	100
F	Goal_Data_Exfiltration	-	-	-	-	-
G	Intrusion_User	5	3	-	4	32
H	Misc_Other	2	-	-	-	9
I	Recon_Footprinting	-	2	-	-	-
	Total	35	39	66	814	$1,735$

Table 4.4: Most frequent attack target machines

hand, represent alerts that happen on a different target machine than the preceding alert.

According to Table 4.5, only 2% of the DoS attack actions were incurred on a different machine than the one targeted by the immediate previous attack action. DoS is an attempt to make a computer resource unavailable to its intended users by saturating the victim machine with external communications requests; therefore, this low tendency for DoS to cover several distinct machines within the same attack sequence is quite expected.

Also according to Table 4.5, 21.6% and 16.7% of the "data exfiltration" and "backdoor, trojan, virus and worm" attacks happened on a different target machine from their preceding attack actions. When comparing these figures with the percentages for other attack categories, one concludes that "data exfiltration" and "backdoor, trojan, virus and worm" are the least likely to target a specific machine.

Figure 4.10 provides an insightful representation for these attack transitions. The x-axis represents the number of transitions in attack target within an attack sequence. The y-axis represents the number of unique target machines visited by the attack sequence; that is, the number of distinct destination IPs. For example, the point $(9, 3)$ represents a sequence in which the attacker hopped 9 times across 3 different machines.

Char	Freq	% of alerts w\ same target IP as previous	Category
A	$2,199$		Recon_Scanning
a	120	5.2%	
B	$1,469$		Intrusion_Root
b	114	7.2%	
C	20		Goal_Backdoor_Trojan_Virus_Worm
c	4	16.7%	
D	528		Intrusion_Other
d	35	6.2%	
E	98		Goal_Dos
e	2	2.0%	
F	58		Goal_Data Exfiltration
f	16	21.6%	
G	40		Intrusion_User
g	7	14.9%	
H	10		Misc_Other
h	1	10%	
I	2	-	Recon_Footprinting

Table 4.5: Attack frequency count based on the combination of destination IP and category fields

Points are expected to lay in the region between the solid and dashed lines. Points cannot fall outside this region. The solid line delimits the maximum number of target machines that an attacker can visit. For example, if there are 8 transitions within an attack sequence, then the attacker can have visited at most 9 different machines. This line is defined as $y = x + 1$. The dashed line represents attacks that visit the minimum number of target machines. Two is the minimum number of machines visited by a sequence that contains at least one transition.

In Figure 4.10, the size of the squares is logarithmically proportional to the number of attack tracks that fall within the given point. For example, most of the attack sequences in our data lie in $(0, 1)$, which is represented by the large square. The $(0, 1)$ coordinate tells that the attacker targeted only one machine; therefore there are no transitions within his or hers attack track.

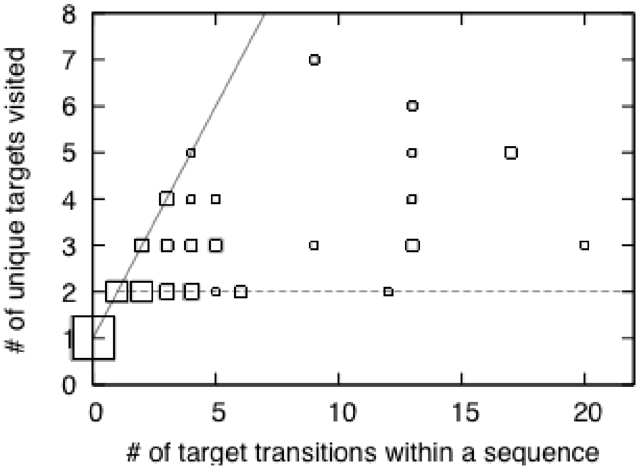

Figure 4.10: Attack target graph: number of unique targets visited by the number of target transitions within a sequence

Figure 4.10 can be used to characterize attack behavior. Points close to the $y = 2$ line that have $x >> 1$ represent attacks that hop many times across a small number of machines. For example, the point $(20, 3)$ represents an attack sequence that contained 20 transitions covering only three different destination IPs. On the other hand, points closed to or at the $y = x + 1$ line that have $x >> 1$ indicate attacks that hop across many distinct machines. For example, $(9, 7)$ represents an attack where 7 different machines were visited through 9 hops.

Note that the topology of the network used when creating the Skaion-2 data-set is not openly available. Despite the absence of this crucial pice of information, it is still possible to derive many insights about attack patterns as shown in the discussion above.

4.3 Entropy

The entropy of correctly predicted and mispredicted characters are given on Table 4.6. Notice that, despite the extremely large standard deviation, the characters that were mispredicted had, on average, higher entropy.

	Repetition	Category	Category IP	Description
Correct predictions	no	0.62 ± 0.48	$.91 \pm .60$	1.07 ± 0.69
Mispredictions	no	0.93 ± 0.63	$1.41 \pm .81$	1.35 ± 0.71
Correct predictions	yes	0.52 ± 0.51	0.58 ± 0.57	0.58 ± 0.68
Mispredictions	yes	0.88 ± 0.63	1.04 ± 0.75	1.23 ± 0.92

Table 4.6: Entropy of correctly predicted and mispredicted characters across three alphabet choices

From Table 4.6, one observes that higher entropies are recorded for sequences without repetition. This is in accordance with Section 4.2.2, where its been shown that the repetition data-set contains many easily predicted sequences that are filtered out of the no-repetition set. Also, in Section 4.2.4 it was shown that larger alphabets yield to sequences that are harder to predict. This is supported by the entropy for alphabets created from description fields being higher than for category and IP, which, in turn, is higher than for category alone — the only exception is category and IP compared to description in the case of sequences with repetition.

4.4 Average log-loss

Each sequence in the test set is classified according to its average log-loss value. Sequences whose average log-loss is above a threshold are labeled as "sophisticated," and labeled as "simple" otherwise. The prediction rate for each group of sequences (sophisticated and simple) is computed and presented on Tables 4.7 and 4.8. These tables are constructed based on four threshold values (1.5, 2.0, 2.5 and 3.0) chosen empirically. Notice that the prediction rate of sequence that were classified as sophisticated is lower than that of simple sequences.

The threshold is a parameter used to indicate how unusual a sequence must be in order for it to be classified as sophisticated. Figures 4.11 and 4.12 illustrate a procedure for

	Threshold	Category	Category IP	Description
Simple Sophisticated	1.5	0.91 0.7	0.86 0.52	- 0.52
Simple Sophisticated	2.0	0.83 0.69	0.91 0.50	0.85 0.51
Simple Sophisticated	2.5	0.79 0.65	0.79 0.48	0.80 0.49
Simple Sophisticated	3.0	0.77 0.60	0.66 0.46	0.71 0.47

Table 4.7: Prediction rate of sequences without repetitions that were classified as simple and sophisticated based on a threshold of the average log-loss

	Threshold	Category	Category IP	Description
Simple Sophisticated	1.5	0.99 0.54	0.99 0.52	1 0.52
Simple Sophisticated	2.0	0.94 0.53	0.94 0.51	0.95 0.51
Simple Sophisticated	2.5	0.84 0.51	0.86 0.50	0.91 0.48
Simple Sophisticated	3.0	0.75 0.48	0.74 0.41	0.89 0.46

Table 4.8: Prediction rate of sequences with repetitions that were classified as simple and sophisticated based on a threshold of the average log-loss

determining the threshold. These figures were constructed based on ten runs of the algorithm with different training and testing sets. During these runs, the average log-loss for sequences in the testing set was computed and a cumulative distribution function (CDF) was plotted. Figure 4.11 shows that a threshold of little over 2 would cause half of the sequences generated from the category field to be classified as simple, and half as sophisticated. Figure 4.11 shows that, in order to split the set in half, one would have to choose a threshold of about 4.

The steeper slope in Figure 4.11 indicates that, for Skaion-2, thresholding based on

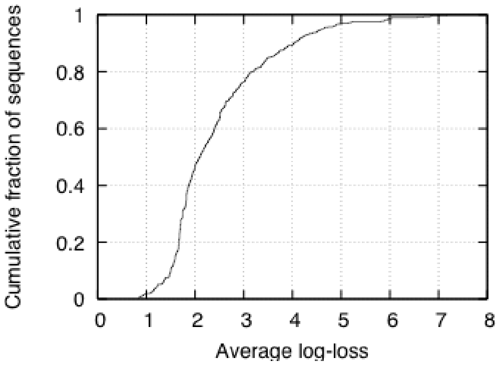

Figure 4.11: Cumulative Distribution Function (CDF) based on the average log-loss of sequences created from the category field

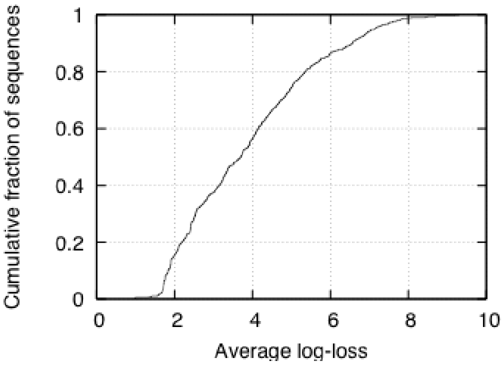

Figure 4.12: Cumulative Distribution Function (CDF) based on the average log-loss of sequences created from the description field

category draws a sharper line between simple and sophisticated sequences when compared to thresholding based on description. An additional experiment was devised in order to determine whether category and description fields classify the same (or similar sequences) as the simple and sophisticated. If the average log-loss of sequences based on category and description fields have a correlation, then the same sequences would be attributed similar average log-losses regardless of whether they were constructed based on category or description.

For some sequences, their description version had a significantly higher average log-loss than their category-based counterpart. For example, description-based sequences $+CD*$, $+CDCD*$, $+Fi$ and $+VF*$ were attributed high average log-losses. However, when translated to category, they became $+CC*$, $+CCCC*$, $+DD*$ and $+GD*$ respectively. These were either filtered out of the no-repetition data-set, since they consist of one type of character only, or were attributed low average log-losses. Therefore, despite the fact that category-based sequences produce a steeper CDF, the amount of information lost in the category-based representation presents a drawback during classification.

4.5 Summary of results

- The 1^{st} order Markov Model yields to higher prediction rates than 0^{th}, 2^{nd} and 3^{rd}. This indicates that a future attack action has a strong correlation with the attacker's immediately previous action.

- The VLMM performs better than the 1^{st} order model, which indicates the benefit in using 0^{th}, 2^{nd}, 3^{rd} order models in addition to 1^{st}. The VLMM requires a "cleaver" linear combination of the predictions from these several Markov Models. Three blending schemes were presented and no difference in prediction rate among them was observed. However, future works may explore other blending schemes.

- Entropy can be used to measure the variability and predictability of an attack within a sequence.

- Removing repetitions from attack sequences $s = \{x_1, x_2, ..., x_n\}$; that is, removing x_{i+1} if $x_{i+1} = x_i$, greatly reduces the size of the data-set without jeopardizing the model's ability to predict future attack actions.

- Attacks that occur more frequently in the data-set tend to be better predicted. This raises the question of how to improve the prediction rate of low occurring characters. Also, interesting exceptions to the correlation between frequency and prediction rate emerged. Low occurring characters that have high prediction rates may belong to specific attack *signatures*. On the other hand, highly occurring characters may be *overshadowed* by others if they share similar contexts. This problem is mitigated by considering n predictions returned by the model (top-n) instead of only taking the highest prediction (top-1) into consideration.

- Fine grained alphabets created from Snort alert description fields provide more detailed information about attacks than alphabets created from alert categories. However, since coarser alphabets have higher prediction rates and lower entropy, there is a trade-off between information granularity and prediction accuracy.

- Destination IP allows for the identification of patterns in attack target machines. Besides relating the types of attack to their target machines, it was also possible to measure the diversity of targets in an attack sequence and relate it to the attacker's tendency to "hop" across target machines.
- Average log-loss can be used to identify complex attack sequences.

Chapter 5

Conclusion and future work

This thesis explores a framework used to characterize cyber attack behavior. The objective is to model the sequential properties of attacks captured by IDSs and grouped by correlation engines into attack tracks. The model is trained from representative attack tracks[1] and, once a newly unfolding attack sequence is observed, the proposed framework uses information entropy to measure the variability in future attack actions and makes a prediction for what the next action will be. In addition, average log-loss is used to identify “sophisticated” or “unusual” attacks. Finally, guidelines for investigating attackers’ target machines are also suggested in this thesis.

Different from previous works, which have focused on the development of IDSs and of correlation engines, this thesis builds upon and beyond these maturing technologies in order to propose a novel approach: the application of universal predictors in the context of cyber security. These predictors are not built based on a priori expectations. Neither are they built from field expert knowledge. Instead, they adapt to the training set.

Attack behavior is a function of the network topology, network equipment, operating systems and services, of the hacker’s expertise and intent, etc. Therefore, the true potential of universal predictors, which are constructed free from the biases of a particular domain, will be seen when these predictors are overlaid on top of domain aware models such as virtual cyber terrains [17]. Virtual cyber terrains capture the relationships between hosts,

[1] The model can be trained on-line; in other words, it can be trained in real-time as IDS alerts are perceived and are correlated into attack tracks.

services, and privileges in a computer network. Overlaying attack prediction onto these terrains will allow a much more comprehensive assessment of cyber attacks. This approach also breaks the complex task of cyber attack analysis into two subtasks: prediction and cyber terrain modeling. The subtasks can then be investigated independently with their results merged when appropriate.

This thesis tackled the problem of prediction and attack characterization with a Variable Length Markov Model. The VLMM was chosen given the intuition that an attacker's next action is strongly correlated with his or hers previous ones. From this work, one concludes that the strongest correlation happens between two consecutive actions, and that correlation decreases as longer contexts are taken into account. This work also shows that by blending several order Markov Models, the VLMM achieves greater performance than any individual Markov Model.

However, we have recently noticed VLMM's intolerance to noise. For example, assume an unfolding attack track t that is converted to a sequence s. Let s' be a sequence generated when the i^{th} attack action in t is not detected by the IDSs or is incorrectly classified by the correlation engine. It can be shown that, in the case of the VLMM, the prediction of future actions in t is heavily dependent on s. When s' is taken in place of s, the performance of the prediction will vary. A similar issue may occur when the hacker purposely introduces noise by performing actions with the intent of generating extra alerts that are not related to his or hers true goal. Since this is likely to degrade the performance of the predictor, one crucial open question is how to build invariance into the model.

Hidden Markov Models (HMMs), also known as finite-state models, may provide further insight into the dynamics of cyber attack behavior. Chapter 2 shows how Markov Models and VLMMs capture the dynamics of k*-testable* systems, which are systems that produce events with finite contexts. Hidden Markov Models were also presented in that chapter. They are usually more complex and are capable of characterizing non k-testable systems, such as the ones in which *counting* occurs. Counting accommodates for a class of systems where events can be arbitrarily far apart from one another. Since attack tracks are finite by definition (and are usually short), accommodating for non k-testable systems did not seem to be a crucial pre-requisite for the model. However, HMMs may have the advantage of being more robust to noise than VLMMs. For example, in the case of HMMs, the accuracy in the predicted probability of occurrence of events depends on the state of the model. Therefore, HMMs are robust as long as a disturbance (due to noise or to an intentionally misleading attack action) is small enough not to cause an undesirable state transition.

One of the challenges in this area of research is the lack of data for testing these hypothesis. The cyber security community has developed openly available data-sets composed of raw attacks and background noise. These sets have been used to test new IDSs and correlation engines. However, there is a lack of data composed of correlated alerts that can be used to test universal predictors.

In addition to presenting results that may have immediate applications, this thesis is intended to motivate the investigation of universal predictors in the cyber domain context, and to motivate further development of models for noisy non-stationary sources where the observables do not belong to a single time series but, instead, are composed of several finite sequences such as cyber attack tracks.

Bibliography

[1] United States Code 44, chapter 35, subchapter iii, 3542, 2006.

[2] Wu-ftpd development group, February 2006. http://www.wu-ftpd.org/.

[3] Ant Allan. Enterasys Networks Dragon Intrusion Detection System (IDS). Technical Report DPRO-105094, Enterasys Networks, Inc., March 2002.

[4] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa, and S. Goto. A new intrusion detection method based on discriminant analysis. In *IEICE Transactions IEICE Transactions on Information and Systems*, pages 570–577, 2001.

[5] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order Markov Models. In *Journal of Artificial Intelligence*, pages 385–421, 2004.

[6] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, 1990.

[7] Michael K. Bergman. The deep web: Surfacing hidden value. White paper, July 2000.

[8] P. Buhlmann and A. Wyner. Variable length markov chains. Technical report, Dept. of Statistics, University of California, Berkeley, 1997.

[9] I-Cheng K. Chen, John T. Coffey, and Trevor N. Mudge. Analysis of branch prediction via data compression. In *Architectural Support for Programming Languages and Operating Systems*, pages 128–137, 1996.

[10] DEFCON conference. DEFCON capture the flag (CTF) contest. http://www.defcon.org/.

[11] G. V. Cormack and R. N. S. Horspool. Data compression using dynamic markov modeling. In *The Computer Science Journal*, volume 30, pages 541–550, 1987.

[12] Microsoft Corp. IIS.net. http://www.iis.net/.

[13] KDD Cup. KDD Cup data, 1999. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[14] William DuMouchel and Matthias Schonlau. A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities. In *Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics*, 1998.

[15] James Early, Carla Brodley, and Catherine Rosenberg. Behavioral authentication of server flows. In 19^{th} *Annual Computer Security Applications Conference*, 2003.

[16] A. Ehrenfeucht and J. Mycielski. A pseudorandom sequence – how random is it? In *American Mathematical Monthly*, volume 99, pages 373–375, 1992.

[17] Daniel Fava, Jarred Holsopple, and Shanchieh Yang. Terrain and behavior modeling for projecting multistage cyber attacks. In 10^{th} *International Conference on Information Fusion*, Québec City, Canada, 2007.

[18] Laura Feinstein, Dan Schnackenberg, Ravindra Balupari, and Darrell Kindred. Statistical approaches to DDoS attack detection and response. In *DARPA Information Survivability Conference and Exposition*, 2003.

[19] R. T. Fielding and G. Kaiser. The apache http server project. In *IEEE Internet Computing Magazine*, 1997.

[20] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[21] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In 1^{st} *Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, 1999.

[22] Robert Giegerich and David Wheeler. Pairwise sequence alignment.

[23] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *International World Wide Web Conference*, 2005.

[24] J. Haines, D. K. Ryder, L. Tinnel, and S. Taylor. Validation of sensor alert correlators. In *IEEE Security and Privacy Magazine*, Jan./Feb. 2003.

[25] Steven Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. In *Journal of Computer Security*, volume 6, pages 151–180, 1998.

[26] Philippe Jacquet, Wojciech Szpankowski, and Izydor Apostol. A universal predictor based on pattern matching. In *IEEE Transactions on Information Theory*, volume 48, pages 1462–1472, 2002.

[27] Guogei Jiang. Weak process models for robust process detection. In *Proceedings of the Society for Photo-optical Instrumentation Engineers (SPIE)*, 2004.

[28] Guogei Jiang and George Cybenko. Temporal and spatial distributed event correlation for network security. In *Proceedings of the American Control Conference*, 2004.

[29] W-H Ju and Y. Vardi. A hybrid high-order markov chain model for computer intrusion detection. Technical report, National Institute of Statistical Sciences, 19 T. W. Alexander Drive, PO Box 14006, Research Triangle Park, NC 27709-4006, 1999.

[30] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating against common enemies. In *Internet Measurement Conference*, 2005.

[31] Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, 1999.

[32] Matthew B. Kennel and Alistair I. Mees. Context-tree modeling of observed symbolic dynamics. In *Physical Review*, 2002.

[33] John Kieffer. Prediction & information theory. In *Six Lectures on Information Theory*.

[34] Calvin Ko, George Fink, and Karl Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the* 10^{th} *Annual Computer Security Applications Conference*, pages 134–144, 1994.

[35] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *IEEE Symposium on: Security and Privacy*, 1997.

[36] Andrew Kosoresow and Steven Hofmeyr. Intrusion detection via system call traces. In *IEEE Software*, volume 14, pages 35–42, 1997.

[37] J. Kevin Lanctot, Ming Li, and En hui Yang. Estimating DNA sequence entropy. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 409–418, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[38] Terran Lane and Carla Brodley. Temporal sequence learning and data reduction for anomaly detection. In *ACM Transactions on Information and System Security*, volume 2, pages 295–331, 1999.

[39] Terran Lane and Carla E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[40] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. In *Information and System Security*, volume 3, pages 227–261, 2000.

[41] Wenke Lee, Salvatore J. Stolfo, and Philip K. Chan. Learning patterns from Unix process execution traces for intrusion detection. In *Proceedings of the AAAI97 workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56, 1997.

[42] Yihua Liao and V. Rao Vemuri. Using text categorization techniques for intrusion detection. In 11^{th} *USENIX Security Symposium*, pages 51–59, 2002.

[43] Massachusetts Institute of Technology Lincoln Laboratory. DARPA intrusion detection evaluation, 2001. http://www.ll.mit.edu/IST/ideval/data/data_index.html.

[44] Robert W. Lucky. *Silicon Dreams: Information, Man, and Machine*. St. Martin's Press, 1989.

[45] Alistar Moffat and Andrew Turpin. *Compression and Coding Algorithms*. Kluwer Academic Publisher, 2002.

[46] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet Denial-of-Service activity. In *ACM Transactions on Computer Systems*, pages 115—139, 2006.

[47] Trevor N. Mudge, I-Cheng Chen, and John Coffey. Limits to branch prediction. Technical Report CSE-TR-282-96, University of Michigan, 1996.

[48] Kihong Park and Heejo Lee. On the effectiveness of probabilistic packet marking for IP traceback under Denial of Service Attack. In *INFOCOM*, pages 338–347, 2001.

[49] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In 20^{th} *Annual Computer Security Applications Conference*, 2004.

[50] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

[51] Jorma Rissanen. A universal data compression system. In *IEEE Transactions on Information Theory*, volume 29, pages 656–664, 1983.

[52] Ken Robinson. *Out of Our Minds: Learning to be Creative*. Capstone, 2001.

[53] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, 1999.

[54] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. In *Journal of Machine Learning*, volume 25, pages 117–149, 1996.

[55] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for IP traceback. In *SIGCOMM*, 2000.

[56] Cosma R. Shalizi and Kristina L. Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *Proceedings of the* 20^{th} *Conference on Uncertainty in Artificial Intelligence*, pages 504–511, 2004.

[57] Cosma R. Shalizi, Kristina L. Shalizi, and James P. Crutchfield. An algorithm for pattern discovery in time series. In *SFI Working Paper*, 2002.

[58] Cosma R. Shalizi, Kristina L. Shalizi, and James P. Crutchfield. Pattern discovery in time series, part i: Theory, algorithm, analysis, and convergence. In *Journal of Machine Learning Research Working Paper*, 2002.

[59] Moises Sudit, Adam Stotz, Michael Holender, William Tagliaferri, and Kathie Canarelli. Measuring situation awareness and resolving inherent high-level fusion

obstacles. In Belur V. Dasarathy, editor, *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications*, 2006.

[60] Peter Tino and Georg Dorffner. Predicting the future of discrete sequences from fractal representations of the past. In *Journal of Machine Learning*, volume 45, pages 187–217, 2001.

[61] Daniel R. Upper. Theory and algorithms for hidden markov models and generalized hidden markov models, 1997.

[62] Prem Uppuluri and R. Sekar. Experiences with specification-based intrusion detection. In *Lecture Notes in Computer Science*, pages 172–194, 2001.

[63] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. *IEEE Transactions on Dependable and Secure Computing*, 01(3):146–169, 2004.

[64] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusion using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[65] Wessels and Claffy. Application of Internet Cache Protocol (ICP), version 2. Technical Report RFC2187, National Laboratory for Applied Network Research/UCSD National Laboratory for Applied Network Research, UCSD, 1997. http://www.squid-cache.org/.

[66] F. Willems, Y. Shtarkov, and T. Tjalkens. The context-tree weighting method: basic properties. In *IEEE Transactions on Information Theory*, 1995.

[67] Simon Yates. Worldwide PC adoption forecast, 2007 to 2015. White paper, June 2007.

[68] Nong Ye, Xiangyang Li, Qiang Chen, S.M. Emran, and Mingming Xu. Probabilistic techniques for intrusion detection based on computer audit data. In *IEEE Transactions on Systems, Man and Cybernetics*, pages 266–274, 2001.

[69] Nong Ye, Yebin Zhang, and Connie M. Borror. Robustness of the markov-chain model for cyber-attack detection. In *IEEE Transactions on Reliability*, volume 53, pages 116–123, 2004.