APPLICATIONS OF DATA MINING IN COMPUTER SECURITY

APPLICATIONS OF DATA MINING IN COMPUTER SECURITY

Edited by

DANIEL BARBARÁ
George Mason University

SUSHIL JAJODIA
George Mason University

Kluwer Academic Publishers
Boston/Dordrecht/London

Foreword

Welcome to the sixth volume of the Kluwer International Series on ADVANCES IN INFORMATION SECURITY. The goals of this series are, one, to establish the state of the art of, and set the course for future research in information security and, two, to serve as a central reference source for advanced and timely topics in information security research and development. The scope of this series includes all aspects of computer and network security and related areas such as fault tolerance and software assurance.

ADVANCES IN INFORMATION SECURITY aims to publish thorough and cohesive overviews of specific topics in information security, as well as works that are larger in scope or contain more detailed background information than can be accommodated in shorter survey articles. The series also serves as a forum for topics that may not have reached a level of maturity to warrant a comprehensive textbook treatment.

The success of this series depends on contributions by researchers and developers such as you. If you have an idea for a book that is appropriate for this series, I encourage you to contact me. I would be happy to discuss any potential projects with you. Additional information about this series can be obtained from www.wkap.nl/series.htm/ADIS.

About This Volume

This sixth volume of the series is entitled *Applications of Data Mining in Computer Security*, edited by Daniel Barbará and Sushil Jajodia.

Computer intrusions are becoming commonplace and outpacing our capacity to detect, analyze, and counteract them. Since intrusions usually leave traces in the audit data trails, it is only natural to think about this problem in a data-centered way. Some research groups have been successfully using data mining techniques for effectively implementing tools to detect and analyze intrusions.

This volume offers nine articles from leading researchers; eight of these articles focus on the use of data mining for intrusion detection, including one that surveys the state of modern intrusion detection using data mining approaches and another that critically examines these approaches. The last article deals with the application of data mining to computer forensics. Collectively, these articles provide a comprehensive summary of current findings in this fruitful research field.

SUSHIL JAJODIA

Consulting Editor

Contents

List of Figures xiii

List of Tables xviii

Preface xix

1
Modern Intrusion Detection, Data Mining, and Degrees of Attack Guilt 1
Steven Noel, *Duminda Wijesekera* and *Charles Youman*

1. Introduction 2
2. Detection Strategies 3
   - 2.1 Misuse Detection 4
   - 2.1.1 Expert Systems 4
   - 2.1.2 Signature Analysis 5
   - 2.1.3 State-Transition Analysis 6
   - 2.1.4 Data Mining 7
   - 2.1.5 Other Approaches 8
   - 2.2 Anomaly Detection 9
   - 2.2.1 Statistical Methods 9
   - 2.2.2 Expert Systems 10
   - 2.2.3 Data Mining 10
   - 2.2.4 Other Approaches 12
3. Data Sources 13
4. Degrees of Attack Guilt 14
   - 4.1 Misuse Detection 16
   - 4.1.1 Knowledge-Based Methods 16
   - 4.1.2 Machine-Learning Methods 17
   - 4.2 Anomaly Detection 18
   - 4.2.1 Knowledge-Based Methods 18
   - 4.2.2 Statistical Methods 19
   - 4.2.3 Machine-Learning Methods 20
5. Conclusion 25

References 25

2
Data Mining for Intrusion Detection 33

Klaus Julisch
1. Introduction 33
2. Data Mining Basics 34
2.1 Data Mining, KDD, and Related Fields 34
2.2 Some Data Mining Techniques 36
2.2.1 Association Rules 37
2.2.2 Frequent Episode Rules 38
2.2.3 Classification 39
2.2.4 Clustering 40
2.3 Research Challenges in Data Mining 41
3. Data Mining Meets Intrusion Detection 42
3.1 MADAM ID 43
3.2 ADAM 45
3.3 Clustering of Unlabeled ID Data 46
3.4 Mining the Alarm Stream 47
3.5 Further Reading 49
4. Observations on the State of the Art 50
4.1 Data Mining, but no Knowledge Discovery 50
4.2 Disregard of Other KDD Steps 51
4.3 Too Strong Assumptions 52
4.4 Narrow Scope of Research Activities 53
5. Future Research Directions 54
6. Summary 56

References 57

3
An Architecture for Anomaly Detection 63
Daniel Barbará, *Julia Couto*, *Sushil Jajodia* and *Ningning Wu*
1. Introduction 63
2. Architecture 65
2.1 Filter 65
2.2 Profile 67
2.3 Profile Builder 67
2.4 Diagnoser 67
3. ADAM: an implementation of the architecture 67
4. Experiences 72
5. Breaking the dependency on training data 74
6. Future 74

References 75

4
A Geometric Framework for Unsupervised Anomaly Detection 77
Eleazar Eskin, *Andrew Arnold*, *Michael Prerau*, *Leonid Portnoy* and *Sal Stolfo*
1. Introduction 78
2. Unsupervised Anomaly Detection 81
3. A Geometric Framework for Unsupervised Anomaly Detection 83
3.1 Feature Spaces 83
3.2 Kernel Functions 84
3.3 Convolution Kernels 85

4. Detecting Outliers in Feature Spaces 85
5. Algorithm 1: Cluster-based Estimation 86
6. Algorithm 2: K-nearest neighbor 87
7. Algorithm 3: One Class SVM 89
8. Feature Spaces for Intrusion Detection 91
8.1 Data-dependent Normalization Kernels 92
8.2 Kernels for Sequences: The Spectrum Kernel 93
9. Experiments 93
9.1 Performance measures 93
9.2 Data Set Descriptions 94
9.3 Experimental Setup 96
9.4 Experimental Results 96
10. Discussion 97

References 98

5
Fusing a Heterogeneous Alert Stream into Scenarios 103
Oliver Dain and *Robert K. Cunningham*
1. Introduction 104
2. Fusion Approach 105
3. Architecture 106
4. Definitions 107
5. Probability Assignment 108
5.1 Data Sources and Use 108
5.2 Naïve Technique 111
5.3 Heuristic Technique 112
5.4 Data Mining Techniques 114
6. Experimental Results 115
6.1 Naïve Technique 116
6.2 Heuristic Technique 117
6.3 Data Mining Techniques 117
7. System Benefits 119
8. Discussion and Summary 120

References 120

6
Using MIB II Variables for Network Intrusion Detection 123
Xinzhou Qin, *Wenke Lee*, *Lundy Lewis* and *João B. D. Cabrera*
1. Introduction 124
2. Background 125
2.1 MIB II 125
2.2 Entropy and Conditional Entropy 126
3. Model Construction 127
3.1 Model Architecture 127
3.2 Anomaly Detection Module 129
3.2.1 Anomaly Detection Model Design Overview 129
3.2.2 Anomaly Detection Module Construction 129
4. Experiments and Performance Evaluation 134

4.1 Normal Data Sets 134
4.2 Evaluation under Attacks 135
4.2.1 Misuse Detection 135
4.2.2 Anomaly Detection 141
5. Discussion 147
6. Related Work 149
7. Conclusions and Future Work 150

References 150

7
Adaptive Model Generation 153
Andrew Honig, *Andrew Howard*, *Eleazar Eskin* and *Sal Stolfo*
1. Introduction 154
2. Components of Adaptive Model Generation 157
2.1 Real Time Components 159
2.2 Data Warehouse 163
2.3 Detection Model Management 165
2.4 Data Analysis Engines 167
2.5 Efficiency consideration 174
3. Capabilities of Adaptive Model Generation 175
3.1 Real Time Detection Capabilities 175
3.2 Automatic Data Collection and Data Warehousing 175
3.3 Model Generation and Management 176
3.4 Data Analysis Capabilities 176
3.5 Correlation of Multiple Sensors 178
4. Model Generation Algorithms 179
4.1 Misuse Detection 179
4.2 Anomaly Detection 179
4.3 Unsupervised Anomaly Detection 180
5. Model Generation Example: SVM 180
5.1 SVM Algorithm 181
5.2 SVM for Misuse Detection in AMG 183
5.3 Unsupervised SVM Algorithm 183
5.4 Unsupervised SVM for Unsupervised Anomaly Detection 184
6. System Example 1: Registry Anomaly Detection 185
6.1 The RAD Data Model 185
6.2 The RAD Sensor 186
6.3 The RAD Classification Algorithm 186
6.4 The RAD Detector 187
7. System Example 2: HAUNT 187
7.1 HAUNT Sensor 188
7.2 HAUNT Classification Algorithm 188
7.3 HAUNT Detector 189
7.4 HAUNT Feature Extraction 189
8. Conclusion 190

References 192

8
Proactive Intrusion Detection 195

João B. D. Cabrera, *Lundy Lewis*, *Xinzhou Qin*, *Wenke Lee* and *Raman K. Mehra*

1. Introduction 196
2. Information Assurance, Data Mining, and Proactive Intrusion Detection 198
   - 2.1 Intrusion Detection Systems 198
   - 2.2 A Thought Experiment 198
   - 2.3 Proactive Intrusion Detection 204
3. A methodology for discovering precursors - Assumptions, Objectives, Procedure and Analysis 206
   - 3.1 Notation and Definitions 206
   - 3.1.1 Time Series, Multivariate Time Series and Collections 206
   - 3.1.2 Events, Event Sequences, Causal Rules and Precursor Rules 207
   - 3.2 Assumptions, Problem Set-Up, Objectives and Procedure 208
   - 3.3 Analysis - Detection and Gradation of Causality in Time Series 211
   - 3.3.1 Notation and Definitions 211
   - 3.3.2 The Granger Causality Test as an Exploratory Tool 212
   - 3.3.3 GCT and the Extraction of Precursor Rules - Modeling and Theoretical Developments 213
4. A Case Study - Precursor Rules for Distributed Denial of Service Attacks 217
   - 4.1 DDoS Attacks and the experiments 217
   - 4.2 TFN2K Ping Flood - Extracting Precursor Rules 219
5. Conclusions 222

References 223

9

E-mail Authorship Attribution for Computer Forensics 229

Olivier de Vel, *Alison Anderson*, *Mal Corney* and *George Mohay*

1. Introduction and Motivation 230
   - 1.1 Computer Forensics 230
   - 1.2 E-mail Forensics 232
2. Authorship Attribution 234
3. E-mail Authorship Attribution 238
4. Support Vector Machine Classifier 239
5. E-mail Corpus and Methodology 240
6. Results and Discussion 245
7. Conclusions 246

References 247

Index 251

List of Figures

1.1	General Degrees of Attack Guilt	15
1.2	Degrees of Attack Guilt for Knowledge-based Misuse Detection	16
1.3	Degrees of Attack Guilt for Machine-learning Misuse Detection	18
1.4	Degrees of Attack Guilt for Knowledge-based Anomaly Detection	19
1.5	Degrees of Attack Guilt for Statistical Anomaly Detection	20
1.6	Combining data mining and classification for anomaly detection	21
1.7	Degrees of Guilt for the Training Phase	22
1.8	Degrees of Guilt for Connection Mining during Detection	23
1.9	Degrees of Guilt for Attack Classification during Detection	23
1.10	Degrees of Guilt with respect to Attack Classification Confidence	24
1.11	Degrees of Guilt for Overall Machine-learning Approach	24
2.1	Data mining process of building misuse detection systems.	44
3.4	Results of using ADAM in the 1999 Lincoln Labs competition data.	73
5.1	(a) Possible scenario assignments for a new alert, C, given 2 existing alerts, A, and B. (b) Possible scenario assignments for a new alert, D, given 3 existing alerts, A, B, and C.	106
5.2	Alert occurrence times in the DEF CON data. Each horizontal line depicts a single scenario. The points on the line indicate when an alert occurred. The symbol indicates the class of the alert. Depicted are the twenty five scenarios containing the largest number of alerts.	109

5.3	Decision surface for a single transition type. The height of the curve is the probability that two alerts in the given time span with the given r value belong in the same scenario. The black plane is the threshold below which the new alert would not join the scenario and would start a new scenario.	113
6.1	MIB II-based ID Architecture	128
6.2	MIB II-based Anomaly ID Model	130
6.3	Conditional Entropy of MIB II object *ipInReceives*	132
6.4	Misclassification rate of MIB II object *ipInReceives*	132
6.5	Accuracy over Cost of MIB II object *ipInReceives*	133
6.6	*icmpInEchos* under normal condition	137
6.7	*icmpInEchos* under Ping Flood attack	137
6.8	*udpInDatagrams* under normal condition	139
6.9	*udpInDatagrams* under UDP Flood attack	139
6.10	*udpInErrors* under normal condition	140
6.11	*udpInErrors* under UDP Flood attack	140
6.12	*ICMP_In* ID sub-module under normal condition	143
6.13	*ICMP_In* ID sub-module under Mix Flood attack	143
6.14	*ICMP_Out* ID sub-module under normal condition	144
6.15	*ICMP_Out* ID sub-module under Mix Flood attack	144
6.16	*UDP_In_Error* ID sub-module under normal condition	145
6.17	*UDP_In_Error* ID sub-module under Mix Flood attack	145
6.18	*TCP_In* ID sub-module under normal condition	146
6.19	*TCP_In* ID sub-module under Mix Flood attack	146
7.1	The AMG System Architecture	158
7.2	Visualization of Data in Database	168
7.3	Visualization of SQL Query	169
8.1	Authentication, Passive Intrusion Detection and Proactive Intrusion Detection. Alarm $A0$ is connected to an Authenticator. Alarm $A1$ enables Passive Intrusion Detection, while alarms $A2$ and $A3$ enable Proactive Intrusion Detection.	199
8.2	Timeline of the outputs of the alarms. It is assumed that the time elapsed between the entry of a malicious agent in the house and malicious activity is negligible.	202
8.3	Proactive Intrusion Detection - Extracting Temporal Rules.	205

8.4 The idealized inputs and output signals. p is the length of the window used for parameter estimation when applying GCT. If $p \geq r$, the representation (8.1) captures model (8.4) exactly, and the **Precursor** $\Rightarrow$ **Phenomenon** rule is "visible" through the model. When $H(q^{-1}) = 1$, the response in y collapses into a blip. 215
8.5 DDoS Attacks - A simplified Timeline. 218
8.6 TFN2K Ping Flood: Selected MIB variables at the Attacker and at the Target. 219

List of Tables

2.1	Sample database table.	37
4.1	Lincoln Labs Data Summary	95
4.2	Selected points from the ROC curves of the performance of each algorithm over the KDD Cup 1999 Data.	99
5.1	r value calculation for addresses 172.16.112.20 and 172.16.112.40	107
5.2	Confusion matrices from Naïve approach on test data set . Two hundred forty six of the 4,041 patterns that should have produced a "join" decision (6.09%) incorrectly produced a "don't join" decision, and ten of the 246,320 patterns that should have produced a "don't join" decision incorrectly produced a "join" decision.	116
5.3	Confusion matrices from heuristic approach on test data set. The algorithm tries to match the human decision. For example, 4,041 test examples should have produced a "join" decision. The algorithm correctly produced the "join" decision 3,589 times (88.81% of the time).	117
5.4	Confusion matrices from a decision tree on the test data set. The decision tree tries to match the human decision. For example, 4,041 examples should have produced a "join" decision. The decision tree produced the correct decision on 4,037 of these (99.90% of the time).	118
6.1	Misuse Detections by MIB II-based ID Model	141
6.2	Anomaly Detections by MIB II-based ID Model	148
8.1	Key Variables at the Attacker for TFN2K - Ground Truth.	221

8.2 TFN2K Ping Flood Run 1: Top MIBs at the Attacker according to the g statistic. 221

8.3 Results of Step 2: Detection Rates and FA Rates for MIB variables that contain precursors to DDoS Attacks. 222

8.4 Final Results: Detection Rates and FA Rates for Events at MIB variables for TFN2K Ping Flood. 222

9.1 Summary statistics of the e-mail newsgroup and author corpus used in the experiment. 241

9.2 E-mail document body style marker attributes. Total of 170 features are used in the experiment. See text for clarification. 242

9.3 E-mail document body structural attributes. Total of 21 attributes/features are used in the experiment. See text for clarification. 243

9.4 Per-author-category P_{AC_i}, R_{AC_i} and F_{1,AC_i} categorisation performance results (in %) for the four different author categories ($i = 1, \ldots, 4$). The newsgroup `aus.tv` is used as the training set (see text).[a] 245

Preface

Data mining is becoming a pervasive technology in activities as diverse as using historical data to predict the success of a marketing campaign, looking for patterns in financial transactions to discover illegal activities, or analyzing genome sequences. From this perspective, it was just a matter of time for the discipline to reach the important area of computer security. This book presents a collection of research efforts on the use of data mining in computer security.

Data mining has been loosely defined as the process of extracting information from large amounts of data. In the context of security, the information we are seeking is the knowledge of whether a security breach has been experienced, and, if the answer is yes, who is the perpetrator. This information could be collected in the context of discovering intrusions that aim to breach the privacy of services, or data in a computer system or, alternatively, in the context of discovering evidence left in a computer system as part of a criminal activity.

This collection concentrates heavily on the use of data mining in the area of intrusion detection. The reason for this is twofold. First, the volume of data dealing with both network and host activity is so large that it makes it an ideal candidate for using data mining techniques. Second, intrusion detection is an extremely critical activity. To understand this it is enough to look at the current statistics. Ten major government agencies accounting for 99% of the federal budget have been compromised in the recent past. In the year 2000, a massive, coordinated attack successfully brought down some of the major e-commerce web sites in the United States. Moreover, it is estimated that less than 4% of the attacks are actually detected or reported. As a society, we have become extremely dependent of the use of information systems, so much so that the danger of serious disruption of crucial operations is frightening. As a result, it is no surprise that researchers have produced a relatively large volume of work in the area of data mining in support of intrusion detection.

The rest of the work presented in this volume addresses the application of data mining to an equally pressing area: computer forensics. This area has widened recently to address activities such as law enforcement using digital evidence. Although the amount of work is not as large as in intrusion detection, computer forensics proves to be a fruitful arena for research in data mining techniques.

Data mining holds the promise of being an effective tool to help security activities and, in some sense, the proof of its applicability can be found in the pages of this book. However, there is still a long road to travel and we hope that this volume will inspire researchers and practitioners to undertake some steps in this direction.

Acknowledgments

We are extremely grateful to authors for their contributions and to Jia-Ling Lin who assisted with every aspect of this book, ranging from collecting of manuscripts to dealing with all matters related to Kluwer style files. It is also a pleasure to acknowledge Joe Giordano, Brian Spink, and Leonard Popyack of the Air Force Research Laboratory/Rome for their support of our research in the application of data mining to intrusion detection.

Daniel Barbará

Sushil Jajodia

Fairfax, VA

Chapter 1

MODERN INTRUSION DETECTION, DATA MINING, AND DEGREES OF ATTACK GUILT

Steven Noel
Center for Secure Information Systems
George Mason University, Fairfax VA 22030-4444, USA
snoel@gmu.edu

Duminda Wijesekera
Center for Secure Information Systems
George Mason University, Fairfax VA 22030-4444, USA
dwijesek@gmu.edu

Charles Youman
Center for Secure Information Systems
George Mason University, Fairfax VA 22030-4444, USA
charles.youman@acm.org

Abstract

This chapter examines the state of modern intrusion detection, with a particular emphasis on the emerging approach of data mining. The discussion parallels two important aspects of intrusion detection: general detection strategy (misuse detection versus anomaly detection) and data source (individual hosts versus network traffic). Misuse detection attempts to match known patterns of intrusion, while anomaly detection searches for deviations from normal behavior. Between the two approaches, only anomaly detection has the ability to detect unknown attacks. A particularly promising approach to anomaly detection combines association mining with other forms of machine learning such as classification. Moreover, the data source that an intrusion detection system employs significantly impacts the types of attacks it can detect. There is a tradeoff in the level of detailed information available ver-

sus data volume. We introduce a novel way of characterizing intrusion detection activities: degree of attack guilt. It is useful for qualifying the degree of confidence associated with detection events, providing a framework in which we analyze detection quality versus cost.

Keywords: Information security, Intrusion detection, data mining

1. Introduction

The goal of intrusion detection is to discover intrusions into a computer or network, by observing various network activities or attributes. Here intrusion refers to any set of actions that threatens the integrity, availability, or confidentiality of a network resource.

Given the explosive growth of the Internet and the increased availability of tools for attacking networks, intrusion detection becomes a critical component of network security. While such detection usually includes some form of manual analysis, we focus on software systems for automating the analysis.

One useful method of classification for intrusion detection systems is according to general strategy for detection. There are two categories under this classification: misuse detection and anomaly detection.

Misuse detection finds intrusions by looking for activity corresponding to known techniques for intrusion. This generally involves the monitoring of network traffic in search of direct matches to known patterns of attack (called signatures). This is essentially a rule-based approach. A disadvantage of this approach is that it can only detect intrusions that follow pre-defined patterns.

In anomaly detection, the system defines the expected behavior of the network (or profile) in advance. Any significant deviations from this expected behavior are then reported as possible attacks. Such deviations are not necessarily actual attacks. They may simply be new network behavior that needs to be added to the profile. The primary advantage of anomaly-based detection is the ability to detect novel attacks for which signatures have not been defined.

Another useful method of classification for intrusion detection systems is according to data source. The two general categories are host-based detection and network-based detection.

For host-based intrusion detection, the data source is collected from an individual host on the network. Host-based detection systems directly monitor the host data files and operating system processes that will potentially be targets of attack. They can, therefore, determine exactly which host resources are the targets of a particular attack.

For network-based intrusion detection, the data source is traffic across the network. This involves placing a set of traffic sensors within the network. The sensors typically perform local analysis and detection and report suspicious events to a central location. Since such monitors perform only the intrusion detection function, they are usually much easier to harden against attack and to hide from the attackers.

We propose another way of characterizing intrusion detection activities, through degree of attack guilt. That is, it is interesting to understand how well a system can correctly separate genuine attacks from normal activity in terms of attack guilt. This is not a classification of detection systems, but rather of network activities. Intrusion degree of guilt is useful for qualifying the degree of confidence associated with detection events, providing a framework for analyzing detection quality versus cost.

This chapter examines the state of modern intrusion detection. Section 2 discusses the state of the art with respect to generally strategy for detection. Section 3 then considers intrusion detection systems in terms of their data sources. In Section 4, we introduce degree of attack guilt as a way of characterizing intrusion detection activities, providing a framework in which we analyze detection quality versus cost. Section 5 has our concluding comments.

2. Detection Strategies

The current generation of commercial intrusion detection systems is largely network-based, and employs misuse detection. As such, current tools completely lack the ability to detect attacks that do not fit a pre-defined signature. Several other researchers have reached this conclusion (CTC-Corporation, 2000; Jackson, 1999; LaPadula, 1999; LaPadula, 2000; Allen et al., 2000; Axelsson, 1999; Axelsson, 2000b; Kvarnstrom, 1999).

Given the shortcomings of misuse detection in commercial systems, an important research focus is anomaly detection, rather than mere extensions of misuse detection. Research is also needed in systems that combine the two approaches. A critical issue for anomaly detection is the need to reduce false alarms, since any activity outside a known profile raises an alarm. Indeed, false alarm rate is the limiting factor in the performance of current intrusion detection systems (Axelsson, 2000a; Lundin and Jonsson, 1999).

Increased network speeds, switched networks, and the application of encryption have prompted a trend toward host-based detection. An-

other interesting new approach is distributed intrusion detection, in which host-based systems monitor a number of hosts on the network and transfer the monitored information to a central site.

Overall, intrusion detection technology is immature and rapidly evolving. In the commercial realm, new vendors appear frequently but are often absorbed by others. On the research front, a variety of approaches are being investigated. However, an overall theoretical framework is still lacking (Allen et al., 2000).

2.1 Misuse Detection

Misuse detection searches for known patterns of attack. This is the strategy employed by the current generation of commercial intrusion detection systems. A disadvantage of this strategy is that it can only detect intrusions that follow pre-defined patterns.

The major approaches that have been proposed for misuse detection are expert systems, signature analysis, state-transition analysis, and data mining. Approaches have also been proposed involving colored Petri nets and case-based reasoning.

Misuse detection searches for known patterns of attack. This is the strategy employed by the current generation of commercial intrusion detection systems. A disadvantage of this strategy is that it can only detect intrusions that follow pre-defined patterns.

The major approaches that have been proposed for misuse detection are expert systems, signature analysis, state-transition analysis, and data mining. Approaches have also been proposed involving colored Petri nets and case-based reasoning.

2.1.1 Expert Systems. The expert system approach to misuse detection uses a set of rules to describe attacks. Audit events are translated into facts carrying their semantic significance in the expert system. An inference engine then draws conclusions using these rules and facts.

Examples of misuse detection systems using expert systems are IDES (Intrusion Detection Expert System) (Denning, 1987; Lunt, 1989; Lunt et al., 1992; Javitz and Valdes, 1991), ComputerWatch (Dowell and Ramstedt, 1990), NIDX (Network Intrusion Detection Expert System) (Bauer and Koblentz, 1988), P-BEST (Production- Based Expert System Toolset) (Lindqvist and Porras, 1999), and ISOA (Information Security Officer's Assistant) (Winkler and Landry, 1992; Winkler, 1990).

IDES (developed at SRI) uses an expert system that encodes known intrusion scenarios, known system vulnerabilities, and site-specific security policies. It addresses external attacks from unauthorized users,

authorized users who masquerade as other users, and authorized users who abuse their privileges by evading access controls.

ComputerWatch (developed at AT&T) takes an expert system approach to summarize security sensitive events and apply rules to detect anomalous behavior. It checks users' actions according to a set of rules that describe proper usage policy, and flags any action that does not fit the acceptable patterns.

NIDX (developed at Bell Communication Research) is a knowledge-based prototype intrusion detection expert system for Unix System V. It combines knowledge of the target system, history profiles of users' past activities, and intrusion detection heuristics. The result is a knowledge-based system capable of detecting specific violations that occur on the target system. A unique feature of NIDX is that it includes facts describing the target system and heuristics embodied in rules that detect particular violations from the target system audit trail. NIDX is thus operating system dependent.

P-BEST (developed at SRI) is a rule-based, forward-chaining expert system that has been applied to signature-based intrusion detection for many years. The main idea is to specify the characteristics of a malicious behavior and then monitor the stream of events generated by system activity, hoping to recognize an intrusion signature.

P-BEST is a general-purpose programmable expert system shell, sporting a rule definition language that is simple enough to be used by non-experts. The system was first deployed in the MIDAS ID system at the National Computer Security Center. Later, P-BEST was chosen as the rule-based inference engine of NIDES, a successor to the IDES prototype. The P-BEST expert system shell is also used in EMERALD's expert, a generic signature-analysis engine (Porras and Neumann, 1997).

ISOA (developed at Planning Research Corporation) is a real time security monitor that supports automated as well as interactive audit trail analysis. It contains a statistical analysis module and an expert system. For the events not constituting direct violations of security policy, their expected behavior is compared against profiles that specify thresholds and the reliability factor for the events. Deviations are identified by statistical checks of expected versus actual behavior. For events that cannot be monitored by examining the thresholds, the expert system component can specify the possible relationships and implied meaning of the events.

2.1.2 Signature Analysis. Signature analysis transforms the semantic description of attacks into information that can be found in the audit trail in a straightforward way. Examples of such information

include the sequences of audit events that attacks generate, or patterns of data that can be sought in the audit trail.

Systems that use signature analysis include Haystack (Smaha, 1988), NetRanger (Net-Ranger, 1999), RealSecure (Real-Secure, 1999), and MuSig (Misuse Signatures) (Lin et al., 1998).

Haystack is a misuse detection system that helps Air Force security officers detect misuse of Unisys mainframes. Working on the reduced audit trails data, it performs misuse detection based on behavioral constraints imposed by official security policies and on models of typical user behavior.

NetRanger (developed at Cisco) is composed of two modules: sensors and directors. Sensors are network security monitors that analyze the network traffic on a network segment and the logging information produced by Cisco routers to detect network-based attacks. Directors are responsible for the management of a group of sensors and can be structured hierarchically to manage large networks.

RealSecure (developed at Internet Security Systems) is composed of three modules: network engines, system agents, and managers. The network engines are network monitors equipped with attack signatures that are matched against the traffic on a network link. The system agents are host-based intrusion detection systems that monitor security sensitive log files on a host. These modules report their finds to the central manager, which displays the information to the user and provides functionalities for remote administration system agents and network engines.

MuSig (developed at George Mason University's Center for Secure Information Systems) applies a high-level language for abstract signatures. It attempts to overcome certain limitations of traditional misuse detection systems, including the limited expressiveness of signatures expressed in low-level language, and fixed monitoring algorithms for misuse that have difficulty adapting to a changing operating environment or security objectives. Through its high-level language, MuSig can represent misuses in a simple form with high expressiveness.

2.1.3 State-Transition Analysis. State-transition analysis describes attacks with a set of goals and transitions based on state-transition diagrams. Any event that triggers an attack state will be considered an intrusion. Examples of systems applying state transition analysis are USTAT (Unix State Transition Analysis Tool) (Porras and Kemmerer, 1992; Ilgun, 1992) and NetSTAT (Network-based State Transition Analysis Tool) (Vigna and Kemmerer, 1998).

USTAT (developed at UC Santa Barbara) is a real-time intrusion detection system for Unix. The original design was STAT (State Transition

Analysis Tool) (Porras, 1992). STAT employs rule-based analysis of the audit trails of multi-user computer systems. In STAT, an intrusion is identified as a sequence of state changes that lead the computer system from some initial state to a target compromised state. USTAT makes use of the audit trails that are collected by the C2 Basic Security Module of SunOS. It keeps track of only those critical actions that must occur for the successful completion of the penetration. This approach differs from other rule-based penetration identification tools that pattern match sequences of audit records.

NetStat (developed at UCSB) performs real-time network-based intrusion detection by extending the state transition analysis technique (first introduced in STAT) to the networked environment. The system works on complex networks composed of several sub-networks. Using state transition diagrams to represent network attacks entails a number of advantages, including the ability to automatically determine the data to be collected to support intrusion analysis. This enables a lightweight and scalable implementation of the network probes.

2.1.4 Data Mining. Data mining refers to a process of non-trivial extraction of implicit, previously unknown, and potentially useful information from databases. Example misuse detection systems that use data mining include JAM (Java Agents for Meta-learning) (W. Lee and Mok, 1998; Lee and Stolfo, 1998; Lee et al., 1999; Lee et al., 2000; Lee, 1999) MADAM ID (Mining Audit Data for Automated Models for Intrusion Detection) (Lee et al., 2000), and Automated Discovery of Concise Predictive Rules for Intrusion Detection (Lee, 1999).

JAM (developed at Columbia University) uses data mining techniques to discover patterns of intrusions. It then applies a meta-learning classifier to learn the signature of attacks. The association rules algorithm determines relationships between fields in the audit trail records, and the frequent episodes algorithm models sequential patterns of audit events. Features are then extracted from both algorithms and used to compute models of intrusion behavior. The classifiers build the signature of attacks. So essentially, data mining in JAM builds a misuse detection model.

JAM generates classifiers using a rule learning program on training data of system usage. After training, resulting classification rules is used to recognize anomalies and detect known intrusions. The system has been tested with data from Sendmail-based attacks, and with network attacks using TCP dump data. MADAM ID uses data mining to develop rules for misuse detection. The motivation is that current systems require extensive manual effort to develop rules for misuse detection.

MADAM ID applies data mining to audit data to compute models that accurately capture behavioral patterns of intrusions and normal activities. MADAM ID performed well in the 1998 DARPA evaluation of intrusion detection systems in detecting known attacks (Lippmann et al., 2000).

Researchers at Iowa State University report on Automated Discovery of Concise Predictive Rules for Intrusion Detection (Helmer et al., 1999). This system performs data mining to provide global, temporal views of intrusions on a distributed system. The rules detect intrusions against privileged programs (such as Sendmail) using feature vectors to describe the system calls executed by each process. A genetic algorithm selects feature subsets to reduce the number of observed features while maintaining or improving learning accuracy. This is another example of data mining being used to develop rules for misuse detection.

2.1.5 Other Approaches. The colored Petri nets approach is a graphical language for design, specification, simulation and verification of systems. It is particularly well-suited for systems in which communication, synchronization and resource sharing are important (Jensen, 1997).

The only known example of an intrusion detection system that uses colored Petri nets is IDIOT (Intrusion Detection in Our Time) (Crosbie et al., 1996), developed at Purdue University. In particular, IDIOT applies colored Petri nets to represent attack signatures. Advantages of colored Petri nets include their generality, their conceptual simplicity, and their ability to be represented as graphs. However, matching a complex signature against the audit trail can become computationally expensive.

Another approach to misuse detection involves case-based reasoning. Case-based reasoning is a problem solving methodology in which previous problem solving situations are used in solving new problems. It is most suitable when it is difficult or impossible to break down the knowledge into a set of rules, and only records of prior cases exist.

The only known application of case-based reasoning to intrusion detection is AUTOGUARD (Esmaili et al., 1996; Esmaili et al., 1997), although it is not clear if the system has been fully implemented.

2.2 Anomaly Detection

A significant disadvantage of misuse detection is the inability to detect attacks for which signatures have not been defined. A detection strategy that addresses this shortcoming is anomaly detection.

In anomaly detection, the system defines the expected network behavior (known as the profile) in advance. Any significant deviations from the profile are then reported as possible attacks. Such deviations are not necessarily actual attacks. They may simply be new network behavior that needs to be added to the profile. Anomaly detection systems have emerged that have very promising performance against novel attacks.

The major approaches to anomaly detection include statistical methods, expert systems, and data mining. Approaches have also been proposed involving neural networks and computer immunology.

2.2.1 Statistical Methods. Statistical methods measure the user and system behavior by a number of variables sampled over time, and build profiles based on the variables of normal behavior. The actual variables are then compared against the profiles, and deviations are considered abnormal.

Example systems employing statistical methods for anomaly detection are IDES (Intrusion Detection Expert System) (Denning, 1987; Lunt, 1989; Javitz and Valdes, 1991; Lunt et al., 1992) NIDES (Next- Generation Intrusion Detection Expert System) (Anderson et al., 1995a; Anderson et al., 1995b), and Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) (Porras and Neumann, 1997; Neumann and Porras, 1999).

Section 2.1.1 describes the IDES system in the context of misuse detection. IDES also employs statistical anomaly detection. In particular, it uses audit data to characterize user activity and detect deviations from normal user behavior. Information extracted from audit data includes user login, logout, program execution, directory modification, file access, system calls, session location change, and network activity. The NIDES system extends IDES by integrating its response logic with the results produced by the anomaly detection subsystem.

EMERALD (developed at SRI) aims to detect intrusions in large networks, and focuses on the scalability of the system. It is a hybrid of misuse detection and anomaly detection that contains an expert system P-BEST and a statistical anomaly detector. It allows hierarchical composition of decentralized service monitors that apply the statistical analysis to network data.

In other work, Cabrera et al.(Cabrera et al., 2000) examine the application of statistical traffic modeling for detecting novel attacks against networks. They show that network activity models efficiently detect denial of service and probe attacks by monitoring the network traffic volume. For application models, they use the Kolmogorov-Smirnov test to demonstrate that attacks using telnet connections in the DARPA dataset (Lippmann et al., 2000) are statistically different from normal telnet connections.

2.2.2 Expert Systems. For anomaly detection, expert systems describe users' normal behavior by a set of rules. Examples of expert systems applied to anomaly detection include ComputerWatch (Dowell and Ramstedt, 1990) and Wisdom & Sense (Liepins and Vaccaro, 1992; Vaccaro and Liepins, 1989; Liepins and Vaccaro, 1989).

ComputerWatch (developed at AT&T) uses an expert system approach to summarize security sensitive events and apply rules to detect anomalous behavior. It checks users' actions according to a set of rules that describe proper usage policy, and flags any action that does not fit the acceptable patterns.

Wisdom & Sense (developed at the Los Alamos National Lab) detects statistical anomalies in users' behavior. It first builds a set of rules that statistically describe behavior based on recordings of user activities over a given period of time. Subsequent activity is then compared against these rules to detect inconsistent behavior. The rule base is rebuilt regularly to accommodate new usage patterns.

Terran Lane of Purdue University studied machine learning techniques for anomaly detection (Lane, 2000). The project profiles Unix user command line data. It shows that anomaly detection is effective at user differentiation under some conditions, but that alone it is insufficient for high-confidence defense. This approach assumes the false alarms generated by an intrusion detection sensor will be filtered out by a higher-level decision maker. The results are also tentative because they do not include any data known to be hostile.

2.2.3 Data Mining. Data mining attempts to extract implicit, previously unknown, and potentially useful information from data. Applications of data mining to anomaly detection include ADAM (Audit Data Analysis and Mining) (Wu, 2001a; Barbara et al., 2001; Barbara et al., 1999), IDDM (Intrusion Detection using Data Mining) (Abraham, 2001), and eBayes (Valdes and Skinner, 2000).

ADAM (developed at George Mason University Center for Secure Information Systems) uses a combination of association rules mining and classification to discover attacks in TCP dump data. Section 2.1.4 discusses the JAM system, which also combines association mining and classification. But there are two significant differences between ADAM and JAM. First, ADAM builds a repository of *normal* frequent itemsets that hold during attack-free periods. It does so by mining data that is known to be free of attacks. Then, ADAM runs a sliding-window algorithm that finds frequent itemsets in the most recent set of TCP connections, and compares them with those stored in the normal itemset repository, discarding those that are deemed normal. With the rest, ADAM uses a classifier that has been previously trained to classify the suspicious connections as a known type of attack, an unknown type, or a false alarm. The system performs especially well with denial of service and probe attacks.

The ADAM system is able to detect network intrusions in real time with a very low false alarm rate. One of the most significant advantages of ADAM is the ability to detect novel attacks, without depending on attack training data, through a novel application of the pseudo-Bayes estimator (Barbara et al., 2001). In the 1999 DARPA Intrusion Detection Evaluation (Lippmann et al., 2000), ADAM ranked 3rd overall. Among the top 3, ADAM is the only system employing anomaly detection (Wu, 2001a). Note also that the DARPA evaluation has no criterion that singles out performance on new attacks. In short, there is no known system that is more effective at detecting unknown attacks than ADAM.

Abraham of the Defense Science and Technology Organization in Australia recently reported on IDDM. The system characterizes change between network data descriptions at different times, and produces alarms when detecting large deviations between descriptions. However, IDDM has problems achieving real-time operation. In particular, results are produced only after sufficient amounts of data are collected and analyzed.

The eBayes system is a newly developed component for the statistical anomaly detector of EMERALD. Defining a session as temporally contiguous bursts of TCP/IP traffic from a given IP, it applies Bayesian inference on observed and derived variables of the session, to obtain a belief for the session over the states of hypotheses. Hypotheses can be either normal events or attacks. Given a naive Bayes model, training data, and a set of hypotheses, a conditional probability table is built for the hypotheses and variables, and is adjusted for the current observations. By adding a dummy state of hypothesis and a new conditional probability table row initialized by a uniform distribution, eBayes can

dynamically generate the new hypothesis that helps it detect new attacks. But eBayes may be computationally expensive as the number of hypothesis states increases.

Kohavi et al.(Kohavi et al., 1997) study different approaches for handling unknowns and zero counts when estimating probabilities for naive Bayes classifiers, and propose a new variant of the Laplace estimator that shows better performance. The method works well if some cells of a row contain zeros. However, if a row is composed of all zeros, each cell of the row will have same conditional probability.

Data mining techniques have also been explored to detect new malicious executables (Schultz et al., 2001).

2.2.4 Other Approaches. Neural networks are used to learn users' normal behavior and predict the expected behavior of users. Ghosh and Schwartzbard (Ghosh and Schwartzbard, 1999) propose applying a neural network to learn a profile of normality.

Somayaji, Hofmeyr, and Forrest of the University of New Mexico have proposed a method of detecting intrusions that is based on the human immune system (Forrest et al., 1996; Somayaji et al., 1997). The technique first collects a set of reference audits representing the appropriate behavior of the service, and extracts a reference table containing all the known *good* sequences of system calls. These patterns are then used for live monitoring to check whether the sequences generated are listed in the table or not. If they do not, an alarm is generated. Although the immune system approach is interesting and intuitively appealing, so far it has proven to be difficult to apply (Engelhardt, 1997).

Wespi et al. (Wespi et al., 2000) propose a technique to build a table of variable length patterns based on Teiresias algorithm. They claim that, compared to a fixed length approach, the variable length pattern model uses fewer patterns to describe the normal process behavior and achieves better detection. Some research has exploited new techniques to model system and users' normal behavior. Lee et al. (Lee and Xiang, 2001) propose several information theoretic measures to describe the characteristics of an audit data set, and suggest the appropriate anomaly detection models. Each proposed measure could describe different regularities in the dataset.

Wagner et al. (Wagner and Dean, 2001) propose static analysis to automatically derive a model of application behavior. Assuming the system call traces of a program's execution are consistent with the program's source code, the approach first computes a model of expected application behavior, built statically from program source code. At run time, it then monitors the program and checks its system call trace for

compliance to the model. Since the proposed models need to include every possible path of system call trace of a program's normal execution, the approach may be not feasible. The run time overhead is high.

3. Data Sources

A useful classification for intrusion detection systems is according to their data source. To a large extent, the data source determines the types of intrusions that can be detected. The two general categories are host-based detection and network-based detection.

For host-based systems, the data source is collected from an individual host on the network. In particular, these systems employ their host's operating system audit trail as the main source of input. Because host-based systems directly monitor the host data files and operating system processes, they can determine exactly which host resources are the targets of a particular attack.

Given the rapid development of computer networks, some traditional single-host intrusion detection systems have been modified to monitor a number of hosts on a network. They transfer the monitored information from multiple monitored hosts to a central site for processing. These are termed distributed intrusion detection systems. Example distributed systems are IDES (Denning, 1987; Lunt, 1989; Lunt et al., 1992), NSTAT (Kemmerer, 1997), and AAFID (Spafford and Zamboni, 2000).

Network-based intrusion detection employs network traffic as the main source of input. This involves placing a set of traffic sensors within the network. The sensors typically perform local analysis and detection and report suspicious events to a central location. These sensors are generally easier to harden against attack and to hide from attackers, since they perform only the intrusion detection function.

Recent trends in computer networks, specifically, the increased use of encryption, increased speed of networks, and greater use of switched network segments have led to a greater emphasis on host-based intrusion detection (Allen et al., 2000). However, there are also an increasing number of attacks where the target of the attack is the network infrastructure, such as the domain name service (DNS). Since 1997, the CERT/CC at Carnegie Mellon University has published twelve documents describing vulnerabilities or exploitation of vulnerabilities in a common implementation of DNS (2001) (CERT Advisory, 2001). There are efforts to use network-based intrusion detection to detect attacks on the network infrastructure.

Examples of this trend are NSM (Network Security Monitor) (Heberlein et al., 1992), DIDS (Distributed Intrusion Detection System) (Snapp et al., 1991), the JiNao system (Wu et al., 1999), and NADIR (Network Anomaly Detection and Intrusion Reporter) (Hochberg et al., 1993).

Network-based intrusion detection systems have been widened to address large, complex network environments. Examples include GrIDS (Graph-based Intrusion Detection System) (Staniford-Chen et al., 1996), EMERALD (Porras and Neumann, 1997), NetStat (Vigna and Kemmerer, 1998), CARDS (Coordinated Attack Response and Detection System) developed at the Center for Secure Information Systems at George Mason University (Yang et al., 2000), NetRanger (Net-Ranger, 1999), and RealSecure (Real-Secure, 1999).

As an internal research and development effort, the MITRE Corporation has studied the use of data mining on intrusion detection alarms to reduce the false alarm rate (Staniford-Chen et al., 1996; Clifton and Gengo, 2000). These reports include some interesting statistics on the level of alarms generated by sensors at MITRE (over 1,000,000 alarms per week).

IBM's emergency response service provides real-time intrusion detection (RTID) services through the Internet for a variety of clients. The emergency response service needs to analyze and respond to thousands of alerts per day. Data mining techniques were used to analyze a database of RTID alerts. They developed profiles of normal alerts and of their clients. Several different types of clients were discovered, each with different alert behaviors and thus different monitoring needs (Manganaris et al., 2000).

4. Degrees of Attack Guilt

In this section, we introduce an interesting way of characterizing intrusion detection activities: degree of attack guilt. This characteristic focuses on how well an intrusion detection system can correctly separate genuine attacks from normal activity. Unlike detection strategy and data source, this characteristic does not apply to the classification of detection systems themselves. Rather, it applies to the classification of network activities, though still with respect to intrusion detection.

In essence, degree of attack guilt is a generalization of detection accuracy. That is, detection accuracy considers whether positive or negative detections are true or false. Degrees of attack guilt consider a continuous spectrum, rather than simply positive or negative detections. This captures something about the degree of confidence of detections, and provides a framework for discussing the costs of improving confidence.

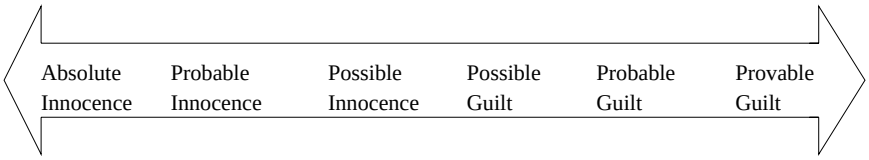


Figure 1.1. General Degrees of Attack Guilt

Degree of attack guilt spans the spectrum from absolute innocence to provable guilt, as shown in Figure 1.1. Provable guilt means that there is no question that the behavior is malicious or unauthorized. Absolute innocence refers to normal, authorized behavior that shows no sign of attack guilt. Actually, absolute innocence is impossible to prove. For example, a user may be involved in activity that is, strictly speaking, authorized and non-malicious. But that same behavior may be part of some subsequent malicious activity.

There is really a continuous spectrum of guilt between the two extremes of absolute innocence and provable guilt. But for general discussion, it is convenient to define a discreet set of degrees of known guilt. The complete lack of knowledge of guilt falls in the center of the scale, though we do not include it in Figure 1.1. Moving further from the center of the scale corresponds to increased knowledge of the nature of the behavior, towards either guilt or innocence. We introduce the categories *possible* and *probable* corresponding to increasing levels of known guilt or innocence.

Because degree of attack guilt concerns the confidence associated with detection events, it provides a framework for analyzing cost versus confidence. For example, there is additional cost associated with moving an activity from possible to provable guilt. In intrusion detection, it is often important to understand these types of quality-to-cost tradeoffs.

Moreover, different levels of monitoring may be warranted for particular desired degrees of guilt. That is, different operating environments may have varying needs of detection. For example, highly secure systems may want to investigate all activity that cannot be considered absolutely innocent. Other environments may allow activities to proceed as long as they are probably innocent.

The remainder of this section applies the idea of degree of attack guilt to the various forms of intrusion detection. Just as for Section 2, the discussion is organized by the general detection strategy, either misuse detection or anomaly detection.

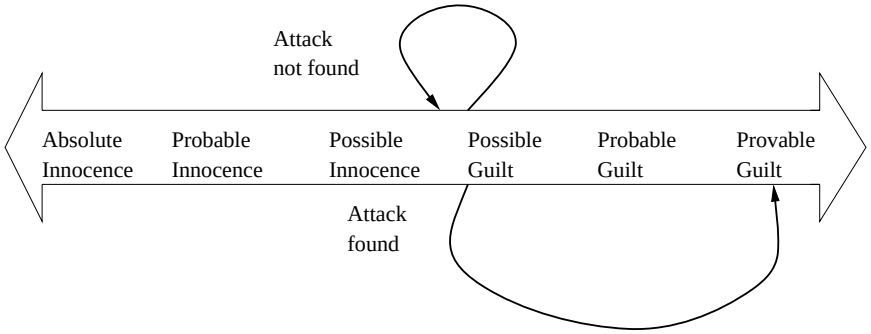


Figure 1.2. Degrees of Attack Guilt for Knowledge-based Misuse Detection

4.1 Misuse Detection

This section discusses degrees of attack guilt for misuse detection. Section 2.1 reviews specific systems that perform misuse detection, and is organized by the predominant approach that a system takes. Here the discussion is more general, though the organization is still based on the detection approach. In particular, the various approaches in Section 2.1 are coalesced to two general types: knowledge-based methods and machine-learning methods.

4.1.1 Knowledge-Based Methods. In knowledge-based methods for misuse detection, network or host events are checked against predefined rules or patterns of attack. The goal is to employ representations of known attacks that are general enough to handle actual occurrences of the attacks. Examples of knowledge-based methods are expert systems, signature analysis, and state-transition analysis.

Figure 1.2 shows degrees of attack guilt for knowledge-based misuse detection. These approaches search for instances of known attacks, by attempting to match with pre-determined attack representations. The search begins as all intrusion detection approaches begin, with a complete lack of knowledge of any attacker guilt This corresponds to the center of the attack-guilt scale.

But when a particular activity is matched with a known attack, the system gains the knowledge of the attack. For knowledge-based misuse detection, the activity matches a rule or pattern of known guilt. This gained knowledge corresponds to moving to the right of the known-guilt scale. To the extent that the system has a correct attack representation and matching scheme, this method would be expected to provide *provable guilt.*

Such provable guilt is expensive, because it is very time consuming to design general representations for detecting possible variations of specific attacks. Also, a different representation must be created for each type of attack, and there are many possible types of attacks.

For activities that are not matched to a known attack, no real guilt information is provided. All that has been proven is that such activities do not match known attacks. In particular, new attacks or even sufficiently different variations of known attacks are missed. This complete lack of attack knowledge corresponds to remaining in the center of the attack-guilt scale, with no additional cost incurred.

4.1.2 Machine-Learning Methods. In machine-learning methods for misuse detection, patterns and classes of attacks are discovered rather than being pre-defined. Such methods exploit any regularities or strong associations inherent in data such as network traffic. The goal is still to create general representations of attacks, just as for knowledge- based methods. The difference is that the representations are automatically induced, avoiding the costly design of representations in knowledge-based approaches.

Examples of machine-learning methods are data mining and classification. In facts, these two methods are usually combined in intrusion detection systems. Data mining discovers strong associations among data elements, which often correspond to attack patterns. Classifiers then induce classes of attacks, based on data mining results and other attributes from training data with known attacks.

Figure 1.3 shows degrees of attack guilt for machine-learning misuse detection. These approaches attempt to classify according to known attacks, by comparing them with previously learned attack representations. Before classification, there is a complete lack of knowledge of any attacker guilt, corresponding to the center of the attack-guilt scale.

During classification, a particular activity may be classified as a known attack. Just as for knowledge-based misuse detection, this corresponds to moving to the right of the known-guilt scale. But because the classification process is inherently a statistical summary that is prone to error, machine-learning methods can move us only as far as *probable guilt*.

Here we see the tradeoff in quality (degree of known guilt) versus cost. It is less expensive to automatically induce attack representations, but the resulting classifications are less reliable (not as provably guilt) compared to handcrafted representations.

In machine-learning approaches to misuse detection, classifiers are only trained for known attacks, not for normal events. That is, the

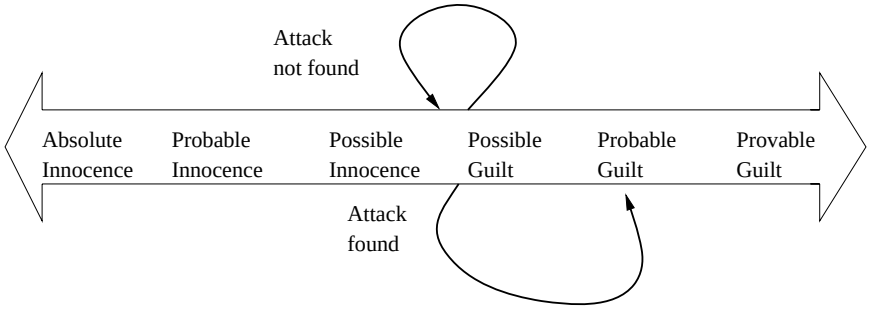


Figure 1.3. Degrees of Attack Guilt for Machine-learning Misuse Detection

intrusion detection system has attack signatures that have been learned, but no model for normal activity. As such, activity not classified as a known attack really corresponds to remaining in the center of the attack guilt scale (lack of guilt knowledge). In particular, misuse detection based on machine-learning methods has no ability to detect unknown attacks, and will misclassify them as normal behavior.

4.2 Anomaly Detection

This section discusses degrees of attack guilt for anomaly detection. Section 2.2 reviews specific systems that perform anomaly detection, and is organized by a system's predominant approach. In this section, the discussion is more general, though it still organizes the material based on the approach to detection. In particular, the various approaches in Section 2.2 are coalesced to three general types: knowledge-based methods, statistical methods, and machine- learning methods.

4.2.1 Knowledge-Based Methods. In knowledge-based methods for anomaly detection, network or host activity is checked against pre-defined rules or patterns of normal behavior. The goal is to employ a representation of normal behavior (a profile), from which anomalous behavior is identified as possible attacks. An example knowledge- based method is the expert systems approach.

Figure 1.4 shows degrees of attack guilt for knowledge-based anomaly detection. These approaches search for instances anomalous behavior, by comparing activities to a pre-determined profile. The search begins with a complete lack of knowledge of any attacker guilt, corresponding to the center of the attack-guilt scale.

When a particular activity is found in the profile, the system gains the knowledge that the activity is not an attack, corresponding to moving to

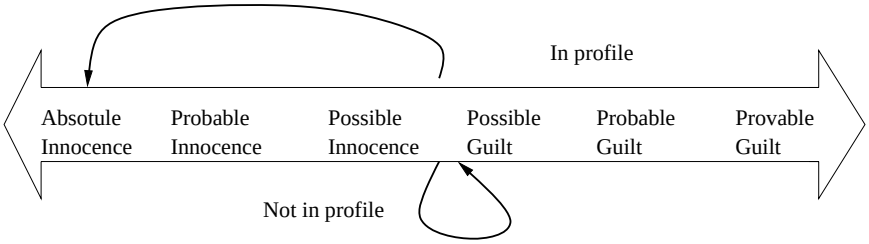


Figure 1.4. Degrees of Attack Guilt for Knowledge-based Anomaly Detection

the left of the known-guilt scale. Assuming that that the system has a correct representation in the profile and a correct matching scheme, this method would be expected to prove *absolute innocence*, at least with respect to current activities.

Obtaining such knowledge of absolute innocence is expensive, because it is involves the time consuming crafting of rules to represent normal behavior. This is particularly true when a comprehensive profile of normal behavior is needed that contains many rules.

For activities that are not found in the profile, no real guilt information is provided. All that has been proven is that such activities match known normal behavior. In particular, activities not in the profile may simply represent normal behavior that needs to be added to the profile. This lack of knowledge corresponds to remaining in the center of the attack-guilt scale, with no additional cost incurred.

4.2.2 Statistical Methods. In statistical methods for anomaly detection, profiles of normal behavior are generated by statistical criteria rather than being handcrafted. The goal is still to create representations of normal behavior, just as for knowledge-based methods of anomaly detection. The difference is that the representations are generated automatically as parameters of pre-determined statistical models. This automatic generation avoids the more costly handcrafting of profiles found in knowledge- based approaches.

Figure 1.5 shows degrees of attack guilt for statistical anomaly detection. This class of approaches attempts to statistically model normal behavior, and compare new behavior to the statistical profile. Before the comparison, there is a complete lack of knowledge of any attacker guilt, corresponding to the center of the attack-guilt scale.

When new behavior is compared to the statistical profile, a particular activity may be associated with normal behavior. Just as for knowledge-

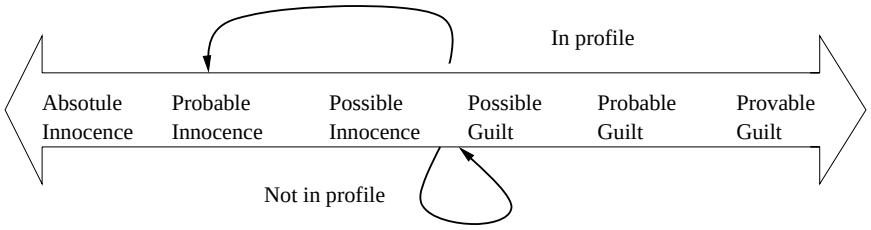


Figure 1.5. Degrees of Attack Guilt for Statistical Anomaly Detection

based anomaly detection, this corresponds to moving to the right of the known-guilt scale. But because the profile model is statistical, it is prone to error. Thus statistical methods of anomaly detection can move us only as far as *probable innocence*.

Again we see the tradeoff in quality versus cost. It is less expensive to automatically generate statistical profiles, but the resulting models are less reliable compared to profiles with handcrafted rules.

In statistical approaches to anomaly detection, statistical models are only built for normal behavior, not actual attacks. Thus activity not found in the profile really corresponds to remaining in the center of the attack guilt scale, i.e. lack of guilt knowledge. That is, statistical anomaly detection can identify normal behavior not in the profile as a possible attack, leading to higher false-alarm rates.

4.2.3 Machine-Learning Methods. A more recent direction in intrusion detection is the application of machine-learning methods to anomaly detection. This approach is particularly promising for the detection of novel attacks. Example machine-learning methods for anomaly detection include data mining and classification. We consider this approach in more depth with respect to degree of attack guilt.

In machine-learning methods for anomaly detection, profiles of normal behavior are automatically discovered. This automatic discovery avoids the costly handcrafting of profiles (via rules) found in knowledge-based approaches. It also avoids the labor-intensive process of carefully designing the applicable models needed for statistical approaches.

An example machine-learning approach to anomaly detection is the application of association mining to network audit data. This discovers rules that capture the predominant associations among traffic senders and receivers. Assuming that the mining is done initially during known attack-free times, the discovered rules then form the profile against which subsequently mined rules are compared.

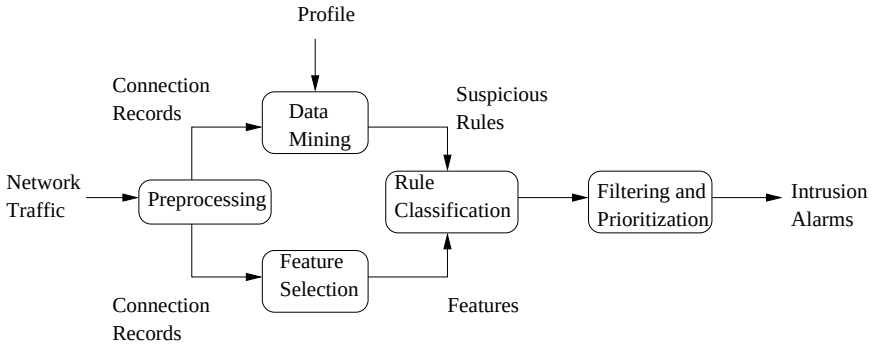


Figure 1.6. Combining data mining and classification for anomaly detection

A current limitation of this approach is that it has difficulty detecting very stealthy attacks. In other words, it detects attacks that involve a relatively large number of events within a period of time. This is because it sends an alarm only when the number of occurrences of an unexpected rule exceeds a threshold. This limitation is not unique to this approach; most of the anomaly detection models have the same problem (Ning, 2001).

A primary focus of research in anomaly detection is in detecting novel attacks while maintaining sufficiently low numbers of false alarms. This is particularly challenging because anomaly detection in general is prone to higher false-alarm rates. It is hard to define abnormal deviations as attacks if they cannot predictably be distinguished from variations of normal behavior.

An approach to this problem is to employ classifiers that are trained to learn the difference between normal and abnormal deviations from user profiles. These classifiers sift true intrusions from normal deviations, greatly reducing false alarms.

Figure 1.6 shows an architecture combining data mining and classification for anomaly detection. An initial training phase builds the profile of normal user behavior, by mining rules from network traffic known to be attack free. Also during training, rule classifiers learn attack classes through network traffic that has been labeled with known attacks. In the case of TCP/IP traffic, a preprocessing module extracts information from packet headers and builds records of TCP connections, which are subsequently mined and classified.

In the detection phase, a dynamic on-line mining algorithm produces suspicious rules. Suspicious rules are those that exceed a particular

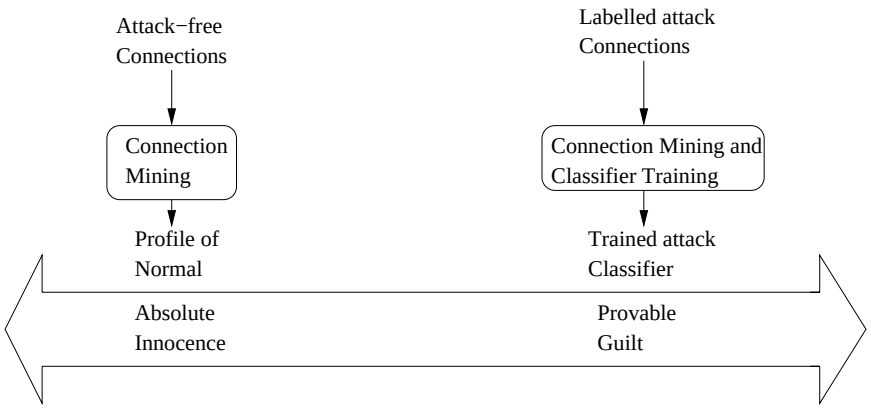


Figure 1.7. Degrees of Guilt for the Training Phase

support level and are missing from the profile of normal behavior. To achieve real-time performance, an incremental mining method can find suspicious rules within a sliding time window, so that datasets need only be scanned once (Wu, 2001b).

In principle, anomaly detection has the ability to detect novel attacks. But in practice this is far from easy, since detecting attacks for which no knowledge is available may seem an ill-posed problem. But attacks can be seen as violations of normal activity whose behavior can be estimated from the existing information.

A pseudo-Bayesian approach can enhance the ability of classifiers to detect new attacks while keeping the false-alarm rate low (Barbara et al., 2001). Pseudo-Bayes estimators estimate the prior and posterior probabilities of new attacks. A naive Bayes classifier can then classify instances as normal, known attacks, or new attacks. One advantage of pseudo-Bayes estimators is that no knowledge about new attacks is needed, since the estimated probabilities of new attacks are derived from the information of normal instances and known attacks.

In the training phase, the degrees of guilt are perfectly known, as illustrated in Figure 1.7 Network connections known to be guilt free are mined, and the discovered connection rules form the profile of normal behavior. Network connections with known attack labels are then mined for connection rules, which are used to train the attack classifiers.

At the beginning of the detection phase, the degrees of guilt for monitored network connections are completely unknown. This corresponds to the center of the known-guilt scale. The connection association mining process then discards mined connection rules that are already in the

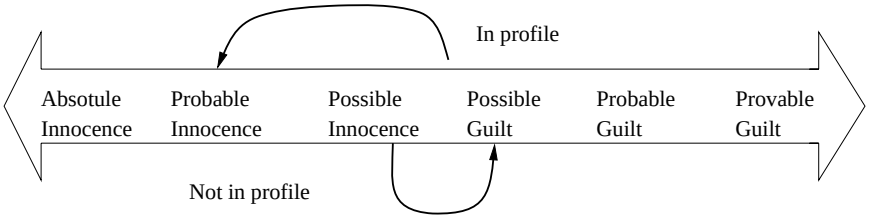


Figure 1.8. Degrees of Guilt for Connection Mining during Detection

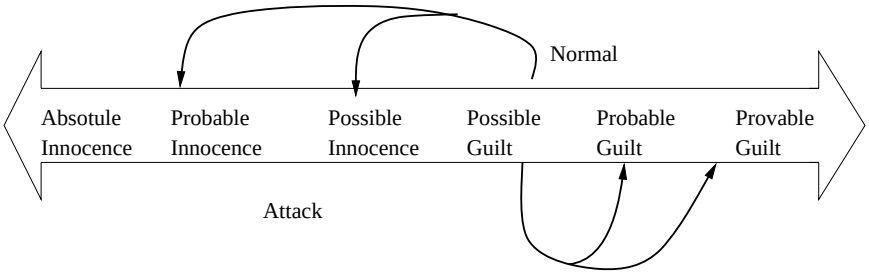


Figure 1.9. Degrees of Guilt for Attack Classification during Detection

profile, and retains those not in the profile for further classification. This corresponds to moving from completely unknown guilt to innocence beyond suspicion for rules already in the profile, or to possible guilt for rules not in the profile. This process is illustrated in Figure 1.8.

Next, mined connection rules not found in the profile are submitted to the rule classification engine. This corresponds to moving from possible guilt to either probable or possible innocence (for rules in the profile), or to probable or possible guilt (for rules not in the profile). This is shown in Figure 1.9.

The potential ambiguity in degrees of guilt in Figure 1.9 arises because of the nature of the attack classifiers. The classifiers undergo supervised training, in which class decision surfaces are induced that minimize some statistical measure of misclassification based on training data. Because of the statistical nature of the classifiers and their dependence on the quality of training data, a question of classification confidence arises.

Figure 1.10 illustrates possible confidence scenarios for the attack classifiers. For connection rules relatively far from the decision surface, the probability of misclassification is small. For these rules, we could thus say that either innocence or guilt (respectively) is very probable. But

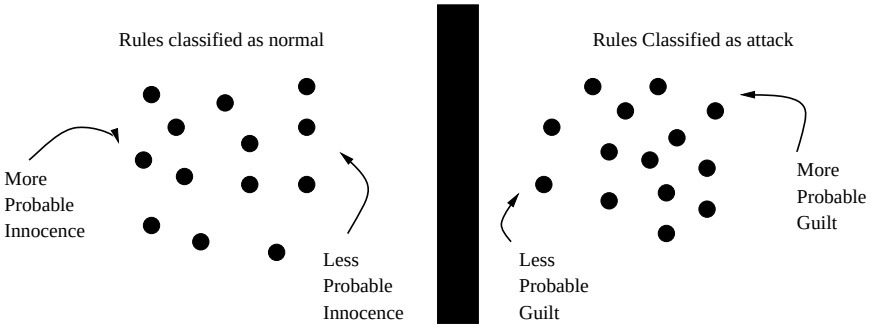


Figure 1.10. Degrees of Guilt with respect to Attack Classification Confidence

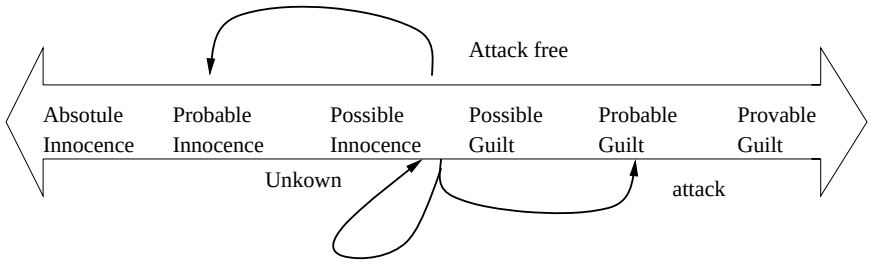


Figure 1.11. Degrees of Guilt for Overall Machine-learning Approach

connection rules relatively close to the decision surface have a higher chance of being misclassified. Innocence or guilt is then less probable.

One possible classifier is the naive Bayes classifier with pseudo-Bayes estimators. This classifier includes a class for unknown attacks, which is missing from the supervised classifiers. This class handles connection rules that cannot be classified either normal or known attacks. The Bayes classifier avoids the misclassification of unknown attacks (as either normal or known attacks) that plague supervised classifiers.

Connection rules classified as unknown remain in the center of the degree-of-guilt scale. That is, such a rule is known to belong to neither the normal class nor a known attack class. At this point we have no knowledge of whether it is an unknown attack or anomalous attack-free behavior. But at least is has not been misclassified as normal or known attack, and can be further investigated to determine its true nature.

Figure 1.11 shows the overall degrees of guilt for this machine-learning approach to anomaly detection. Initially, there is no knowledge of the state of network intrusion. This complete lack of knowledge corresponds to the center of the guilt scale. Connection rules either found in the

profile or classified as normal generate no alarms. This corresponds to moving from the center of the scale to probable innocence. Connection rules classified as known attacks correspond to moving from the center of the scale to probable guilt. Rules classified as unknown remain in the center of the scale, pending further investigation.

5. Conclusion

In this chapter, we examine the state of modern intrusion detection. The discussion follows two well-known criteria for categorizing intrusion detection systems: detection strategy and data source. The general detection strategies are misuse detection and anomaly detection, and data source categories are host-based and network-based. We introduce degree of attack guilt as an interesting way of characterizing intrusion detection activities. It provides a framework in which we analyze detection quality versus cost.

Intrusion detection systems have been an area of active research for over 15 years. Current commercial intrusion detection systems employ misuse detection. As such, they completely lack the ability to detect new attacks. The absence of this capability is a recognized gap in current systems.

Given the shortcomings of current commercial systems, an important research focus is anomaly detection through machine learning, particularly through data mining. A critical issue for anomaly detection is the need to reduce false alarms, since any activity outside a known profile raises an alarm. Research prototypes combining data mining and classification have shown great promise in this area.

Acknowledgments

This work was partially supported by the National Science Foundation under the grant CCR-0113515. We thank Sushil Jajodia and Ningning Wu for sharing their thoughts during various stages.

References

Abraham, T. (2001). IDDM: Intrusion Detection using Data Mining Techniques. Technical Report DSTO-GD-0286, DSTO Electronics and Surveillance Research Laboratory.

Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., and Stoner, E. (2000). State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, CMU, Pittsburgh, PA.

Anderson, D., Lunt, T. F., Javitz, H., Tamaru, A., and Valdes, A. (1995a). Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES). Technical Report SRI-CSL-95-06, SRI International, Menlo Park, CA.

Anderson, D., Lunt, T. F., Javitz, H., Tamaru, A., and Valdes, A. (1995b). Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES). Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA.

Axelsson, S. (1999). Research in intrusion-detection systems: A survey. Technical Report TR: 98-17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Axelsson, S. (2000a). The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security*, 3(1):186–205.

Axelsson, S. (2000b). Intrusion detection systems: A survey and taxonomy. Technical report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Barbara, D., Jajodia, S., Wu, N., and Speegle, B. (1999). Mining unexpected rules in network audit trails. Technical report, George Mason University.

Barbara, D., Wu, N., and Jajodia, S. (2001). Detecting novel network intrusions using bayes estimators. In *First SIAM Conference on Data Mining*, Chicago, IL. Society for Industrial and Applied Mathematics.

Bauer, D. S. and Koblentz, M. E. (1988). NIDX-An Expert System for Real-time. In *Computer Networking Symposium*.

Cabrera, J. B. D., Ravichandran, B., and Mehra, R. K. (2000). Statistical traffic modeling for network intrusion detection. In *8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA.

CERT Advisory (2001). Multiple vulnerabilities in bind, computer emergency response. Technical Report CA-2001-02, Computer Emergency Response Center, Carnegie Mellon University. Available as http://www.cert.org/advisories/CA-2001-02.html.

Clifton, C. and Gengo, G. (2000). Developing custom intrusion detection filters using data mining. In *21st Century Military Communications Conference*, volume 1, pages 440–443. IEEE Computer Society.

Crosbie, M., Dole, B., Ellis, T., Krsul, I., and Spafford, E. (1996). *IDIOT Users Guide*. Purdue University, West Lafayette, IN. TR-96-050.

CTC-Corporation (2000). Best of breed appendices. Tech Report 0017-UU-TE-000712.

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13:222–232.

Dowell, C. and Ramstedt, P. (1990). The computerwatch data reduction tool. In *13th National Computer Security Conference*, Washington, DC.

Engelhardt, D. (1997). Directions for intrusion detection and response: A survey. Technical Report DSTO-GD-0155, DSTO Electronics and Surveillance Research Laboratory.

Esmaili, M., Balachandran, B., Safavi-Naini, R., and Pieprzyk, J. (1996). Case-based reasoning for intrusion detection. In *12th Annual Computer Security Applications Conference*, San Diego, CA.

Esmaili, M., Safavi-Naini, R., and Balachandran, B. M. (1997). Autoguard: A continuous case-based intrusion detection system. In *Twentieth Australasian Computer Science Conference*.

Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. (1996). A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA. IEEE Computer Society.

Ghosh, A. K. and Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Usenix Security Symposium*, Washington, DC.

Heberlein, L. T., Mukherjee, B., and Levitt, K. N. (1992). Internet security monitor: An intrusion detection system for large-scale networks. In *15th National Computer Security Conference*, Baltimore, MD.

Helmer, G., Wong, J., Honavar, V., and Miller, L. (1999). Automated discovery of concise predictive rules for intrusion detection. Technical Report TR 99-01, Department of Computer Science, Iowa State University, Ames, IA.

Hochberg, J., Jackson, K., Stallings, C., McClary, J., DuBois, D., and Ford, J. (1993). NADIR: An Automated System for Detecting Network Intrusions and Misuse. *Computers and Security*, 12(3):248–253.

Ilgun, K. (1992). *USTAT A Real-time Intrusion Detection System for UNIX*. Master of science, University of California Santa Barbara.

Jackson, K. A. (1999). Intrusion Detection System (IDS) Product Survey. Technical Report LA-UR-99-3883, Los Alamos National Laboratory, Los Alamos, NM.

Javitz, H. S. and Valdes, A. (1991). The SRI IDES Statistical Anomaly Detector. In *IEEE Symposium on Research in Security and Privacy*, Oakland, CA.

Jensen, K. (1997). A Brief Introduction to Coloured Petri Nets. Technical report, presented at Tools and Algorithms for the Construction and Analysis of Systems (TACAS) Workshop, Enschede, The Netherlands.

Kemmerer, R. A. (1997). NSTAT: A Model-based Real-time Network Intrusion Detection System. Technical Report TR 1997-18, University of California Santa Barbara Department of Computer Science.

Kohavi, R., Beckeer, B., and Sommerfield, D. (1997). Improving simple bayes. In *European Conference on Machine Learning*, Prague, Czech Republic.

Kvarnstrom, H. (1999). A survey of commercial tools for intrusion detection. Technical Report TR 99-8, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden.

Lane, T. D. (2000). *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. Doctor of philosophy, Purdue University.

LaPadula, L. J. (1999). State of the art in anomaly detection and reaction. Technical Report MP 99B0000020, The MITRE Corporation, Bedford, MA.

LaPadula, L. J. (2000). Compendium of anomaly detection and reaction tools and projects. Technical Report MP 99B0000018R1, The MITRE Corporation, Bedford, MA.

Lee, W. (1999). A data mining framework for constructing features and models for intrusion detection systems. Technical report, Graduate School of Arts and Sciences, Columbia University.

Lee, W., Stolfo, S., and Mok, K. (2000). Adaptive intrusion detection: a data mining approach. *Artificial Intelligence Review*, 14:533–567.

Lee, W. and Stolfo, S. J. (1998). Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX.

Lee, W., Stolfo, S. J., and Mok, K. W. (1999). A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*.

Lee, W. and Xiang, D. (2001). Information-theoretic measures for anomaly detection. In *IEEE Symposium on Security and Privacy*, pages 130–143, Oakland, CA. IEEE Computer Society.

Liepins, G. and Vaccaro, H. (1989). Anomaly detection purpose and framework. In *12th National Computer Security Conference*, pages 495–504, Baltimore, MD. NIST and NSA.

Liepins, G. E. and Vaccaro, H. S. (1992). Intrusion detection: It's role and validation. *Computers and Security*, pages 347–355.

Lin, J.-L., Wang, X. S., and Jajodia, S. (1998). Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *11th IEEE Computer Security Foundations Workshop*.

Lindqvist, U. and Porras, P. A. (1999). Detecting Computer and Network Misuse Through the Production-based Expert System Toolset (P-BEST). In *IEEE Symposium on Security and Privacy*.

Lippmann, R. P., Fried, D. J., Graf, I., J. W. Haines, K. R. K., D., McClung, D. Weber, S. E. W., Wyschogrod, D., Cunningham, R. K., , M., and Zissman, A. (2000). Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition*.

Lundin, E. and Jonsson, E. (1999). Some practical and fundamental problems with anomaly detection. In *Proceedings of the Nordic Workshop on Secure Computer Systems*.

Lunt, T., Tamaru, A., Gilham, F., Jagannathan, R., Jalali, C., Neumann, P. G., Javitz, H. S., Valdes, A., and Garvey, T. D. (1992). A Real Time Intrusion Detection Expert System (IDES). Technical report, SRI.

Lunt, T. F. (1989). Real-time intrusion detection. In *presented at COMPCON: Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage*.

Manganaris, S., Christensen, M., Zerkle, D., and Hermiz, K. (2000). A Data Mining Analysis of RTID Alarms. *Computer Networks*, 34(No. 4):571–577.

Net-Ranger (1999). *NetRanger*. Available at www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr.

Neumann, P. G. and Porras, P. A. (1999). Experience with EMERALD to Date. In *First Usenix Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA.

Ning, P. (2001). *Abstraction-based Intrusion Detection in Distributed Environments*. Doctor of philosophy, George Mason University.

Porras, P. (1992). *STAT: A State Transition Analysis for Intrusion Detection*. Master of science, University of California Santa Barbara.

Porras, P. A. and Kemmerer, R. A. (1992). Penetration state transition analysis: A rule-based intrusion detection approach. In *Eighth Annual Computer Security Applications Conference*.

Porras, P. A. and Neumann, P. G. (1997). EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 20th National Information Systems Security Conference*, Baltimore, MD.

Real-Secure (1999). *RealSecure*. Internet Security Systems. Available at www.iss.net/customer_care/resource_center.

Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy*, Oakland, CA. IEEE Computer Society.

Smaha, S. E. (1988). Haystack: An Intrusion Detection System. In *Fourth Aerospace Computer Security Applications Conference*.

Snapp, S., Brentano, J., Dias, G., Goan, T., Grance, T., Heberlein, L., Ho, C.-L., Levitt, K. N., Mukherjee, B., Mansur, D. L., Pon, K. L., and Smaha, S. E. (1991). A system for distributed intrusion detection. In *Compcon Spring*, pages 170–176. IEEE Computer Society.

Somayaji, A., Hofmeyr, S., and Forrest, S. (1997). Principles of a computer immune system. In *New Security Paradigms Workshop*, Langdale, Cumbria UK.

Spafford, E. H. and Zamboni, D. (2000). Intrusion detection using autonomous agents. *Computer Networks*, 34(4):547–570.

Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D. (1996). GrIDS-A Graph Based Intrusion Detection System for Large Networks. In *19th National Information Systems Security Conference*, pages 361–370, Baltimore, MD. NIST and NSA.

Vaccaro, H. and Liepins, G. (1989). Detection of anomalous computer session activity. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society.

Valdes, A. and Skinner, K. (2000). Adaptive, model-based monitoring for cyber attack detection. In *Recent Advances in Intrusion Detection*, pages 80–93, Toulouse, France. Springer-Verlag.

Vigna, G. and Kemmerer, R. A. (1998). NetSTAT: A Network-based Intrusion Detection Approach. In *Proceedings of the International Conference on Knowledge and Data Mining*, New York, NY.

W. Lee, S. J. S. and Mok, K. W. (1998). Mining audit data to build intrusion detection models. In *Proceedings of the International Conference on Knowledge and Data Mining*, New York, NY.

Wagner, D. and Dean, R. (2001). Intrusion detection via static analysis. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society.

Wespi, A., Dacier, M., and Debara, H. (2000). Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection*, pages 110–129, Toulouse, FR. Springer-Verlag.

Winkler, J. R. (1990). A unix prototype for intrusion and anomaly detection in secure networks. In *13th National Computer Security Conference*, Washington, DC.

Winkler, J. R. and Landry, L. C. (1992). Intrusion and anomaly detection, isoa update. In *15th National Computer Security Conference*, Baltimore, MD.

Wu, N. (2001a). *Audit Data Analysis and Mining*. PhD thesis, George Mason University, Department of Information and Software Engineering. Fairfax, VA.

Wu, N. (2001b). Research statement.

Wu, S. F., Chang, H., Jou, F., Wang, F., Gong, F., Sargor, C., Qu, D., and Cleaveland, R. (1999). JiNao: Design and Implementation of a Scalable Intrusion Detection System for the OSPF Routing Protocol.

Yang, J., Ning, P., Wang, X. S., and Jajodia, S. (2000). CARDS: A Distributed System for Detecting Coordinated Attacks. In *IFIP TC11 16th Annual Working Conference on Information Security*, pages 171–180. Kluwer.

Chapter 2

DATA MINING FOR INTRUSION DETECTION

A Critical Review

Klaus Julisch
IBM Research
Zurich Research Laboratory
kju@zurich.ibm.com

Abstract Data mining techniques have been successfully applied in many different fields including marketing, manufacturing, process control, fraud detection, and network management. Over the past five years, a growing number of research projects have applied data mining to various problems in intrusion detection. This chapter surveys a representative cross section of these research efforts. Moreover, four characteristics of contemporary research are identified and discussed in a critical manner. Conclusions are drawn and directions for future research are suggested.

Keywords: Intrusion detection, data mining.

1. Introduction

Intrusion detection is the process of monitoring and analyzing the events occurring in a computer system in order to detect signs of security problems (Bace, 2000). Over the past ten years, intrusion detection and other security technologies such as cryptography, authentication, and firewalls have increasingly gained in importance (Allen et al., 2000). However, intrusion detection is not yet a perfect technology (Lippmann et al., 2000; Allen et al., 2000). This has given data mining the oppor-

tunity to make several important contributions to the field of intrusion detection (cf. Section 3).

This chapter gives a critical account of the past five years of data mining research in intrusion detection. To this end, we begin by introducing data mining basics in Section 2. Section 3 surveys a representative selection of research projects that used data mining to address problems in intrusion detection. In Section 4, we identify and discuss four characteristics of contemporary and past research efforts. This discussion leads to Section 5, where we suggest new directions for future research. Section 6 summarizes the chapter.

We have attempted to make this chapter as self-contained as possible. However, given the interdisciplinary nature of the topic, it was not possible to write complete introductions to both, intrusion detection and data mining. We assumed that the reader has an intrusion detection background, and consequently put more emphasis on data mining basics. Complementary to this chapter, there is an abundance of excellent introductory material to both intrusion detection (Bace, 2000; Allen et al., 2000; Debar et al., 2000) as well as data mining (Han and Kamber, 2000; Mannila et al., 2001; Berry and Linoff, 1997) that can be consulted if needed.

2. Data Mining Basics

Historically, the notion of finding useful patterns in data has been given a variety of names including data mining, knowledge discovery in databases, information harvesting, data archaeology, and data pattern analysis (Fayyad et al., 1996a; Han and Kamber, 2000). Moreover, there has been some confusion about how data mining relates to the fields machine learning and statistics (Mannila, 1996). In Subsection 2.1, we clarify the terminology and the link to related fields. Section 2.2 describes four well-known data mining techniques that have been extensively used in intrusion detection. Section 2.3 concludes the discussion by summarizing several open research challenges in the field of data mining.

2.1 Data Mining, KDD, and Related Fields

The term *data mining* is frequently used to designate the process of extracting useful information from large databases. In this chapter, we adopt a slightly different view, which is identical to the one expressed by Fayyad et al. (1996b, Chapter 1) [1]. In this view, the term *knowledge discovery in databases (KDD)* is used to denote the process of extracting useful knowledge from large data sets. *Data mining*, by contrast,

refers to one particular step in this process. Specifically, the data mining step applies so-called *data mining techniques* to extract patterns from the data. Additionally, it is preceded and followed by other KDD steps, which ensure that the extracted patterns actually correspond to useful knowledge. Indeed, without these additional KDD steps, there is a high risk of finding meaningless or uninteresting patterns (Fayyad, 1998; Klemettinen et al., 1997; Stedman, 1997).

In other words, the KDD process uses data mining techniques along with any required pre- and post-processing to extract high-level knowledge from low-level data. In practice, the KDD process is interactive and iterative, involving numerous steps with many decisions being made by the user (Fayyad et al., 1996b, Chapter 2). Here, we broadly outline some of the most basic KDD steps:

1. **Understanding the application domain:** First is developing an understanding of the application domain, the relevant background knowledge, and the specific goals of the KDD endeavor.

2. **Data integration and selection:** Second is the integration of multiple (potentially heterogeneous) data sources and the selection of the subset of data that is relevant to the analysis task.

3. **Data mining:** Third is the application of specific algorithms for extracting patterns from data.

4. **Pattern evaluation:** Fourth is the interpretation and validation of the discovered patterns. The goal of this step is to guarantee that actual knowledge is being discovered.

5. **Knowledge representation:** This step involves documenting and using the discovered knowledge.

We next turn to the link between data mining and the related disciplines of machine learning and statistics. To begin with, data mining extensively uses known techniques from machine learning, statistics, and other fields. Nevertheless, several differences between data mining and related fields have been identified in the literature (Mannila, 1996; Glymour et al., 1997; Fayyad et al., 1996a). Specifically, one of the most frequently cited characteristics of data mining is its focus on finding relatively simple, but interpretable models in an efficient and scalable manner.

In other words, data mining emphasizes the efficient discovery of simple, but understandable models that can be interpreted as interesting or useful knowledge. Thus, for example, neural networks — although

a powerful modeling tool — are relatively difficult to understand compared to rules (Cohen, 1995), trees (Quinlan, 1986), sequential patterns (Rigoutsos and Floratos, 1998), or associations (Agrawal et al., 1993). As a consequence, neural networks are of less practical importance in data mining. This should not come as a surprise. In fact, data mining is just a step in the KDD process. As such, it has to contribute to the overall goal of knowledge discovery. Clearly, only understandable patterns can qualify as "knowledge". Hence the importance of understandability in data mining.

2.2 Some Data Mining Techniques

Data mining techniques essentially are pattern discovery algorithms. Some techniques such as association rules (Agrawal et al., 1993) are unique to data mining, but most are drawn from related fields such as machine learning or pattern recognition. In this section, we introduce four well-known data mining techniques that have been widely used in intrusion detection. A broader and more detailed treatment of data mining techniques can be found elsewhere (Han and Kamber, 2000; Mannila et al., 2001; Berry and Linoff, 1997).

A potential source of confusion is that different data mining techniques assume different input data representations. For example, association rules have historically been discussed under the assumption that the input data is represented as a set of transactions (Agrawal et al., 1993; Agrawal and Srikant, 1994). Later, association rule mining over relational databases has been investigated (Srikant and Agrawal, 1996; Miller and Yang, 1997). Depending on the input data representations (sets of transactions versus relational databases), the association rule concept is presented differently. A related problem is that there are many different ways to represent the same data set in a relational database (Elmasri and Navathe, 1994). So one might wonder whether all these representations are equally adequate for the purpose of data mining. To avoid issues of data representation, we next define a unified input data format, for which all subsequent data mining techniques will be described. In practice, the available input data does not necessarily follow this format. Then, it is the responsibility of the second KDD step ("Data integration and selection", as defined on page 35) to transform the available data into the format required by the data mining techniques.

To be concrete, this section describes data mining techniques in the context of mining a single relational database table. The columns of this table are called *attributes* or *features*, and the rows are called *records*.

Table 2.1. Sample database table.

SourcePort	*SourceIP*	*DestPort*	*DestIP*	*AlarmNo*	*Time*
1234	172.16.22.3	80	172.30.100.10	90	18:20:18
1631	172.16.22.2	80	172.30.100.10	47	18:20:20
1932	172.16.22.1	23	172.23.10.100	13	18:20:21
7777	172.22.16.3	80	172.30.100.10	90	18:20:27
3284	172.16.22.3	80	172.30.100.10	90	18:20:31
6269	172.16.22.2	80	172.30.100.10	47	18:20:31
7230	172.16.22.1	27	172.23.10.100	13	18:20:32

Attributes are identifiers that allow humans to interpret the columns. Records are actual database entries. For example, knowing the attributes "SourcePort", "SourceIP", ..., "AlarmNo", and "Time", we are able to interpret the records in Table 2.1 as intrusion detection alarm messages. Finally, for a database record r and an attribute A, we write $r[A]$ to denote the projection of r on the attribute A. For example, let r = (1234, 172.16.22.3, 80, 172.30.100.10, 90, 18:20:18) be the first record in Table 2.1 and be A = Time. Then, we have $r[A]$ = 18:20:18.

2.2.1 Association Rules. Association rules, which were first proposed by Agrawal et al. (1993), capture implications between attribute values. More formally, association rules have the form

$$\left(\bigwedge_{i=1}^{m} A_i = v_i\right) \Longrightarrow \left(\bigwedge_{i=m+1}^{n} A_i = v_i\right) \; [s, c]\,, \tag{2.1}$$

where the A_i (for $i \in \{1, \ldots, n\}$) are pairwise distinct attributes, and the v_i are attribute values. The parameters s and c are called *support* and *confidence*. The support s indicates the percentage of database records that satisfy the conjunction of the rule's left- and right-hand side (i.e. the conjunction $(\wedge_{i=1}^{n} A_i = v_i)$). The confidence c is the conditional probability that a database record satisfies the rule's right-hand side, provided it satisfies the left-hand side.

Support and confidence are generally used to measure the relevance of association rules. Indeed, a high support value implies that the rule is statistically significant. Similarly, a high confidence value is characteristic of a strong association rule, whose left-hand side is predictive of its right-hand side. Obviously, strong and statistically significant as-

sociation rules are attractive. Therefore, the association rule mining problem is formulated as "find all association rules that have support and confidence beyond a user-specified minimum value." For example, imposing a minimum support of 0.4 and a minimum confidence of 0.95, we find, among others, the following association rule in Table 2.1: $(\text{AlarmNo} = 90) \Longrightarrow (\text{DestIP} = 172.30.100.10 \wedge \text{DestPort} = 80)$. From Table 2.1, we can see that this association rule has support of 3/7 (that is, 3 of the 7 records in the table satisfy the left- and right-hand side of this rule) and confidence of 1.0 (that is all the records that satisfy the rule's left-hand side also satisfy its right-hand side).

2.2.2 Frequent Episode Rules. Most data mining techniques are adapted to the analysis of unordered collections of records. Association rules are a typical representative of this class of data mining techniques. Sometimes, however, the order of records contains important information that must not be ignored. This is frequently the case when records represent events such as intrusion detection alarms. Frequent episode rules take record order into account by revealing correlations among temporally close records. Clearly, frequent episode rules are only applicable for database records that have a time attribute, as temporal proximity is otherwise undefined.

The data mining literature contains several variants of frequent episode rules (Mannila et al., 1997; Lee et al., 1998). The commonality of all these variants is that they describe implications between records that frequently occur together (i.e. at approximately the same time). In their simplest form, frequent episode rules are expressed as

$$(P \wedge Q \Longrightarrow R)\ [s, c, w]\,, \tag{2.2}$$

where P, Q, and R are predicates over a user-defined class of admissible predicates (Hätönen et al., 1996). Intuitively, this rule says that two records that satisfy P and Q, respectively, are generally accompanied by a third record that satisfies R. The parameters s, c, and w are called *support*, *confidence*, and *window width*, and their interpretation in this context is as follows:

> The support s is the probability that a time window of w seconds contains three records p, q, and r that satisfy P, Q, and R, respectively. Moreover, consider a time window of w seconds that contains a record p satisfying P and a record q satisfying Q; then, the confidence c is the probability that a record r satisfying R is also contained within the same time window.

Note that the terms support and confidence have already been used in the context of association rules. Here, however, they are redefined to reflect the new semantics of the implication, which is implication between records, rather than between attribute values.

The problem of finding frequent episode rules is stated in analogy to the problem of finding association rules. Specifically, given the triple $[s, c, w]$, the problem is to find all frequent episode rules that have support and confidence of at least s and c, respectively. In particular, this also involves finding appropriate predicates P, Q, and R out of the user-defined class of admissible predicates. Algorithms for finding frequent episode rules have been described in the literature (Mannila et al., 1997), but are beyond the scope of this section.

By way of illustration, be $s = 0.2$, $c = 0.60$, and $w = 4$ sec. For these values we find, among others, the following frequent episode rule in Table 2.1: $(\text{AlarmNo} = 90) \wedge (\text{AlarmNo} = 47) \Longrightarrow (\text{AlarmNo} = 13)$. This rule holds in Table 2.1 because a 4-sec time window that contains the alarm numbers 90 and 47 also contains the alarm number 13 with a probability that exceeds the confidence of 0.6. Moreover, the probability of a 4-sec time window to contain all three alarm numbers (i.e. 90, 47, and 13) is higher than the support of 0.2. The exact support and confidence values in this example are 4/18 and 4/6, respectively.

2.2.3 Classification. Classification (Mitchell, 1997; Han and Kamber, 2000) is the task of assigning database records to one out of a pre-defined set of target classes. The difficulty is that the target classes are not explicitly given in the database records, but must be derived from the available attribute values. In practice, building classifiers is far more difficult than using them. Therefore, most classification texts concentrate on how classifiers can be built. Moreover, building a classifier generally means "learning" it from examples.

An example will illustrate these concepts. Consider Table 2.1 and suppose we want to classify these alarms into true positives and false positives. To build a classifier for this task, we follow common practice and use a so-called *training data set*. The training data set consists of alarm messages that have already been classified into true positives and false positives. From these examples, it is possible to automatically learn a classifier that predicts the class labels of future, previously unknown database records (Mitchell, 1997). Clearly, such a classifier can be used to solve the initial task, namely, to classify the records in Table 2.1.

Classifiers can use different representations to store their classification knowledge. Two common knowledge representations are if-then rules and decision trees. *If-then rules* check record attributes in their

if-parts and postulate class labels in their then-parts. In continuation of the above example, a possible if-then rule would be "*if* SourceIP = 172.16.22.1 *and* DestPort = 27 *then* false_positive". A *decision tree* is a flow-chart-like tree structure, in which each node denotes a test on an attribute value, each branch represents an outcome of the test, and each leaf indicates a class label. Algorithms for learning classifiers, as well as techniques for evaluating their accuracy would exceed the scope of this chapter, but can be found in the book by Han and Kamber (2000).

2.2.4 Clustering. Clustering (Gordon, 1999; Jain and Dubes, 1988) seeks to group database records so that the records within a given group/cluster are similar, whereas the records of different groups/clusters are dissimilar. Obviously, the notion of similarity is key to this definition. In practice, similarity is relatively easy to define for numerical attributes, but categorical attributes such as IP addresses or port numbers are more difficult to handle (Han and Kamber, 2000; Guha et al., 2000; Ganti et al., 1999). Specifically, categorical attributes assume discrete and unordered values, which makes it difficult to define similarity. In contrast, Euclidean distance offers a widely accepted similarity measure for numerical attributes (Jain et al., 1999).

It is important to clearly distinguish clustering from classification. In classification, we are given labeled training records and our task is to learn a discrimination rule that classifies future records according to their class membership. Clustering, on the other hand, cannot rely on training data. Instead, clustering methods group records based on their similarity. This is also called "unsupervised learning" because there is no teacher that would know the correct clustering. Classification, on the other hand, is known as "supervised learning".

Clustering methods can be roughly classified into *partitioning methods* and *hierarchical methods* (Jain and Dubes, 1988; Han and Kamber, 2000). This distinction is based on the type of structure the methods impose on the data. Specifically, partitioning methods split a given data set into a user-defined number of disjoint partitions. Occasionally, however, it is desirable to obtain fuller information about the structure of a data set. In these cases, hierarchical methods may be preferable, as they decompose a data set into a hierarchically structured sequence of partitions. The root of this hierarchy is the complete data set, the leaves are the individual data records, and intermediate layers represent partitions of varying granularity. For more detailed information on clustering, we refer the reader to the already mentioned textbooks.

2.3 Research Challenges in Data Mining

In a recent paper, Smyth (2001) has identified research challenges in data mining. Three years earlier, a similar list had been compiled by different authors (Grossman et al., 1998). In this section, we summarize the subset of the research challenges that are of direct relevance to intrusion detection:

Self-tuning data mining techniques: A problem with current data mining techniques is that they often require long years of experience to be used effectively. To make data mining accessible to a broader audience, research into what could be called "self-tuning" data mining techniques is needed.

Pattern-finding and prior knowledge: Finding interesting patterns that are both useful and novel in the context of what is already known constitutes a difficult problem. In fact, it has been observed both by researchers (Brin et al., 1997; Klemettinen et al., 1994; Liu and Hsu, 1996; Silberschatz and Tuzhilin, 1996) and practitioners (Stedman, 1997) that many existing tools tend to generate large numbers of obvious or irrelevant patterns. This problem has also been observed by intrusion detection researchers (Li et al., 2000; Lee et al., 1998).

Modeling of temporal data: Today, most data mining algorithms work with vector-valued data. It is an important challenge to extend data mining algorithms to better support time series data. For example, advances in the areas of trend analysis, periodicity analysis, sequential pattern analysis, or similarity search in time series (Han and Kamber, 2000) could be of immediate benefit to intrusion detection.

Scalability: Most data mining algorithms assume that the data fits into main memory. Although success on large data sets is often claimed, this usually is the result of sampling the data sets until they fit into memory. As audit data is abundant, scalability is of clear relevance to intrusion detection.

Incremental mining: Security logging produces an endless stream of audit records. Similarly, intrusion detection systems continuously trigger new alerts. This motivates the need for incremental data mining techniques that incorporate new information as it becomes available without having to mine the entire data set again "from scratch" (Ester et al., 1998; Domingos and Hulten, 2000).

3. Data Mining Meets Intrusion Detection

The goal of intrusion detection is to detect security violations in information systems. Intrusion detection is a passive approach to security as it monitors information systems and raises alarms when security violations are detected. Examples of security violations include the abuse of privileges or the use of attacks to exploit software or protocol vulnerabilities.

Traditionally, intrusion detection techniques are classified into two broad categories: *misuse detection* and *anomaly detection* (Mounji, 1997, Chapter 2). Misuse detection works by searching for the traces or patterns of well-known attacks. Clearly, only known attacks that leave characteristic traces can be detected that way. Anomaly detection, on the other hand, uses a model of normal user or system behavior and flags significant deviations from this model as potentially malicious. This model of normal user or system behavior is commonly known as the *user or system profile*. A strength of anomaly detection is its ability to detect previously unknown attacks.

Additionally, intrusion detection systems (IDSs) are categorized according to the kind of input information they analyze. This leads to the distinction between *host-based* and *network-based* IDSs. Host-based IDSs analyze host-bound audit sources such as operating system audit trails, system logs, or application logs. Network-based IDSs analyze network packets that are captured on a network. More information on intrusion detection in general can be found, for example, in a recent book by Bace (2000).

In the past five years, a growing number of research projects have applied data mining to intrusion detection. Here, we survey a representative cross section of these projects. The intention of this survey is to give the reader a broad overview of the work that has been done at the intersection between intrusion detection and data mining. As a consequence, this section includes the most prominent projects in the field as well as some interesting niche projects that pursue less known avenues. Specifically, our rationale for including the various projects into this survey is as follows:

- MADAM ID (cf. Section 3.1) is one of the first and, with almost a dozen conference and journal papers, certainly one of the best-known data mining projects in intrusion detection.

- In our eyes, ADAM (cf. Section 3.2) is the second most widely known and well-published project in the field.

- The clustering project of Section 3.3 is still very young and probably less known, but represents a novel and interesting research thrust.
- All of the above projects perform data mining on raw network data. In Section 3.4, we present three projects that apply data mining to intrusion detection alarms. This will broaden and balance our overview of the field.
- Section 3.5 rounds off this review and briefly mentions some of the other projects that we could not discuss in more detail.

3.1 MADAM ID

The MADAM ID project at Columbia University (Lee et al., 1999a; Lee et al., 1999b; Lee and Stolfo, 2000) has shown how data mining techniques can be used to construct IDSs in a more systematic and automated manner than by manual knowledge engineering. Specifically, the approach pursued by MADAM ID is to learn classifiers that distinguish between intrusions and normal activities. Unfortunately, classifiers can perform really poorly when they have to rely on attributes that are not predictive of the target concept (Lee et al., 1997). Therefore, MADAM ID proposes association rules and frequent episode rules as means to construct additional, more predictive attributes. In the terminology of MADAM ID, these attributes are called *features*.

MADAM ID has been most extensively documented for the case of building network-based misuse detection systems. Therefore, this section also describes MADAM ID in the context of network-based misuse detection. Note, however, that there have also been experiments in applying MADAM ID to anomaly detection as well as to host-based misuse detection (Lee and Stolfo, 2000). Because of space limitations, these experiments are not described here.

Let us now consider how MADAM ID is used to construct network-based misuse detection systems. The base version of MADAM ID that we discuss here does *not* consider the packet payload of network traffic. Indeed, all network traffic is abstracted to so-called *connection records*. The attributes of connection records are intrinsic connection characteristics such as the source host, the destination host, the source and destination ports, the start time, the duration, the header flags, etc. In the case of TCP/IP networks, connection records summarize TCP sessions.

The most notable characteristic of MADAM ID is that it *learns* a misuse detection model from examples. Thus, in order to apply MADAM ID, one needs a large set of connection records that have already been classified into "normal records" or some kind of intrusion records. If

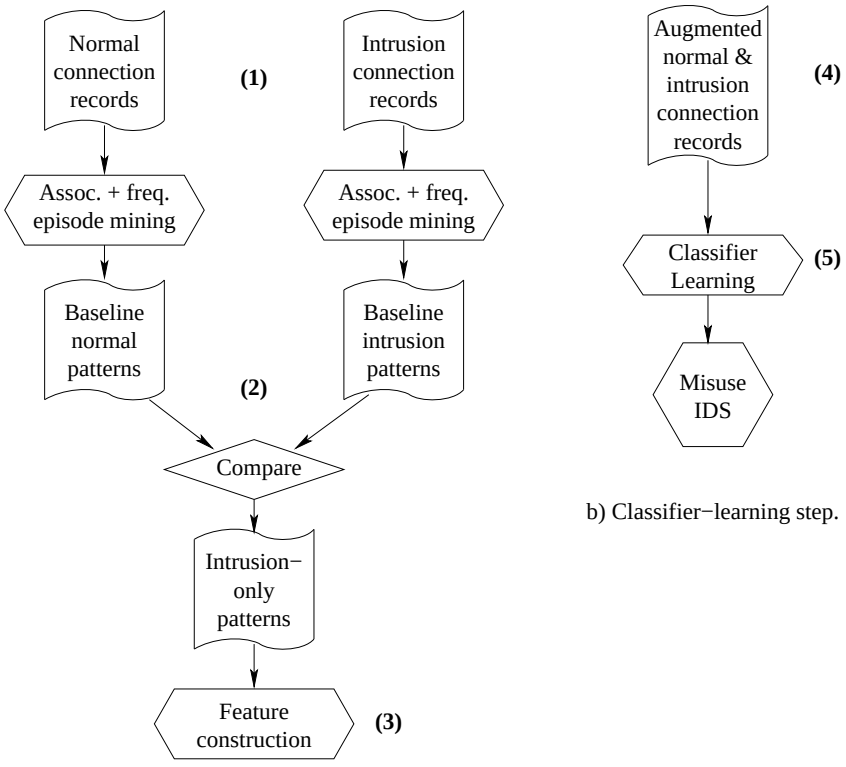


Figure 2.1. Data mining process of building misuse detection systems.

these training records are available, then MADAM ID proceeds in two steps. The *feature-construction step* augments the connection records by additional attributes that are deemed to be relevant for distinguishing intrusions from normal activities. For example, this step might add a new attribute that counts the number of connections that have been initiated during the preceding two seconds to the same destination host as the current connection. The feature-construction step is followed by the *classifier-learning step*, which learns a classifier from the augmented training records (i.e. the original training records extended by the newly constructed attributes). The entire process is illustrated in more detail in Figure 2.1:

(1) The training connection records are partitioned into two sets, namely, the *normal connection records* and the *intrusion connection records*.

(2) Association rules and frequent episode rules are mined separately from the normal connection records and from the intrusion connection records. The resulting patterns are compared, and all patterns that are exclusively contained in the intrusion connection records are collected to form the *intrusion-only patterns*.

(3) The intrusion-only patterns are used in a mostly automatic procedure to derive additional attributes, which are expected to be indicative of intrusive behavior. These additional attributes are the count, average, or percentage over connection records that share some attribute values with the current connection record.

(4) The initial training connection records are augmented by the newly constructed attributes. Note that the augmented training connection records still have their original labels.

(5) A classifier is learned that distinguishes normal connection records from intrusion connection records. This classifier — the *misuse IDS* — is the end product of MADAM ID.

It is worth pointing out that the MADAM ID process of Figure 2.1 heavily relies on intrusion detection expert knowledge. For example, expert knowledge is used to prune the number of patterns produced during association and frequent episode mining. Also, feature construction is restricted to adding attributes that an intrusion detection expert would consider promising. Moreover, separate classifiers are constructed for different attack classes, and these classifiers are subsequently combined. Finally, domain knowledge is used to manually define suitable features that summarize payload contents. All these advanced aspects of MADAM ID have been thoroughly described in numerous publications (Lee et al., 1999a; Lee et al., 1999b; Lee and Stolfo, 2000).

3.2 ADAM

ADAM (Barbará et al., 2001a; Barbará et al., 2001b; Li et al., 2000) is a network-based anomaly detection system. Like the base version of MADAM ID, it does not use the packet payload, but solely operates on connection records. ADAM learns normal network behavior from attack-free training data and represents it as a set of association rules, the so-called profile. At run-time, the connection records of the past δ seconds (δ being a parameter) are continuously mined for new asso-

ciation rules that are not contained in the profile. These "anomalous" association rules are sent to a classifier that has previously been trained to distinguish "real" anomalies (i.e. attacks) from false positives. Only anomalies that are classified as real attacks are presented to the operator.

It has been observed that association rule mining can lead to a large number of redundant or irrelevant association rules (Klemettinen et al., 1994). ADAM addresses this problem by using the following variant of the association rule concept of Section 2.2.1:

(1) The left- and right-hand sides of association rules are combined. Thus, in the ADAM system, association rules have the form $\wedge_{i=1}^{n} A_i = v_i$, instead of (2.1). Note that the confidence concept is obsolete for this kind of association rules.

(2) Only association rules that have the source host as well as the destination host or port among their attributes are allowed, i.e $\wedge_{i=1}^{n} A_i = v_i$ must satisfy $\exists k, l : (A_k = \mathit{SourceIP} \wedge A_l \in \{\mathit{DestIP}, \mathit{DestPort}\})$.

The basic ADAM system as described above has been extended in several directions. First, the normal association rules that were derived from attack-free training data have been cataloged according to the time of the day and the day of the week (Li et al., 2000). In that way, it is possible to further refine the specificity of the profile to variations during different time periods. Second, multi-level association rules (Han and Fu, 1995) have been introduced to capture coordinated and distributed attacks (Barbará et al., 2001b).

3.3 Clustering of Unlabeled ID Data

Traditional anomaly detection systems require "clean" training data (i.e. training data without anomalies) in order to learn the model of normal behavior. A major drawback of these systems is that clean training data is not always easy to obtain. Moreover, if trained over imperfect ("noisy") data, then these systems might learn to accept intrusive behavior as normal. To overcome these weaknesses, more recent research has investigated the possibility of training anomaly detection systems over noisy data (Eskin, 2000; Portnoy et al., 2001). This section surveys one such approach.

Anomaly detection over noisy data makes two key assumptions about the training data. First, the number of normal elements in the training data is assumed to be significantly larger than the number of anomalous elements. Second, anomalous elements are assumed to be qualitatively different from normal ones. Then, given that anomalies are both rare

and different, they are expected to appear as outliers that stand out from the normal baseline data. By *explicitly* modeling outliers, it is attempted to factor out anomalies and only learn the normal baseline behavior.

Specifically, the work presented by Portnoy et al. (2001) applies clustering to the training data. Here, the hope is that under a reasonable definition of similarity, intrusive elements should bundle with other intrusive elements, whereas normal elements should bundle with normal ones. Moreover, as intrusive elements are assumed to be rare, they should end up in small clusters. Thus, all small clusters are assumed to contain intrusions/anomalies, whereas large clusters are assumed to represent normal activities. At run-time, new elements are compared against all clusters, and the most similar cluster determines the new element's classification (i.e. "normal" or "intrusive").

Based on this idea, a network-based anomaly detection system has been built. Like MADAM ID, this system operates on connection records that are derived from the raw network traffic. Aside from the intrinsic attributes (e.g. source host, destination host, start time, etc.) connection records also include derived attributes such as the number of failed login attempts, the number of file-creation operations, as well as various counts and averages over temporally adjacent connection records. Similarity between connection records is defined via the Euclidean distance. Moreover, in the definition of Euclidean distance, a constant value is added whenever categorical (i.e. discrete and unordered) attributes assume distinct values.

A greedy algorithm is used to derive clusters from the training data. This algorithm starts with an empty set of clusters and generates clusters as it passes through the training data. Specifically, each connection record from the training data is compared with all existing clusters. If the distance to all existing clusters exceeds some threshold W, then the connection record starts a new cluster of its own. Otherwise, it is inserted into the most similar existing cluster. When clustering is complete, the N biggest clusters are labeled "normal" (N being a parameter), whereas all other clusters are labeled "intrusive". At this point, the system is operational and can classify new connection records according to their distance to the closest labeled cluster.

3.4 Mining the Alarm Stream

This section summarizes three research projects that applied data mining to the alarms triggered by an IDS. In the first project (Manganaris et al., 2000), the goal was to model the normal alarm stream

so as to henceforth raise the severity of "abnormal" alarms, which were deemed to be particularly noteworthy. The other two projects (Clifton and Gengo, 2000; Julisch, 2001) use data mining to extract predominant alarm patterns, which a human expert can understand and act upon. For example, based on his or her understanding, the human expert might decide to write filtering rules, fix some configuration problem, or patch a weak IDS signature. This approach is related to work that has been done in the field of network management (Klemettinen, 1999; Hellerstein and Ma, 2000; Garofalakis and Rastogi, 2001).

A preliminary remark on our terminology is in order. We need to distinguish between *alarm types* and *alarms*, the latter being instances of alarm types. For example, a source host that scans a target host will trigger an alarm of type "Reconnaissance Scan". Clearly, there can be many alarms of type "Reconnaissance Scan", but they describe different instances of the same attack.

We first discuss the work by Manganaris et al. (2000), which applies anomaly detection to the intrusion detection alarm stream. The foremost characteristics of this work is that it models alarms as tuples (t, A), where t is a timestamp and A an alarm type. In particular, all other alarm attributes such as source IP, destination IP, or port numbers are not considered in this model. The profile of normal alarm behavior is learned in two steps from historical alarm data. First, the time-ordered stream of historical alarms is partitioned into bursts, and second, association rules are mined from these bursts (more on this later). The resulting association rules constitute the profile of normal alarm behavior.

Alarm bursts are groups of alarms that are separated by prolonged alarm-free periods. Internally, alarm bursts are represented as records that have one attribute A_i for each possible alarm type A_i. Specifically, let B be an alarm burst, and let r be its internal representation. Then, $r[A_i] = 1$ holds if and only if B contains an alarm of type A_i. Otherwise, attribute A_i is set to 0, i.e. $r[A_i] = 0$. Note that this representation suppresses all temporal information about alarms. The records corresponding to alarm bursts are mined for association rules of the form $(\wedge_{i=1,\ldots,m} A_i = 1) \Rightarrow (\wedge_{i=m+1,\ldots,n} A_i = 1)$. However, association rules about *non-existing* alarm types (e.g. $(A_1 = 1) \Rightarrow (A_2 = 0)$) are explicitly not searched for.

The set of discovered association rules forms the profile of normal alarm behavior. At run-time, deviations from this profile are detected as follows: The incoming alarm stream is partitioned into bursts, and each burst is considered on its own. Various tests are used to decide whether an alarm burst B is anomalous. For example, suppose the

profile contains the association rule $(\wedge_i A_i = 1) \Rightarrow (\wedge_j \hat{A}_j = 1)$. Moreover, let us assume that alarm burst B contains all the alarm types A_i (i.e. $\forall i : A_i \in B$), but lacks some of the $\hat{A}_j$ (i.e. $\exists j : \hat{A}_j \notin B$). Then, alarm burst B is considered anomalous as it does not contain all the $\hat{A}_j$, which would have been expected based on the association rule $(\wedge_i A_i = 1) \Rightarrow (\wedge_j \hat{A}_j = 1)$. As has been said above, there are more tests like this, but these are beyond the scope of this survey.

We now summarize the two other research projects that perform data mining on intrusion detection alarms (Clifton and Gengo, 2000; Julisch, 2001). Both projects are identical in spirit as they both mine historical alarm logs in quest for new and actionable knowledge. Moreover, the intended use of this knowledge is to reduce the future alarm load, for instance, by writing filtering rules that automatically discard well-understood false positives. The major difference between the two projects lies in the choice of data mining techniques: Clifton and Gengo (2000) use frequent episode rules, whereas Julisch (2001) uses a variant of attribute-oriented induction (Han et al., 1992). Moreover, Julisch (2001) offers a more detailed experimental validation.

Attribute-oriented induction, as used by Julisch (2001), can be viewed as a data summarization technique, which operates by repeatedly replacing attribute values by more abstract values. These more abstract values are taken from user-defined generalization hierarchies, which, for example, might state that IP addresses can be generalized to networks, timestamps to weekdays, and ports to port ranges. Because of generalization, previously distinct alarms become identical and can be merged. In this way, huge alarm logs can be condensed into short and highly comprehensible summaries that make it easy to identify alarm root causes. In practice, knowledge of alarm root causes can be used to either fix them or to write filtering rules that specifically remove their associated alarms. This approach has been shown to reduce the future alarm load by an average of 80%.

3.5 Further Reading

In this section, we briefly survey other relevant work that has not yet been mentioned[2]. Wisdom & Sense (Vaccaro and Liepins, 1989) is probably the earliest system that can be considered as being based on data mining. Wisdom & Sense is an anomaly detection system that mines association rules from historical audit data to represent normal behavior. Similarly, Teng et al. (1990) use a form of automatically learned frequent episode rules to represent normal user behavior. The idea of Lankewicz and Benard (1991) is to cluster audit log records and to represent each

cluster by a single "typical" audit record. These typical audit records form the model of normal behavior against which future audit records are compared. A similar idea has been pursued by Lane and Brodley (1999), who cluster attack-free shell command sequences and define the "cluster centers" to represent normal behavior. Subsequently, anomalous command sequences can be detected based on their distance to the cluster centers. Mukkamala et al. (1999) use data mining techniques to reduce the amount of audit data that needs to be maintained and analyzed for intrusion detection. Similar work in audit data reduction has been reported by Lam et al. (1996). Finally, there is a long list of research projects that have tried to model system call sequences by a variety of different models, including neural networks, hidden Markov models, as well as fixed and variable length patterns. The work by Warrender et al. (1999) and Debar et al. (1998) is representative of this thrust of research.

4. Observations on the State of the Art

This section makes the following four observations about contemporary data mining efforts in intrusion detection:

- Most research concentrates on the construction of operational IDSs, rather than on the discovery of new and fundamental insights into the nature of attacks and false positives.
- It is very common to focus on the data mining step, while the other KDD steps are largely ignored.
- Much research is based on strong assumptions that complicate practical application.
- Up to now, data mining in intrusion detection focuses on a small subset of the spectrum of possible applications.

In the following sections, these observations will be discussed in a critical manner.

4.1 Data Mining, but no Knowledge Discovery

As discussed in Section 2.1, data mining is a step towards the overall goal of knowledge discovery. Data mining has, for example, been used to identify the biological function of genes and proteins (Brejová et al., 2000). Similarly, data mining has been used to discover correlation rules for network management (Klemettinen et al., 1997). An other case in point is the Bank of America, which has used data mining to discover

novel loan marketing strategies (Berry and Linoff, 1997). In all these cases, data mining, along with any required pre- and post-processing steps, has been used to create new and useful knowledge.

Admittedly, the position that data mining is a step towards knowledge discovery might seem academic. For example, in financial investment applications, the accuracy of predicted developments is paramount, even if the models that make these predictions are difficult to understand (Brachman et al., 1996). However, if we want to differentiate data mining from related fields such as machine learning, then we support the common position (Mannila, 1996; Glymour et al., 1997; Fayyad et al., 1996a) that data mining is directed towards knowledge discovery. In other words, the focus on knowledge discovery is one of the key characteristics that differentiate data mining from related fields such as machine learning.

However, we have the impression that data mining in intrusion detection hardly ever contributes to the discovery of new knowledge (although exceptions exist). In intrusion detection, data mining is mainly used to construct a "black box" that detects intrusions. This observation is confirmed by virtually all the systems of Section 3, including MADAM ID and ADAM. In all these systems, the desire to build an operational intrusion detection system prevails over the ideal of gaining new insights into the field of intrusion detection. As has been said before, this by no means limits the scientific contributions of these systems. However, we feel that these systems do not fully capitalize on the unique strength of data mining, which is the discovery of new and useful knowledge.

4.2 Disregard of Other KDD Steps

We have observed that intrusion detection researchers tend to use data mining as a stand-alone technique, rather than as a step in the KDD process. The limited interest in knowledge discovery, which has been discussed above, is a corollary of this observation. However, using data mining in isolation can adversely affect the quality of the results obtained. It is for this reason that we dedicate an entire section to this point.

It has been observed that the isolated application of data mining techniques can be a dangerous activity, easily leading to the discovery of meaningless or even misleading patterns (Fayyad, 1998). In particular, data mining without a proper understanding of the application domain should be avoided. To illustrate the danger of insufficient domain expertise, we reconsider the work of Manganaris et al. (cf. Section 3.4). Recall that they model alarms as tuples (t, A), where t is a timestamp and A

is an alarm type. Specifically, all other attributes such as source host or target host are ignored. In our eyes, this model is an oversimplification of reality, which limits its practical usefulness. This could probably have been avoided if more time had been spent in the early KDD steps (cf. Section 2.1).

The validation step of the KDD process is equally important, as it ensures the quality and meaningfulness of discovered patterns. By way of illustration, we think that the clustering technique of Section 3.3 could benefit from a more pronounced validation step. Specifically, the use of Euclidean distance as a similarity measure for connection records is not fully convincing. Given the existence of categorical attributes and given that attacks of one class are generally only similar in some, but not all attributes, it is unclear why Euclidean distance is a sound similarity measure. Moreover, the experiments by Portnoy et al. (2001) show that normal and intrusive connection records get mixed in the same clusters. Therefore, we believe that a thorough evaluation of the discovered clusters is needed to prove the validity of this theoretically appealing approach.

4.3 Too Strong Assumptions

There is a tendency in intrusion detection to base data mining projects on two non-trivial assumptions. First, it is frequently assumed that labeled training data is readily available, and second, it is assumed that this training data is of high quality (i.e. representative and well-distributed). We will illustrate these assumptions by means of examples.

To begin with, MADAM ID and ADAM assume the availability of labeled training data. Recall that in labeled training data, all data records have been marked as normal or some kind of intrusion. Different authors have remarked that in many cases, it is not easy to obtain labeled training data (Portnoy et al., 2001; Barbará et al., 2001a). Firstly, manual classification is error-prone and labor-intensive (Dain and Cunningham, 2001). Second, even though one could obtain labeled training data by simulating intrusions, this raises many other problems that have been discussed by McHugh (2000). Additionally, attack simulation limits the approach to the set of known attacks. It becomes clear that the availability of labeled training data is a non-trivial assumption, to say the least.

The second (frequently implicit) assumption is that training data is of high quality. For example, the clustering technique of Section 3.3 makes the following three assumptions about the quality of training data: First, the amount of normal training data is assumed to outweigh the intrusive

training data by a factor of 40 or more. Second, normal instances are assumed to be qualitatively different from intrusive ones. Third, the training data is required to contain a broad and representative collection of intrusions. Note that these intrusions do not have to be marked as such, but they are required to be in the training data (Portnoy et al., 2001). It follows that the training data has to abide by strict quality standards in order to be usable.

MADAM ID makes some non-trivial assumptions about the quality of training data, as well. Most notably, it is assumed that the training data contains *many* instances of each intrusion type. To see why this is necessary, let us assume that intrusion instances were rare in the training data. Then, the set of baseline intrusion patterns (cf. Figure 2.1) would be virtually empty as there are no association or frequent episode rules that exceed their user-defined minimum support values. This, in turn, blocks the entire MADAM ID approach. An additional difficulty arises because classifiers are not well equipped for learning "weak" signals (Cohen, 1995; Mitchell, 1997). In an attempt to avoid "overfitting", classifiers generally prune away weak signals (or rare intrusion types, for that matter). Even though it is hard to quantify these effects, it seems clear that "just any" training data will not be sufficient.

Given the difficulties associated with the generation of high-quality training data, we come to the following conclusion: Data mining techniques that rely on high-quality labeled training data will remain difficult to apply, unless we learn how to obtain such data more easily.

4.4 Narrow Scope of Research Activities

Most of the projects we have discussed in Section 3 roughly fall into two broad categories. The first category consists of projects that use data mining to build profiles for anomaly detection. This is very similar to early work in intrusion detection, except that data mining techniques instead of statistics (Smaha, 1988; Javitz and Valdes, 1991) are used to model normal behavior. The second category comprises projects that investigate feature selection for intrusion detection. MADAM ID as well as most audit log reduction projects fall into this category. Even though anomaly detection and feature selection are important, we feel that the field could benefit from a wider spectrum of research activities.

The section on future directions will illustrate this point by suggesting several new or hardly explored applications of data mining in intrusion detection. Very briefly, interesting future applications of data mining might include the discovery of new attacks, the development of better

IDS signatures, the construction of alarm correlation systems, as well as the adoption of techniques from bioinformatics. Even though this list is not extensive, it illustrates the point that many interesting research directions have not yet been sufficiently explored. Conversely, the above list highlights the narrow scope of most of today's research.

5. Future Research Directions

Building on the preceding review and discussion, we next suggest directions for future research. We first list our recommendations for future research and subsequently propose concrete example projects to illustrate the new kind of research that we advocate. Specifically, we believe that future research should address the following issues:

- Data mining projects in intrusion detection should pay closer attention to the KDD process. In that way, new and useful knowledge might be discovered (cf. Section 4.1) and stronger results can be obtained (cf. Section 4.2).
- Either more work must address the (semi-)automatic generation of high-quality labeled training data, or research should move away from assuming that such data is available. This point has been made in Section 4.3.
- Future research should investigate completely new applications of data mining in intrusion detection. Specifically, it seems desirable to break the tradition that data mining is mostly used for feature selection and the construction of anomaly detection profiles (cf. Section 4.4).
- The development of data mining techniques that are specific to intrusion detection might become important. Section 2.3 has listed five research challenges in data mining that are of direct relevance to intrusion detection. Even though the data mining community is working on these challenges, it might be easier and faster to find special-purpose solutions that are tailored to intrusion detection.

We shall now propose five example projects that illustrate the kind of future research that we advocate. The commonality of the first four project proposals is that they use data mining for knowledge discovery, that they employ no training data, and that they pursue novel or largely unexplored avenues. The fifth project proposal addresses the need for intrusion detection specific data mining techniques. Wherever possible, we include references to show that work into some of the suggested directions has already begun.

To begin with, data mining could be used to support the discovery of novel attacks. It has been observed that "bad boys" that attacked a site in the past most probably keep attacking it in the future. However, it is not uncommon for IDSs to detect only a fraction of these attacks (Almgren et al., 2000). Thus, there are *new* attacks hidden in the audit logs of misfeasors, and it would be interesting to investigate how data mining can be used to discover these attacks. The DEF CON "Capture the Flag" attack logs (DEF CON, 2000) could be an attractive starting point for such a data mining project.

Another idea is to improve IDS signatures by means of data mining. Specifically, IDS vendors could run their IDSs in operational environments where all alarms as well as the underlying audit logs are collected. Then, data mining could be used to search for audit log patterns that are closely correlated with particular alarms. This might yield new insights into why false positives arise and how they could be avoided. In fact, it has been shown that data mining on IDS alarms without access to audit logs can identify weak signatures that cause false positives (Julisch, 2001). We therefore believe that very interesting results can be obtained if IDS alarms are correlated with the underlying audit events.

Bioinformatics inspires another possibility to add more breadth to data mining in intrusion detection. Specifically, it seems that intrusion detection could benefit from the data mining techniques that have been used to analyze DNA sequences. Biologists assume that 95% of the human DNA has no function whatsoever (so-called "junk-DNA"). Finding the five percent of functional DNA, the so-called *motifs*, is a challenging problem that has received much attention (Pevzner and Sze, 2000). There is a clear parallel between motif discovery and the detection of attacks in huge audit logs. Exploring this parallel could be the subject of an interesting and potentially very rewarding data mining project.

Moreover, we believe that more data mining projects should focus on the construction of alarm correlation systems. The meta-classification idea of MADAM ID (Lee and Stolfo, 2000) as well as the alert stream fusion project by Dain and Cunningham (2001) are first steps into this direction. However, more work along these lines would be desirable. For example, we used data mining to analyze the alarm stream of a commercial IDS in an operational environment. The experiments were only preliminary, but we discovered many interesting alarm patterns with immediate application to alarm correlation:

- We discovered alarm sequences that were characteristic of particular attack tools.

- We found alarm sequences that provided an early warning about forthcoming more aggressive attacks.
- We learned that particular alarms are only relevant when they occur in combination.
- We saw that there are alarms that, for some IDS-specific reason, always occur together.

We conclude this section by proposing a project that investigates a "notion of interestingness" for intrusion detection. Many classical data mining techniques, including association rules and frequent episode rules, are restricted to finding frequent (i.e. statistically significant) patterns. Such techniques have limitations in intrusion detection, where *rare* patterns such as attack traces can be the most interesting and relevant ones. This raises the need for data mining techniques that specifically search for patterns that are interesting and relevant from an intrusion detection point of view. Lee et al. (1998) have also seen this need when introducing axis attributes, reference attributes, and level-wise mining. We believe that more research along these lines is needed.

6. Summary

This chapter has reviewed the past five years of data mining in intrusion detection. Based on this review, we have made four observations about contemporary and past research efforts. Very briefly, we observed a focus on building operational IDSs, a disregard for the overall KDD process, the reliance on labeled high-quality training data, and the focus on a few, admittedly important problems. We have discussed these observations in a critical manner, which has lead us to the following recommendations for future research:

- Future projects should pay closer attention to the KDD process.
- Either more work should address the (semi-)automatic generation of high-quality labeled training data, or the existence of such data should no longer be assumed.
- Future projects should explore novel applications of data mining that do not fall into the categories feature selection and anomaly detection.
- To deal with some of the general challenges in data mining, it might be best to develop special-purpose solutions that are tailored to intrusion detection.

Acknowledgments

The author thanks Birgit Baum-Waidner, Marc Dacier, Andreas Wespi, and Diego Zamboni for their valuable comments on earlier versions of this chapter.

This research was supported by the European IST Project MAFTIA (IST-1999-11583), which is partially funded by the European Commission and the Swiss Federal Office for Education and Science. The views herein are those of the author and do not necessarily reflect the views of the supporting agencies.

Notes

1. Other authors also support this view (Mannila, 1996; Han and Kamber, 2000).
2. To avoid redundancy, we deliberately refrain from discussing the other papers in this volume.

References

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining Associations between Sets of Items in Massive Databases. In *Proceedings of the ACM-SIGMOD 1993 International Conference on Management of Data*, pages 207–216.

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases*, pages 487–499.

Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., and Stoner, E. (2000). State of the Practice of Intrusion Detection Technologies. Technical report, Carnegie Mellon University. `http://www.cert.org/archive/pdf/99tr028.pdf`.

Almgren, M., Debar, H., and Dacier, M. (2000). A Lightweight Tool for Detecting Web Server Attacks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'00)*, pages 157–170.

Bace, R. (2000). *Intrusion Detection*. Macmillan Technical Publishing.

Barbará, D., Couto, J., Jajodia, S., Popyack, L., and Wu, N. (2001a). ADAM: Detecting Intrusions by Data Mining. In *Proceedings of the IEEE Workshop on Information Assurance and Security*.

Barbará, D., Wu, N., and Jajodia, S. (2001b). Detecting Novel Network Intrusions Using Bayes Estimators. In *Proceedings of the first SIAM International Conference on Data Mining (SDM'01)*.

Berry, M. J. A. and Linoff, G. (1997). *Data Mining Techniques*. John Wiley and Sons, Inc.

Brachman, R. J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., and Simoudis, E. (1996). Mining Business Databases. *Communications of the ACM*, 39(11):42–48.

Brejová, B., DiMarco, C., Vinar, T., and Hidalgo, S. (2000). Finding Patterns in Biological Sequences. Technical report, University of Waterloo.

Brin, S., Motwani, R., Ullman, J., and Tsur, S. (1997). Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 255–264.

Clifton, C. and Gengo, G. (2000). Developing Custom Intrusion Detection Filters Using Data Mining. In *Military Communications International Symposium (MILCOM2000)*.

Cohen, W. W. (1995). Fast Effective Rule Induction. In *Proceedings 12th International Conference on Machine Learning*, pages 115–123.

Dain, O. and Cunningham, R. K. (2001). Fusing Heterogeneous Alert Streams into Scenarios. In *Proceedings of the ACM CCS Workshop on Data Mining for Security Applications*.

Debar, H., Dacier, M., Nassehi, M., and Wespi, A. (1998). Fixed vs. Variable-Length Patterns for Detecting Suspicious Process Behavior. In *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 1–15.

Debar, H., Dacier, M., and Wespi, A. (2000). A Revised Taxonomy for Intrusion Detection Systems. *Annales des Télécommunications*, 55(7–8):361–378.

DEF CON (2000). DEF CON Capture The Flag Contest. `http://www.defcon.org`.

Domingos, P. and Hulten, G. (2000). Mining High-Speed Data Streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80.

Elmasri, R. and Navathe, S. B. (1994). *Fundamentals of Database Systems*. Addison-Wesley.

Eskin, E. (2000). Anomaly Detection over Noisy Data Using Learned Probability Distributions. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Ester, M., Kriegel, H.-P., Sander, J., Wimmer, M., and Xu, X. (1998). Incremental Clustering for Mining in a Data Warehousing Environment. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB'98)*, pages 323–333.

Fayyad, U. (1998). Mining Databases: Towards Algorithms for Knowledge Discovery. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 22(1):39–48.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996a). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17(3):37–54.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996b). *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press.

Ganti, V., Gehrke, J., and Ramakrishnan, R. (1999). CACTUS – Clustering Categorical Data Using Summaries. In *5th ACM SIGKDD International Conference on Knowldege Discovery and Data Mining*, pages 73–83.

Garofalakis, M. and Rastogi, R. (2001). Data Mining Meets Network Management: The Nemesis Project. In *Proceedings of the ACM SIGMOD International Workshop on Research Issues in Data Mining and Knowledge Discovery*.

Glymour, C., Madigan, D., Pregibon, D., and Smyth, P. (1997). Statistical Themes and Lessons for Data Mining. *Data Mining and Knowledge Discovery*, 1(1):11–28.

Gordon, A. (1999). *Classification*. Chapman and Hall.

Grossman, R., Kasif, S., Moore, R., Rocke, D., and Ullman, J. (1998). Data Mining Research: Opportunities and Challenges. Technical report, Workshop on Managing and Mining Massive and Distributed Data (M3D2).

Guha, S., Rastogi, R., and Shim, K. (2000). ROCK: A Robust Clustering Algorithm for Categorical Attributes. *Information Systems*, 25(5):345–366.

Han, J., Cai, Y., and Cercone, N. (1992). Knowledge Discovery in Databases: An Attribute-Oriented Approach. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 547–559.

Han, J. and Fu, Y. (1995). Discovery of Multi-Level Association Rules from Large Databases. In *Proceedings of the 21th Very Large Databases Conference*, pages 420–431.

Han, J. and Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publisher.

Hätönen, K., Klemettinen, M., Mannila, H., Ronkainen, P., and Toivonen, H. (1996). Knowledge Discovery from Telecommunication Network Alarm Databases. In *Proceedings of the 12th International Conference on Data Engineering*, pages 115–122.

Hellerstein, J. L. and Ma, S. (2000). Mining Event Data for Actionable Patterns. In *The Computer Measurement Group*. `http://www.research.ibm.com/PM/`.

Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data*. Prentice-Hall.

Jain, A., Murty, M., and Flynn, P. (1999). Data Clustering: A Review. *ACM Computing Surveys*, 31(3).

Javitz, H. S. and Valdes, A. (1991). The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA*. SRI International.

Julisch, K. (2001). Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*.

Klemettinen, M. (1999). *A Knowledge Discovery Methodology for Telecommunication Network Alarm Data*. PhD thesis, University of Helsinki (Finland).

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo, A. (1994). Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 401–407.

Klemettinen, M., Mannila, H., and Toivonen, H. (1997). A Data Mining Methodology and Its Application to Semi-Automatic Knowledge Acquisition. In *Proceedings of the 8th International Workshop on Database and Expert System Applications (DEXA '97)*, pages 670–677.

Lam, K.-Y., Hui, L., and Chung, S.-L. (1996). A Data Reduction Method for Intrusion Detection. *Journal of Systems and Software*, 33:101–108.

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

Lankewicz, L. and Benard, M. (1991). Real-Time Anomaly Detection Using a Non-Parametric Pattern Recognition Approach. In *Proceedings of the 7th Annual Computer Security Applications Conference*.

Lee, W. and Stolfo, S. J. (2000). A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4):227–261.

Lee, W., Stolfo, S. J., and Mok, K. W. (1997). Data Mining Approaches for Intrusion Detection. In *Proceedings of the Seventh USENIX Security Symposium (SECURITY '98)*, pages 120–132.

Lee, W., Stolfo, S. J., and Mok, K. W. (1998). Mining Audit Data to Build Intrusion Detection Models. In *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 66–72.

Lee, W., Stolfo, S. J., and Mok, K. W. (1999a). A Data Mining Framework for Building Intrusion Detection Models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 120–132.

Lee, W., Stolfo, S. J., and Mok, K. W. (1999b). Mining in a Data-flow Environment: Experience in Network Intrusion Detection. In *Proceed-*

ings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99), pages 114–124.

Li, Y., Wu, N., Jajodia, S., and Wang, X. S. (2000). Enhancing Profiles for Anomaly Detection Using Time Granularities. In *Proceedings of the First ACM Workshop on Intrusion Detection Systems (WIDS)*.

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., and Zissman, M. A. (2000). Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, pages 12–26.

Liu, B. and Hsu, W. (1996). Post-Analysis of Learned Rules. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 828–834.

Manganaris, S., Christensen, M., Zerkle, D., and Hermiz, K. (2000). A Data Mining Analysis of RTID Alarms. *Computer Networks*, 34(4).

Mannila, H. (1996). Data Mining: Machine Learning, Statistics, and Databases. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management*, pages 1–8.

Mannila, H., Smyth, P., and Hand, D. J. (2001). *Principles of Data Mining*. MIT Press.

Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1:259–289.

McHugh, J. (2000). The 1998 Lincoln Laboratory IDS Evaluation – A Critique. In *3th Workshop on Recent Advances in Intrusion Detection (RAID)*, pages 145–161.

Miller, R. and Yang, T. (1997). Association Rules Over Interval Data. In *Proceedings of the 1997 ACM-SIGMOD Conference on Management of Data*, pages 452–461.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mounji, A. (1997). *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix Namur (Belgium).

Mukkamala, R., Gagnon, J., and Jajodia, S. (1999). Integrating Data Mining Techniques with Intrusion Detection Methods. In *Proceedings of the 13th IFIP WG11.3 Working Conference on Database Security*, pages 33–46.

Pevzner, P. A. and Sze, S.-H. (2000). Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 269–278.

Portnoy, L., Eskin, E., and Stolfo, S. J. (2001). Intrusion Detection with Unlabeled Data Using Clustering. In *Proceedings of the ACM CCS Workshop on Data Mining for Security Applications.*

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1):81–106.

Rigoutsos, I. and Floratos, A. (1998). Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm. *Bioinformatics*, 14(1):55–67.

Silberschatz, A. and Tuzhilin, A. (1996). On Subjective Measures of Interestingness in Knowledge Discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 275–281.

Smaha, S. E. (1988). Haystack: An Intrusion Detection System. In *Proceedings of the 4th IEEE Aerospace Computer Security Applications Conference, Orlando, FL*, pages 37–44.

Smyth, P. (2001). Breaking out of the Black-Box: Research Challenges in Data Mining. In *Proceedings of the ACM SIGMOD International Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'01).*

Srikant, R. and Agrawal, R. (1996). Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of the 1996 ACM-SIGMOD Conference on Management of Data*, pages 1–12.

Stedman, C. (1997). Data Mining for Fool's Gold. *Computerworld*, 31(48).

Teng, H. S., Chen, K., and Lu, S. C. (1990). Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 278–284.

Vaccaro, H. S. and Liepins, G. E. (1989). Detection of Anomalous Computer Session Activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 280–289.

Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting Intrusions Using System Calls: Alternative Data Models. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA*, pages 133–145.

Chapter 3

AN ARCHITECTURE FOR ANOMALY DETECTION *

Daniel Barbará
ISE Dept. and Center for Secure Information Systems, MSN 4A4
George Mason University, Fairfax, VA 22030
dbarbara@gmu.edu

Julia Couto
Center for E-business
George Mason University, Fairfax, VA 22030
jics@ise.gmu.edu

Sushil Jajodia
Center for Secure Information Systems
George Mason University, Fairfax, VA 22030
jajodia@gmu.edu

Ningning Wu
I.S. Dept.
University of Arkansas at Little Rock, Little Rock, AR 72204
nxwu@ualr.edu

1. Introduction

Anomaly detection systems have become popular over the years. Their basic principle is the comparison of the incoming traffic with a previously-

*This research has been funded by the Air Force Research Laboratory, Rome, NY under the contract F30602-00-2-0512.

built profile that contains a representation of the "normal" or expected traffic. The system flags anything that exceeds the normal activity (usually by means of thresholds) as an attack. Unfortunately, not everything that surpasses the expected activity is indeed an attack. Thus, anomaly detection systems have the proclivity of generating lots of false alarms. In this chapter we present an efficient architecture that can effectively be used to design anomaly detection systems and keep false alarms at a manageable level. We also present an implementation of this architecture that we have realized and experimented with.

Some IDS systems [11, 8, 2, 14] are built with a traditional signature-based component that coexists with a statistical profiling unit. The statistical profiling unit has as its guiding principle finding behavior that looks anomalous with respect to a profile (in data mining this is known as finding "outliers").

The statistical unit of these IDS maintains a knowledge base of profiles, i.e., descriptions of normal behavior with respect to a set of selected measures. (Full details about the unit can be found in [10, 3].) The idea is to describe the audited activity as a vector of intrusion-detection variables and compare this vector to another one defined by the expected values stored in the profiles. If the audited activity vector proves to be sufficiently far from the expected behavior, an anomaly is flagged. This vector, or *summary test statistic* (in the terminology of IDES) is formed from many individual measures, such as CPU usage and file access. Each measure reflects the extent to which a particular type of behavior is similar to the historical profile built for it. The way that this is computed is by associating each measure to a corresponding random variable. The frequency distribution of is built (and updated) over time, as more audit records are analyzed. Examples of measures are the rate of audit record activity every 60 seconds and the user CPU time.

The frequency distribution is computed as an exponential weighted sum with a half-life of 30 days. The half-life value makes audit records that were gathered 30 days in the past to contribute with half as much weight as recent records; those gathered 60 days in the past contribute one-quarter as much weight, and so on. This is, in effect, a way to control the number of audit records that play a role in the distribution. The frequency distribution of Q_i can be computed in this manner for both continuous (numerical) and categorical measures. (For details see [10, 3].) The frequency distribution is kept in the form of a histogram with probabilities associated with each one of the possible ranges, or bins, that the measure can take. The cumulative frequency distribution is then built by using the ordered set of bin probabilities. Using this frequency distribution, and the value of the corresponding measure for the

current audit record, it is possible to compute a value that reflects how far away from the "normal" value of the measure the current value is. The actual computation whose details can be found in [10, 3], renders a value that is correlated with how abnormal this measure is. Combining the values obtained for each measure, and taking into consideration the correlation between measures, the unit computes an index of how far the current audit record is from the normal state. Records beyond a threshold are flagged as possible intrusions. Again, this have the potential of generating a large rate of false alarms.

2. Architecture

Figure 3.1 shows the basic architecture. This is a generic architecture that can be implemented in many ways. In fact two types of implementation choices have to be made. First, the type of *events* that the system is going to deal with (e.g., packets, connections, commands). Secondly, the ways that each of the different modules are going to be implemented.

The architecture is composed of the following modules:

2.1 Filter

The role of this module is to reduce the number of events that need attention, thereby improving the speed of the final diagnose. In this way, the module that will diagnose the cases can take a more detailed look at a series of features that characterize these events. In an ideal situation, the profiler should choose events in a way that produces no false negatives. In other word, every intrusion must be flagged by the filter. The reason for this is that events that are not flagged by the filter will not be looked upon anymore, so intrusions that are missed by the filter will go unnoticed. On the other hand, it is allright if events that do not correspond to intrusions are flagged (false positives). This is done to allow more flexibility to the design of the filter, and it is precisely the reason why a diagnosing step is added later. This concept is in tune with the well-known fact that anomaly systems do produce a considerable amount of false alarms. Trying to cut down on false alarms at this stage will also bring about the increase of false negatives.

In reality, it is impossible to design a filter that does not allow any false negatives (simply because it is always possible to conceive an attack that goes unnoticed by a given system). So we aim to minimize those while sacrificing the rate of false positives. In simple terms: if something looks suspicious the filter will flag it, at the risk of including events that are perfectly normal.

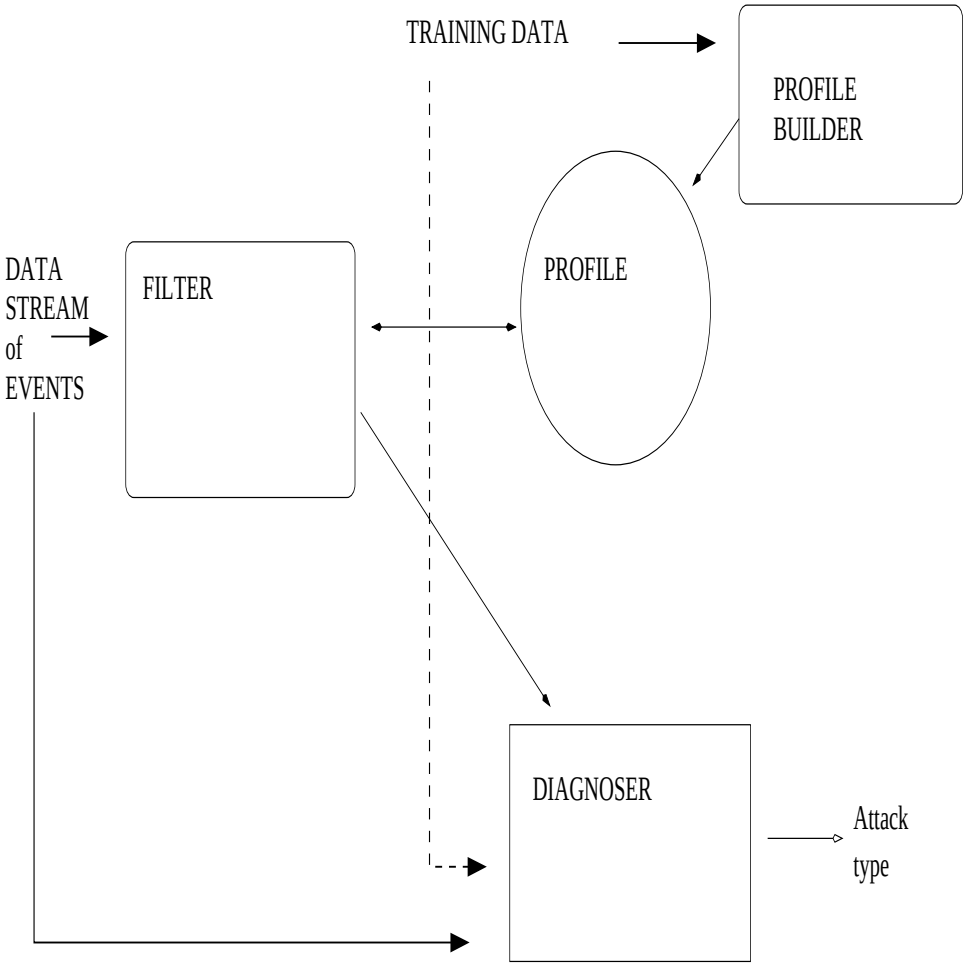


Figure 3.1: The Architecture.

The filter bases its decisions on a database called the *profile*, whose description is given in the next subsection. Simply put, the filter will compare incoming events to those included in the profile. If the event is substantially different (by a given measure) to the ones stored in the profile, it will be flagged as suspicious.

2.2 Profile

As mentioned before, the profile is the guiding block for the filtering process. The profile can be viewed as a database of *normal activities* in the system. The description of the activities in the profile has to be formal and concrete, and well-understood by the filter. The filter has to generate the same kind of activity format, based on the incoming events, so these incoming activities can be compared with the ones stored in the profile. Activities can range from rules to usage statistics, with the only constraint that they ought to be readily generated by the filter.

2.3 Profile Builder

This module takes the training data available (which is supposed to consist of events labeled as attacks or normal activity, or even possibly labeled with the name of the attack), and produces the profile described above.

2.4 Diagnoser

This module is the key to reducing the false alarm rate. The module takes the suspicious events flagged by the filter and, possibly using other features from the stream of events, produces a final diagnose. This diagnose could be as simple as deciding whether the event is an attack or not, or more elaborate, identifying the class of attack. The diagnoser may need to be previously using the training data available.

3. ADAM: an implementation of the architecture

ADAM is a testbed for using data mining techniques to detect intrusions. ADAM [4] uses a combination of association rules mining and classification to discover attacks in a TCPdump audit trail. As an implementation of the architecture shown in Figure 3.1, ADAM uses as events the *TCP connections* observed by sniffing at network traffic. It obtains those by pre-processing an audit trail such as the one obtained by TCPDUMP.

First, ADAM builds a repository of "normal" frequent itemsets that hold during attack-free periods. It does so by mining data that is known to be free of attacks. Secondly, ADAM runs a sliding-window, on-line algorithm that finds frequent itemsets in the last D connections and compares them with those stored in the normal itemset repository, discarding those that are deemed normal. With the rest, ADAM uses a classifier which has been previously trained to classify the suspicious connections as a know type of attack, an unknown type or a false alarm.

The filter, profile, and profile builder of ADAM are all based on the concept of association rules. Association rules are used to gather necessary knowledge about the nature of the audit data, on the assumption that discovering patterns within individual records in a trace can improve the classification task. The task of mining association rules, first presented in [1] consists in deriving a set of rules in the form of $X \longrightarrow Y$ where X and Y are sets of attribute-values, with $X \bigcap Y = \emptyset$ and $\|Y\| = 1$. The set X is called the antecedent of the rule while the item Y is called consequent. For example, in a market-basket data of supermarket transactions, one may find that customers who buy *milk* also buy *honey* in the same transaction, generating the rule $milk \longrightarrow honey$. There are two parameters associated with a rule: *support* and *confidence*. The rule $X \longrightarrow Y$ has *support* s in the transaction set T if $s\%$ of transactions in T contain $X \cup Y$. The rule $X \longrightarrow Y$ has *confidence* c if $c\%$ of transactions in T that contain X also contain Y. The most difficult and dominating part of an association rules discovery algorithm is to find the itemsets $X \bigcup Y$, that have strong support. Once an itemset is deemed to have strong support, it is an easy task to decide which item in the itemset can be the consequent by using the confidence threshold. In fact, we are only interested in itemsets, rather than in rules.

ADAM uses connections as events, obtaining the connections from the raw packet data of the audit trail. This preprocessing results in a table with the following schema:

$R(T_s, Src.IP, Src.Port, Dst.IP, Dst.Port, FLAG)$.

In this schema, T_s represents the beginning time of a connection, $Src.IP$ and $Src.Port$ refer to source IP and port number respectively, while $Dst.IP$ and $Dst.Port$, represent the destination IP and port number. The attribute $FLAG$ describes the status of a TCP connection. The relation R contains the dataset that is subject of the association mining. The number of potential itemsets is large: connections may come from a large base of source IP addresses and ports. We focus in itemsets that contain items that indicate the source of the connection (like source IP and port), and items that indicate its destination (like destination IP and port). We also consider itemsets that are "aggregations" of source IP or

Port values, e.g., connections that come from a source domain and have the same destination IP. We call these itemsets *domain-level* itemsets. (For instance, we want to discover frequent connections from Source IP X to Destination IP Y, or from Source Domain W to Destination IP Y.)

First, ADAM is trained using a data set in which the attacks and the attack-free periods are correctly labeled. In a first step, association rule mining is applied to the training set in order to create a profile. To make the profile independent of individual IP addresses, ADAM manipulates the results of the association rule mining to create *generalized association itemsets*. These itemsets differ from the ones explained above in that the values of $Source.IP$ and $Source.Port$ are generic. In other words, even though we perform a regular association mining task, the frequent itemsets are transformed erasing the actual source IP and port addresses. For example, an itemset in the profile could look as follows:

$Source.IP, Source.Port, 127.255.255.255, 40, m, s$

This indicates itemsets of connections coming from an specific source IP and port to destination 127.255.255.255 and port 40. The values m and s correspond to the mean and standard deviation support of all the itemsets of this form found in the training data set. It is important to remark that the itemset shown does not represent *aggregated* connection traffic to the destination IP and port. Rather it represents a characterization of the traffic going to that IP address and port.

Domain itemsets are found by the *aggregated* traffic of selected IP addresses over a destination. They are stored in the profile marking the source as a generalized, domain level source. For instance, the itemset:

$Domain - Source_i, 127.255.255.255, 40, s$

would indicate that the aggregation of several sources sending connections to 127.255.255.255 port 40, resulted in a frequent itemset. The index i indicates what type of aggregation was used to find this frequent domain itemset.

The itemsets in this profile database can be cataloged according to the time of the day and day of the week, to further refine the specificity of these rules to variations of workload during the different time periods. The mining algorithm used for this first step of the training phase is an off-line algorithm. Thus, a conventional association rule mining algorithm can be used to drive this phase. (Although we use an algorithm tailored specifically for the kinds of itemsets we aim to find, and which runs considerably faster than a general-purpose association rules algorithm.)

For the filter, ADAM uses an incremental, on-line algorithm to detect itemsets that receive strong support within a period of time. This algorithm is driven by a sliding window of tunable size δ. The algo-

rithm outputs itemsets that have received strong support during this window, and compares any itemset that starts receiving support with itemsets in the profile database for an analogous time and day of the week. Notice that the comparison has to be done with generalized itemsets. So, for instance, if a frequent itemset of the form 100.255.255.255, 127.255.255.255, 40 is found by the filter, indicating frequent traffic that generates in IP 100.255.255.255 to the destination 127.255.255.255 port 40, this itemset would be compared to a generalized itemset of the form

$$Source.IP, 127.255.255.255, 40, s,$$

if such itemset exists in the profile. If the generalized itemset is present in the profile database, the filter compares its support to the measure m of the generalized itemset. As a rule of thumb, if the support of the newly discovered itemset is greater that m plus a multiple of the standard deviation for the generalized itemset, s, (where the multiple can be simply 1), the itemset is flagged as suspicious. On the other hand, if the itemset is not in the profile, and its support surpasses a threshold, that itemset is also reported as suspicious. For a set of suspicious itemsets, we provide two services. First the ability to drill down and find the raw data in the audit trail that gives rise to these rules. Secondly, We annotate suspicious itemsets with a vector of parameters (based on the raw audit trail data that gave rise to the rules). Since we know where the attacks are in the training set, the corresponding suspicious itemsets along with their feature vectors are used to train a classifier. This classifier corresponds to the diagnoser module of our architecture. The trained classifier will be able to, given a suspicious itemset and a vector of features, classify it as a known attack (and label it with the name of the attack), as an unknown attack (whose name is not known), or as a false alarm. It is important to remark here that ADAM has the ability of classifying a suspicious event (itemset and features) as an unknown attack. Notice that no training set can possibly prepare a classifier for an unknown attack (since there can be no examples of such an event). In general labeling events as unknown attacks (or anomalies) is a very difficult problem. We are able to include such a provision by using an artifact present in some classifiers: the inclusion of a "default" label by which the classifier expresses its inability to recognize the class of the event as one of the known classes. We take the approach that any event flagged by the association rules software that cannot be classified as a known attack or as a normal event (false alarm) by the classifier, ought to be considered, conservatively, as an unknown attack. Using this assumption, we change the label in the classifier from "default" to "unknown." Our experiments have shown that this is a very efficient way to detect attacks whose nature is not fully understood.

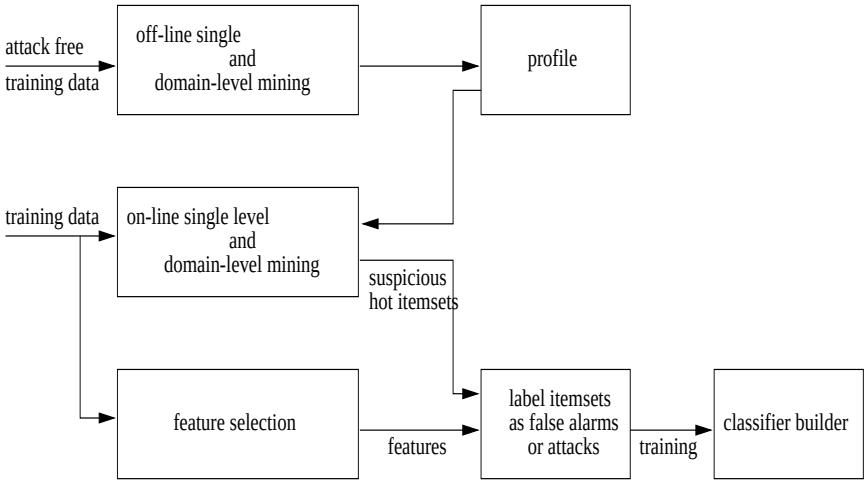


Figure 3.2: The training phase of ADAM.

ADAM is then ready to detect intrusions online. Again, the on-line association rules mining algorithm is used to process a window of the current connections. Suspicious connections are flagged and sent along with their feature vectors to the trained classifier, where they will be labeled accordingly.

Figures 3.2 and 3.3 show the basic architecture of our system. Our system performs its task in two phases. In the training mode , depicted in Figure 3.2, we use a data stream for which we know where the attacks (and their type) are located. The attack-free parts of the stream are fed into a module that performs off-line association rules discovery. The output of this module is a profile of rules that we call "normal," i.e., that depict the behavior during periods where there are no attacks. The profile along with the training data set is also fed into a module that uses a combination of a dynamic, on-line algorithm for association rules, whose output consists of frequent itemsets that characterize attacks to the system. These itemsets, along with a set of features extracted from the data stream by a features selection module are used as the training set for a classifier (decision tree). This whole phase takes place once (off-line), before we use the system to detect intrusions.

The other phase, i.e., the actual detection of intrusions is implemented as depicted in 3.3. Here the dynamic algorithm is used to produce itemsets that are considered as suspicious and, along the features extracted by the features selection module are fed to the (already trained) classifier, which labels the events as attacks (including its presumed type),

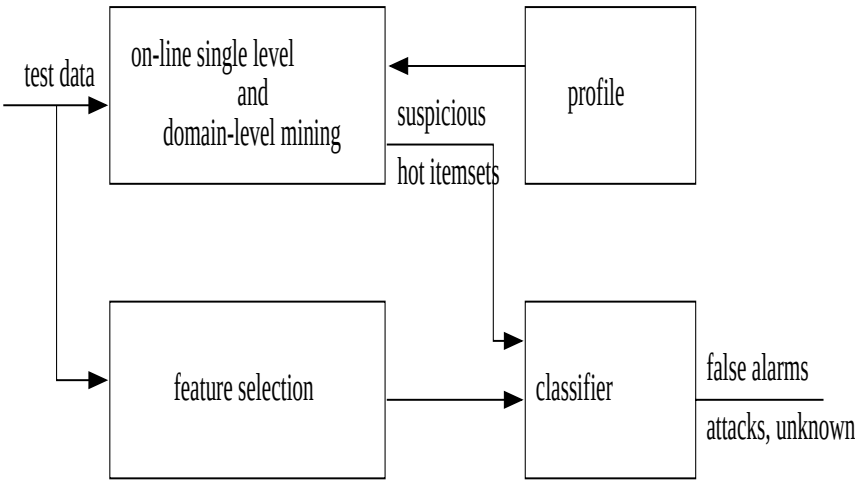


Figure 3.3: Discovering intrusions with ADAM.

false alarms, or unknown. When the classifier labels connections as false alarms, it is filtering them out of the attacks set, avoiding passing these alarms to the security officer. The last class, i.e., unknown, is reserved for events whose exact nature cannot be pinpointed by the classifier (they cannot be classified as known attacks). We consider those as attacks and include them in the set of alarms passed to the security officer.

4. Experiences

An earlier version of ADAM competed successfully in the MIT Lincoln Labs Intrusion Detection competition [12]. ADAM ranked third, after SRI's EMERALD [14] and the University of California Santa Barbara's STAT system [13, 3, 9, 16]. The attacks detected by those systems and missed by ADAM were those that fell below our thresholds, being attacks that involved usually only one connection and that can be best identified by signature-based systems. ADAM came very close to EMERALD in overall attack identification. A summary of these results can be found in [4]. The version of ADAM that competed in 1999 differs from the one described here in two important ways:

- It did not use generalized association itemsets in the profile, but rather plain itemsets with their associated support. The itemsets that were stored in the profile were the ones that exhibited high support in the training data set.

Name of attack	Number in test data	Detected by old version of ADAM	Detected by ADAM	
			mean	mean+ std. dev.
Apache2	3	3	3	3
back	1	0	1	1
dict+guest	11	3	6	10
ipsweep	7	4	5	5
mailbomb	4	4	4	4
mscan	1	0	1	1
neptune	4	3	3	3
pod	4	4	4	4
portsweep	15	2	2	5
processtable	4	3	2	3
Satan	2	1	2	2
smurf	5	4	5	5
snmpget	4	3	2	2
teardrop	3	3	3	3
udpstorm	2	2	2	2
Total	70	39	46	51
%detected	-	55.71%	65.57%	72.85%

Figure 3.4. Results of using ADAM in the 1999 Lincoln Labs competition data.

- The filter did not take anything but support (no mean or standard deviation) for the comparison of itemsets in the discovery step.

Experiments conducted with the version of ADAM that uses generalized rules and tagged with mean and standard deviation show a substantial improvement (around 30%) in the ability of ADAM to detect attacks over the old version of the system. Some of these results are summarized in Figure 3.4, which shows the name of each attack detected, the number of these attacks present in the test data set, and the number detected by ADAM using two different thresholds: the mean, and the mean plus standard deviation (recall that a rule is flagged as suspicious if its support is greater than the median plus a multiple of the standard deviation). The number of false alarms is kept under 10 per day.

5. Breaking the dependency on training data

Training data is difficult to come by for two reasons. First, organizations rarely keep logs of labeled attacks and normal connections. Secondly, labeling audit trails is a labor intensive task which will overtax the already burdened security officers.

In [5], we propose an investigate a method based on pseudo-Bayes estimators to enhance the capabilities of ADAM. The method aims to effectively detect novel attacks for which no training data is available. Pseudo-Bayes estimators is a well used technique in the area of discrete multivariate analysis [7]. It is used to provide the estimated cell values of contingency tables which may contain a large number of sampling zeros. All too often, the observed table of counts provides an unsatisfactory table of estimated values since there are many cells and few observations per cell, and some zeros are "smaller" than others, especially when we are computing rates. For instance, it may be misleading to report both 0/5 and 0/500 as being equal to zero, since as rates they carry quite different information. In order to distinguish such zero properties from one another , the observed counts need to be smoothed since the observed table of counts seems too abrupt. The basic problem here is one of simultaneously estimating a large number of parameters (the expected cell frequencies). One way to provide such estimates is to assume an underlying parametric model for the expected frequencies, where the number of parameters in the model is typically much smaller than the total number of cells. pseudo-Bayes is another approach to solve the problem which does not involve the problem of model selection. Our experimental results show that the method is very effective. in detecting the new attacks whose properties are different and distinguishable from the normal instances of training data.

6. Future

As pointed out early, ADAM is a testbed to research which data mining techniques are appropriate for intrusion detection. To this effect we continue enhancing the system by adding new methods of detection. In particular we are now investigating the following two problems:

- Breaking the dependency on training data for normal events: Even though the pseudo-Bayes method allows us to avoid the dependency on training data for attacks, ADAM still requires some training data to build the profile of normal activity. The next step is to take a trail of audit data, and break it into chunks of certain duration (e.g., a day), purging from each chunk the outliers, using

statistical methods (for a good treatment of these methods see [6]), and then merge the results (perhaps by voting) obtained on each chunk to form a profile of normal activity.

- Merging the results from different sensors: In spite of the furious activity in intrusion detection, the problem of false alarms and missed attacks (false positives and negatives) is still prevalent. Although we believe that no system will ever be built that completely eliminate these problems, we are also convinced that the way to minimize their effects is to deploy a network of sensors, each with limited capacity, and to fusion their decisions to obtain a more reliable assessment of the alarms. This idea is not new: the usage of a set of less-than-perfect components to obtain a more robust system has been a successful strategy used in different fields, such as improving the reliability of hardware and software components, improving the robustness of a model in machine learning and data mining, and improving the accuracy decision-making systems such as those used in air-traffic control, oil exploration and radar and sonar processing (see [15]). We propose to borrow some of the successful techniques to fusion intrusion detection sensors, thereby improving the diagnosing power of the whole system.

References

[1] R. Agrawal, T. Imielinski, , and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Washington D.C., May 1993.

[2] D. Anderson and T. Frivold and A. Valdes. NIDES: A Summary. In http://www.sdl.sri.com/nides/index5.html

[3] D. Anderson and T. Lunt and H. Javitz and A. Tamaru and A. Valdes. Detecting unusual program behavior using the statistical component of the Next-generation Intrusion Detection Expert System (NIDES). Technical Report, SRI-CSL-95-06, Computer Science Laboratory, SRI International, May 1995.

[4] D. Barbará and J. Couto and S. Jajodia and N. Wu. ADAM: Detecting Intrusions by Data Mining. Proceedings of the IEEE SMC Information Assurance Workshop, West Point, NY, 2001.

[5] D. Barbará and N. Wu and S. Jajodia. Detecting Novel Network Intrusions Using Bayes Estimators. Proceedings of the First SIAM International Conference on Data Mining , April 2001, Chicago, USA.

[6] V. Barnett and T. Lewis. Outliers in Statistical Data. 3rd Edition. Wiley, 1994.

[7] Y.M.M. Bishop and S.E. Fienberg. Discrete Multivariate Analysis: Theory and Practice. The MIT Press, 1975.

[8] D.E. Denning. An Intrusion Detection Model. In *IEEE Transactions on Software Engineering*, February 1997, pp. 222-228.

[9] K. Ilgun. USTAT: A Real-Time Intrusion Detection System for UNIX. Master Thesis, University of California, Santa Barbara, November 1992.

[10] H.S. Javitz and A. Valdes, The SRI IDES Statistical Anomaly Detector. In http://www.sdl.sri.com/nides/index5.html

[11] T.F. Lunt and R Jagannathan. A Prototype Real-Time Intrusion-Detection Expert System. In *Proceedings of the IEEE Symposium on Security and Privacy,* 1988, pp. 18-21.

[12] MIT Lincoln Laboratories DARPA Intrusion Evaluation Detection. In http://www.ll.mit.edu/IST/ideval/

[13] P.A. Porras. STAT: A State Transition Analysis for Intrusion Detection. Master Thesis, Computer Science Department, University of California, Santa Barbara, 1992.

[14] P.A. Porras and P.G. Neumann EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the National Information Systems Security Conference,* 1997, pp. 353-365.

[15] H.L. Van Trees. Detection, Estimation, and Modulation Theory, Radar-Sonar Signal Processing and Gaussian Signals in Noise. John Wiley & Sons, 2001.

[16] G. Vigna and R. Kemmerer. NetStat: A Network-Based Intrusion Detection Approach. In *Proceedings of the 14t Annual Information Theory : 50 Years of Discovery Computer Security Application Conference*, Dec. 1998.

Chapter 4

A GEOMETRIC FRAMEWORK FOR UNSUPERVISED ANOMALY DETECTION

Detecting Intrusions in Unlabeled Data

Eleazar Eskin
Department of Computer Science
Columbia University
eeskin@cs.columbia.edu

Andrew Arnold
Department of Computer Science
Columbia University
aoa5@cs.columbia.edu

Michael Prerau
Department of Computer Science
Columbia University
mjp61@cs.columbia.edu

Leonid Portnoy
Department of Computer Science
Columbia University
lp178@cs.columbia.edu

Sal Stolfo
Department of Computer Science
Columbia University
sal@cs.columbia.edu

Abstract

Most current intrusion detection systems employ signature-based methods or data mining-based methods which rely on labeled training data. This training data is typically expensive to produce. We present a new geometric framework for *unsupervised anomaly detection*, which are algorithms that are designed to process unlabeled data. In our framework, data elements are mapped to a feature space which is typically a vector space $\Re^d$. Anomalies are detected by determining which points lies in sparse regions of the feature space. We present two feature maps for mapping data elements to a feature space. Our first map is a data-dependent normalization feature map which we apply to network connections. Our second feature map is a spectrum kernel which we apply to system call traces. We present three algorithms for detecting which points lie in sparse regions of the feature space. We evaluate our methods by performing experiments over network records from the KDD CUP 1999 data set and system call traces from the 1999 Lincoln Labs DARPA evaluation.

Keywords: Outlier Detection, Intrusion Detection, Kernel Functions.

1. Introduction

Intrusion detection systems (IDSs) are an integral part of any complete security package of a modern, well managed network system. The most widely deployed and commercially available methods for intrusion detection employ signature-based detection. These methods extract features from various audit streams, and detect intrusions by comparing the feature values to a set of attack signatures provided by human experts. Such methods can only detect previously known intrusions since these intrusions have a corresponding signature. The signature database has to be manually revised for each new type of attack that is discovered and until this revision, systems are vulnerable to these attacks. Because of this, there has been a lot of research in the use of data mining and machine learning algorithms which train on labeled (i.e. with instances preclassified as being an attack or not) data to detect intrusions. These approaches leverage the generalization ability of data mining methods in order to detect new attacks.

There are two major paradigms for training data mining-based intrusion detection systems: *misuse detection* and *anomaly detection*. In misuse detection approaches, each instance in a set of data is labeled as normal or intrusion and a machine learning algorithm is trained over the labeled data. For example, the MADAM/ID system Lee et al., 1999 extracts features from network connections and builds detection models over connection records that represent a summary of the traffic from a

given network connection. The detection models are generalized rules that classify the data with the values of the extracted features. These approaches have the advantage of being able to automatically retrain intrusion detection models on different input data that include new types of attacks.

Traditional anomaly detection approaches build models of normal data and detect deviations from the normal model in observed data. Anomaly detection applied to intrusion detection and computer security has been an active area of research since it was originally proposed by Denning Denning, 1987. Anomaly detection algorithms have the advantage that they can detect new types of intrusions, because these new intrusions, by assumption, will deviate from normal usage Denning, 1987; Javitz and Valdes, 1993. In this problem, given a set of normal data to train from, and given a new piece of data, the goal of the algorithm is to determine whether or not that piece of data is "normal" or is an anomaly. The notion of "normal" depends on the specific application, but without loss of generality, we can assume that this means stemming from the same distribution. We refer to this problem for reasons that will become clear below as *supervised anomaly detection.*

Most research in supervised anomaly detection can loosely be termed as performing generative modeling. These approaches build some kind of a model over the normal data and then check to see how well new data fits into that model. A survey of these techniques is given in Warrender et al., 1999. An approach for modeling normal sequences using look ahead pairs and contiguous sequences is presented in Hofmeyr et al., 1998, and a statistical method to determine sequences which occur more frequently in intrusion data as opposed to normal data is presented in Helman and Bhangoo, 1997. One approach uses a prediction model obtained by training decision trees over normal data Lee and Stolfo, 1998, while another one uses neural networks to obtain the model Ghosh and Schwartzbard, 1999. Ensemble-based approaches are presented in Fan and Stolfo, 2002. Lane and Brodley Lane and Brodley, 1997 evaluated unlabeled data for anomaly detection by looking at user profiles and comparing the activity during an intrusion to the activity during normal use. A technique developed at SRI in the EMERALD system Javitz and Valdes, 1993 uses historical records as its normal training data. It then compares distributions of new data to the distributions obtained from those historical records and differences between the distributions indicate an intrusion. Recent works such as Ye, 2000 and Eskin et al., 2001 estimate parameters of a probabilistic model over the normal data and compute how well new data fits into the model.

Supervised anomaly detection algorithms require a set of purely normal data from which they train their model. If the data contains some intrusions buried within the training data, the algorithm may not detect future instances of these attacks because it will assume that they are normal.

However, in practice, we do not have either labeled or purely normal data readily available. This makes the use of the traditional data mining-based approaches impractical. Generally, we must deal with very large volumes of audit data, and thus it is prohibitively expensive to classify it manually. We can obtain labeled data by simulating intrusions, but then we would be limited to the set of known attacks that we were able to simulate and new types of attacks occurring in the future will not be reflected in the training data. Even with manual classification, we are still limited to identifying only the known (at classification time) types of attacks, thus restricting our detection system to identifying only those types. In addition, if we collect raw data from a network environment, it is very hard to guarantee that there are no attacks during the time we are collecting the data.

Recently there has been work on a third paradigm of intrusion detection algorithms, *unsupervised anomaly detection* (also known as anomaly detection over noisy data Eskin, 2000), to address these problems Portnoy et al., 2001. These algorithms takes as input a set of unlabeled data and attempt to find intrusions buried within the data. In the unsupervised anomaly detection problem, we are given a set of data where it is unknown which are the normal elements and which are the anomalous elements. The goal is to recover the anomalous elements. Unsupervised anomaly detection is a variant of the classical outlier detection problem. After these anomalies or intrusions are detected and removed, we can then train a misuse detection algorithm or a traditional anomaly detection algorithm over the data.

In practice, unsupervised anomaly detection has many advantages over supervised anomaly detection. The main advantage is that they do not require a purely normal training set. Unsupervised anomaly detection algorithms can be performed over *unlabeled* data, which is easy to obtain since it is simply raw audit data collected from a system. In addition, unsupervised anomaly detection algorithms can be used to analyze historical data to use for forensic analysis.

In this paper, we present a geometric framework for unsupervised anomaly detection. Our frameworks maps the data to a feature space which are points in $\Re^d$. We then determine what points are outliers based on the position of the points in the feature space. We label points that are in sparse regions of the feature space as anomalies. Exactly how

we determine which points are in a sparse region of the feature space is dependent on the specific algorithm within our framework that we are using. However, in general, our algorithms will detect anomalies because they will tend to be distant from other points.

A major advantage of our framework is its flexibility. We can define our mappings to feature spaces that better capture intrusions as outliers in the feature space. We can define mappings over different types of data such as network connection records and system call traces. Once we perform the mapping to the feature space, we can apply the same algorithm to these different kinds of data. For network data, we present a data-dependent normalization feature map specifically designed for outlier detection. For system call traces, we apply a spectrum kernel feature map. Using these feature maps, we are able to process both network data which is a vector of features and system call traces which are sequences of system calls using the same algorithms.

We present three algorithms for detecting outliers in the feature space. All of the algorithms are very efficient and can deal with high dimensional data. This is required for the application of intrusion detection. Our first algorithm is a variant of the cluster-based algorithm presented in Portnoy et al. Portnoy et al., 2001. The second algorithm is a k-nearest neighbor based algorithm. The third algorithm is a Support Vector Machine-based algorithm.

We evaluated our three unsupervised anomaly detection algorithms over two types of data sets, a set of network connections and sets of system call traces. The network data the we examined was from the KDD CUP 99 data KDD99-Cup, 1999, which is a very popular and widely used intrusion attack data set. The system call data set was obtained from the 1999 Lincoln Labs DARPA intrusion detection evaluation Lippmann et al., 1999. Over both data sets, the methods show very promising results.

2. Unsupervised Anomaly Detection

In the unsupervised anomaly detection problem, we are given a large data set where most of the elements are normal, and there are intrusions buried within the data set Portnoy et al., 2001. Unsupervised anomaly detection algorithms have the major advantage of being able to process unlabeled data and detect intrusions that otherwise could not be detected. In addition, these types of algorithms can semi-automate the manual inspection of data in forensic analysis by helping analysts focus on the suspicious elements of the data.

Unsupervised anomaly detection algorithms make two assumptions about the data which motivate the general approach. The first assumption is that the number of normal instances vastly outnumbers the number of anomalies. The second assumption is that the anomalies themselves are qualitatively different from the normal instances. The basic idea is that since the anomalies are both different from normal and are rare, they will appear as outliers in the data which can be detected. An example of an intrusion that an unsupervised algorithm will have a difficulty detecting is a *syn-flood* DoS attack. The reason is that often under such an attack there are so many instances of the intrusion that it occurs in a similar number to normal instances. Our algorithms may not label these instances as an attack because the region of the feature space where they occur may be as dense as the normal regions of the feature space.

Unsupervised anomaly detection algorithms are limited to being able to detect attacks only when the assumptions hold over that data which is not always the case. For example, these algorithms will not be able to detect the malicious intent of someone who is authorized to use the network and who uses it in a seemingly legitimate way. The reason is that this intrusion is not qualitatively different from normal instances of the user. In our framework, these instances would be mapped very close to each other in the feature space and the intrusion would be undetectable.

A previous approach to unsupervised anomaly detection involves building probabilistic models from the training data and then using them to determine whether a given network data instance is an anomaly or not Eskin, 2000. In this algorithm, a mixture model for explaining the presence of anomalies is presented, and machine learning techniques are used to estimate the probability distributions of the mixture to detect the anomalies.

There is recent work in distance based outliers which is similar to our approach to unsupervised anomaly detection Knorr and Ng, 1998; Knorr and Ng, 1999; Breunig et al., 2000. These approaches examine interpoint distances between instances in the data to determine which points are outliers. A difference between these approaches and the problem of unsupervised anomaly detection is that the nature of the outliers are different. Often in network data, the same intrusion occurs multiple times which means there are many similar instances in the data. However, the number of instances of this intrusion is significantly smaller than the typical cluster of normal instances.

A problem related to unsupervised anomaly detection is the study of outliers in the field of statistics. Various techniques have been developed for detecting outliers in univariate, multivariate and structured data,

using a given probability distribution. A survey of outliers in statistics is given by Barnett and Lewis, 1994.

3. A Geometric Framework for Unsupervised Anomaly Detection

The key to our framework is mapping the records from the audit stream to a feature space. The feature space is a vector space typically of high dimension. Inside this feature space, we assume that some probability distribution generated the data. We wish to label the elements that are in low density regions of the probability distribution as anomalies. However, in general we do not know the probability distribution. Instead we label as anomalies points that are in sparse regions of the feature space. For each point, we examine the point's location within the feature space and determine whether or not it lies in a sparse region of the feature space. Exactly how we determine this depends on the algorithm that we are using.

The choice of algorithm to determine which points lie in sparse regions and the choice of the feature map is application dependent. However, critical to the practical use of these algorithms for intrusion detection is the efficiency of the algorithms. This is because the data sets in intrusion detection are typically very large.

3.1 Feature Spaces

Our data is collected from some audit stream of the system. Without loss of generality, this data is split into data elements $x_1, ..., x_l$. We define the space of all possible data elements as the *input (instance) space* X. Exactly what our input space is depends on the type of data that is being analyzed. Our input space can be the space of all possible network connection records, event logs, system call traces, etc.

We map elements of our input space to points in a *feature space* Y. In our framework a feature space is typically a real vector space of some high dimension d, $\Re^d$, or more generally a Hilbert space. The main requirement for our applications in the feature space is that we can define a dot product between elements of the feature space.

We define a *feature map* to be a function that takes as input an element in the input space and maps it to a point in the feature space. In general, we use ϕ to define a feature map and we get

$$\phi : X \to Y \quad . \tag{4.1}$$

We use the term *image* of a data element x to denote the point in the feature space $\phi(x)$.

Since our feature space is a Hilbert space, for any points y_1 and y_2 their dot product $< y_1, y_2 >$ is defined. Any time we have a dot product, we can also define a norm on the space as well as a distance.

The norm of a point y in the feature space $||y||$ is simply the square root of the dot product of the point with itself, $||y|| = \sqrt{< y, y >}$. Using this and the fact that a dot product is a symmetric bilinear form we can define the distance between two elements of the feature space y_1 and y_2 with

$$\begin{aligned} ||y_1 - y_2|| &= \sqrt{< y_1 - y_2, y_1 - y_2 >} \\ &= \sqrt{< y_1, y_1 > -2 < y_1, y_2 > + < y_2, y_2 >} \quad . \end{aligned}$$

Using our framework, we can use the feature map to define relations between elements of the input space. Given two elements in the input space x_1 and x_2, we can use the feature map to define a distance between the two elements as the distance between their corresponding images in the feature space. We can define our distance function d_ϕ as

$$\begin{aligned} & d_\phi(x_1, x_2) \\ = \; & ||\phi(x_1) - \phi(x_2)|| \qquad (4.2) \\ = \; & \sqrt{< \phi(x_1), \phi(x_1) > -2 < \phi(x_1), \phi(x_2) > + < \phi(x_2), \phi(x_2) >} \quad . \end{aligned}$$

For notational convenience, we often drop the subscript from d_ϕ.

If the feature space is $\Re^d$, this distance corresponds to standard Euclidean distance in that space.

3.2 Kernel Functions

In many cases, it is difficult to explicitly map a data instance to a point in its feature space. One reason is that the feature space has a very high dimension which makes it difficult to explicitly store the points in the feature space because of memory considerations. In some cases, the explicit map may be very difficult to determine.

However, for our purposes, we are interested in the dot products of points in the feature space. If have a mechanism for computing the dot product of the images of two data elements, we do not need to explicitly map the data elements to their images.

We can define a *kernel function* to compute these dot products in the feature space. A kernel function is defined over a pair of elements in the feature space and returns the dot product between the images of those elements in the feature space. More formally

$$K_\phi(x_1, x_2) =< \phi(x_1), \phi(x_2) > \quad . \qquad (4.3)$$

We can redefine our distance measure (4.1) through a kernel function as

$$d_\phi(x_1, x_2) = \sqrt{K_\phi(x_1, x_1) - 2K_\phi(x_1, x_2) + K_\phi(x_2, x_2)} \quad . \tag{4.4}$$

In many cases, the kernel function can be computed efficiently without explicitly mapping the elements from the input space to their images. A function is a kernel function if there exists a feature space which is a Hilbert space and for which the kernel function corresponds to a dot product. There are conditions on whether or not a function is a kernel which are described in detail in Cristianini and Shawe-Taylor, 2000.

An example of a kernel that performs the mapping implicitly is the radial basis kernel. The radial basis kernel is a function of the form

$$K_{\mathrm{rb}}(x_1, x_2) = e^{-\frac{||x_1 - x_2||^2}{\sigma^2}} \quad . \tag{4.5}$$

The radial basis kernel corresponds to an infinite dimensional feature space Cristianini and Shawe-Taylor, 2000.

In addition to the computational advantages of kernels, we can define kernels to take advantage of our intuitions about the application. We can easily weight various features higher or lower depending on domain knowledge.

3.3 Convolution Kernels

Although the examples of kernels we have given up to this point have been defined over input spaces which are vector spaces, we can define kernels over arbitrary input spaces. These kinds of kernels are referred to as *convolution kernels* Haussler, 1999; Watkins, 2000.

In this framework, we can define our kernels directly over our audit data without needing to first convert our data into a vector in $\Re^n$. In addition, since kernels can be defined on not only numerical features, but other types of structures such as sequences, we can define kernels to handle many different types data such as sequences of system calls and event logs. This allows us to handle different kinds of data in a consistent framework using different kernels but using the same algorithms which are defined in terms of kernels.

4. Detecting Outliers in Feature Spaces

After mapping our data to points in the feature space, we can more easily formalize the problem of unsupervised anomaly detection. The main idea is that we want to detect points that are distant from most

other points or in relatively sparse regions of the feature space. This is similar to the problem of outlier detection.

We present three algorithms for detecting anomalies in the feature space. All of the algorithms can be implemented in terms of dot products of the input elements which allows us to use kernel functions to perform implicit mappings to the feature space. Each algorithm detects points that lie in sparse regions in a different way.

The first algorithm is a variant of the cluster-based algorithm presented in Portnoy et al., 2001. For each point, the algorithm approximates the density of points near the given point. The algorithm makes this approximation by counting the number of points that are within a sphere of radius w around the point. Points that are in a dense region of the feature space and contain many points within the ball are considered normal. Points that are in a sparse region of the feature space and contain few points within the ball are considered anomalies. We implement an efficient approximation to this algorithm. We first perform fixed-width clustering over the points with a radius of w. We then sort the clusters based on the size. The points in the small clusters are labeled anomalous. We present details of the efficient algorithm below.

The second algorithm detects anomalies based on computing the k-nearest neighbors of each point. If the sum of the distances to the k-nearest neighbors is greater than a threshold, we consider the point an anomaly. We present below an efficient algorithm to detect outliers which uses a fixed-width clustering algorithm to significantly speed up the computation of the k-nearest neighbors.

The third algorithm is a support vector machine based algorithm that identifies low support regions of a probability distribution by solving a convex optimization problem Schölkopf et al., 1999. The points in the feature space are further mapped into another feature space using a Gaussian kernel. In this second feature space, a hyperplane is drawn to separate the majority of the points away from the origin. The remaining points represent the outliers or anomalies.

Each of these three algorithms is optimizing a clearly defined objective function over points in the feature space. We can present efficient algorithms to solve these problems given their formulation.

5. Algorithm 1: Cluster-based Estimation

The goal for our first algorithm is to compute how many points are "near" each point in the feature space. A parameter to the algorithm is a radius w also referred to as the cluster width. For any pair of points x_1 and x_2 we consider the two points "near" each other if their distance

is less that or equal to w, $d(x_1, x_2) \leq w$ with distance defined as in equation (4.1).

For each point x, we define $N(x)$ to be the number of points that are within w of point x. More formally we define

$$N(x) = |\{s|d(x,s) \leq w\}| \quad . \tag{4.6}$$

The straightforward computation of $N(x)$ for all points has a complexity of $O(n^2)$ where n is the number of points. The reason is that we have to compute the pairwise distances between all points.

However, since we are really only interested in determining what the outliers are, we can effectively approximate the computation as follows.

We first perform fixed-width clustering over the entire data set with cluster width w. Then we label the points in the small clusters as anomalies.

A fixed width clustering algorithm is as follows. The first point is center of the first cluster. For every subsequent point, if it is within w of a cluster center, it is a added to that cluster. Otherwise it is a center of a new cluster. Note that some points may be added to multiple clusters. The fixed width clustering algorithm requires only one pass through the data. The complexity of the algorithm is $O(cn)$ where c is the number of clusters and n is the number of data points. For a reasonable w, c will be significantly smaller than n.

Note that by the definition in (4.6), for each cluster, the number of points near the cluster center, $N(c)$, is the number of points in the cluster c. For each point x, not a center of a cluster, we approximate $N(x)$, by $N(c)$ for the cluster c that contains x. For points in very dense regions where there is a lot of overlap between clusters, this will be an inaccurate estimate. However, for points that are outliers, there will be relatively few overlapping clusters in these regions and $N(c)$ will be an accurate approximation of $N(x)$. Since we are only interested in the points that are outliers, the points in the dense regions will be higher than the threshold anyway. Thus the approximation is reasonable in our case.

With the efficient approximation algorithm, we are able to process significantly larger data sets then possible with the straightforward algorithm because we do not need to perform a pairwise comparison of points.

6. Algorithm 2: K-nearest neighbor

Our second algorithm determines whether or not a point lies in a sparse region of the feature space by computing the sum of the the dis-

tances to the k-nearest neighbors of the point. We refer to this quantity as the k-NN score for a point.

Intuitively, the points in dense regions will have many points near them and will have a small k-NN score. If the size of k exceeds the frequency of any given attack type in the data set and the images of the attack elements are far from the images of the normal elements, then the k-NN score is useful for detecting these attacks.

The main problem with computing the k-NN score is that it is computationally expensive to compute the k-nearest neighbors of each point. The complexity of this computation is $O(n^2)$ which is impractical for intrusion detection applications.

Since we are interested in only the k-nearest points to a given point, we can significantly speed up the algorithm by using a technique similar to canopy clustering McCallum et al., 2000. Canopy clustering is used as a means of breaking down the space into smaller subsets so as to remove the necessity of checking every data point. We use the clusters as a tool to reduce the time of finding the k-nearest neighbors.

We first cluster the data using the fixed-width clustering algorithm of the previous section with a variation where we place each element into only one cluster. Once the data is clustered with width w, we can compute the k-nearest neighbors for a given point x by taking advantage of the following properties.

We denote as $c(x)$ the point which is the center of the cluster that contains a given point x. For a cluster c and a point x we use the notation $d(x, c)$ to denote the distance between the point and the cluster center. For any two points x_1 and x_2, if the points are in the same cluster

$$d_\phi(x_1, x_2) \leq 2w \tag{4.7}$$

and in all cases

$$d_\phi(x_1, x_2) \leq d_\phi(x_1, c(x_2)) + w \tag{4.8}$$

$$d_\phi(x_1, x_2) \geq d_\phi(x_1, c(x_2)) - w \quad . \tag{4.9}$$

The algorithm uses these three inequalities to determine the k-nearest neighbors of a point x.

Let C be a set of clusters. Initially C contains all of the clusters in the data. At any step in the algorithm, we have a set of points which are potentially among the k-nearest neighbor points. We denote this set P. We also have a set of points that are in fact among the k-nearest neighbor points. We denote this set K. Initially K and P are empty. We precompute the distance from x to each cluster. For the cluster with center closest to x, we remove it from C and add all of its points

to P. We refer to this operation as "opening" the cluster. The key to the algorithm is that we can obtain a lower bound the distance from all points in the clusters in set C using equation (4.9). We define

$$d_{\min} = \min_{c \in C} d(x, c) - w \quad . \tag{4.10}$$

The algorithm performs the following. For each point in $x_i \in P$, we compute $d(x, x_i)$. If $d(x, x_i) < d_{\min}$, we can guarantee that x_i is closer point to x then all of the points in the clusters in C. In this case, we remove x_i from P and add it to K. If we can not guarantee this for any element of P (including the case that if P is empty), then we "open" the closest cluster by adding all of its points to P and remove that cluster from C. Notice that when we remove the cluster from C, $d_{\min}$ will increase. Once K has k elements, we terminate.

Most of the computation is spent checking the distance between points in D to the cluster centers. This is significantly more efficient than computing the pairwise distances between all points.

The choice of width w does not affect the k-NN score, but instead only affects the efficiency of computing the score. Intuitively, we want to choose a w that splits the data into reasonably sized clusters.

7. Algorithm 3: One Class SVM

Our third algorithm uses an algorithm presented in Schölkopf et al., 1999 to estimate the region of the feature space where most of the data occurs. We first map our feature space into a second feature space with a radial basis kernel, equation (4.5), and then work in the new feature space.

The standard SVM algorithm is a supervised learning algorithm. It requires labeled training data to create its classification rule. In Schölkopf et al., 1999, the SVM algorithm was adapted into an unsupervised learning algorithm. This unsupervised variant does not require its training set to be labeled to determine a decision surface.

Whereas the supervised version of SVM tries to maximally separate two classes of data in feature space by a hyperplane, the unsupervised version instead attempts to separate the entire set of training data from the origin. This is done by solving a quadratic program that penalizes any points not separated from the origin while simultaneously trying to maximize the distance of this hyperplane from the origin. At the end of this optimization, this hyperplane then acts as our decision function, with those points that it separates from the origin classified as normal, and those which are on the other side of the hyperplane, we classify as anomalous.

The algorithm is similar to the standard SVM algorithm in that it uses kernel functions to perform implicit mappings and dot products. It also uses the same kind of hyperplane for the decision surface. The solution is only dependent on the support vectors as well. However, the support vectors are determined in a different way. This algorithm attempts to find a small region where most of the data lies and label points in that region as class $+1$. Points in other regions are labeled as class -1. The main idea is that the algorithm attempts to find the hyperplane that separates the data points from the origin with *maximal margin*. The decision surface that is chosen is determined by solving an optimization problem that determines the "best" hyperplane under a set of criteria which is beyond the scope of this paper and described fully in Cristianini and Shawe-Taylor, 2000.

The specific optimization that is solved for estimating the hyperplane specified by the hyperplane's normal vector in the feature space w and offset from the origin ρ is

$$\min_{w \in Y, \zeta_i \in \Re, \rho \in \Re} \quad \tfrac{1}{2}||w||^2 + \tfrac{1}{vl}\sum_i^l \zeta_i - \rho \tag{4.11}$$

$$\text{subject to:} \quad (w \cdot \phi(x_i)) \geq \rho - \zeta_i, \zeta_i \geq 0 \tag{4.12}$$

where $0 < v < 1$ is a parameter that controls the trade off between maximizing the distance from the origin and containing most of the data in the region created by the hyperplane and corresponds to the ratio of expected anomalies in the data set. ζ_i are slack variables that penalize the objective function but allow some of the points to be on the other wrong side of the hyperplane.

After we solve the optimization problem, the decision function for each point x is

$$f(x) = \text{sgn}((w \cdot \phi(x)) - \rho) \quad . \tag{4.13}$$

If we introduce a Lagrangian and rewrite this optimization in terms of the Lagrange multipliers α_i we can represent the optimization as

$$\text{minimize:} \ \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j K_\phi(x_i, x_j)$$

$$\text{subject to:} \ 0 \leq \alpha_i \leq \frac{1}{vl}, \sum_i \alpha_i = 1$$

at the optimum, ρ can be computed from the Lagrange multipliers for any x_i such that the corresponding Lagrange multiplier α_i satisfies $0 < \alpha_i < \frac{1}{vl}$

$$\rho = \sum_j \alpha_j K_\phi(x_j, x_i)$$

In terms of the Lagrange multipliers, the decision function is

$$f(x) = \text{sgn}\left(\sum_i \alpha_i K_\phi(x_i, x) - \rho\right) \quad . \tag{4.14}$$

One property of the optimization is that for the majority of the data points, α_i will be 0 which makes the decision function efficient to compute.

The optimization is solved with a variant of the Sequential Minimal Optimization algorithm Platt, 1999. Details on the optimization, the theory behind the relation of this algorithm and the estimation of the probability density of the original feature space, and details of the algorithm are fully described in Schölkopf et al., 1999.

8. Feature Spaces for Intrusion Detection

The choice of feature space for unsupervised anomaly detection is application specific. The performance greatly depends on the ability of the feature space to capture information relevant to the application. For optimal performance, it is best to analyze the specific application and choose a feature space accordingly.

In our experiments, we analyze two data sets. The first data set is a set of network connection records. This data set contains records which contain 41 features describing a network connection. The second data set is a set of system call traces. Each entry is a sequence of all of the system calls that a specific process makes during its execution.

We use two different feature maps for the different kinds of data that we analyze. For data which are records, we use a *data-dependent normalization kernel* to implicitly define the feature map. This feature map takes into account how abnormal a specific feature in a record is when it performs the mapping. We have found that this significantly improves detection performance.

For system call data where each trace is a sequence of system calls, we apply a string kernel over these sequences. The kernel we use is called a *spectrum kernel* which was previously used to analyze biological sequences Christina Leslie and Noble, 2002. The spectrum kernel maps short sub-sequences of the string into the feature space. This is consistent with our intuitions for intrusion detection because short sub-sequences have been the primary basis for analysis of system call

sequences previously Forrest et al., 1996; Lee et al., 1997; Eskin et al., 2001.

8.1 Data-dependent Normalization Kernels

For data which is a network connection record, we use a data-dependent normalization feature map. This feature map takes into account the variability of each feature in the mapping in such a way that normalizes the relative distances between feature values in the feature space.

There are two types of attributes in a connection record. There are either numerical attributes or discrete attributes. Examples of numerical attributes in connection records are the number of bytes in a connection or the amount of connections to the same port. An example of discrete attributes in network connection records is the type of protocol used for the connection. Even some attributes that are numerical are in fact discrete values, such as the destination port of a connection. We handle discrete and numerical attributes differently in our kernel mapping.

One problem with the straightforward mapping of the numerical attributes is that they are all on different scales. If a certain attribute is a hundred times larger than another attribute, it will dominate the second attribute.

We normalize all of our attributes to the number of standard deviations away from the mean. This scales our distances based on the likelihood of the attribute values. This feature map is data dependent because the distance between two points depends on the mean and standard deviation of of the attributes which in turn depends on the distribution of attribute values over all of the data.

For discrete values, we use a similar data dependent idea. Let Σ_i be the set of possible values for discrete attribute i. For each discrete attribute we have $|\Sigma_i|$ coordinates in the feature space corresponding to this attribute. There is one coordinate for every possible value of the attribute. A specific value of the attribute gets mapped to the feature space as follows. The coordinate corresponding to the attribute value has a positive value $\frac{1}{|\Sigma_i|}$ and the remaining coordinates corresponding to the feature have a value of 0. The distance between two vectors is weighted by the size of the range of values of the discrete attributes. A different value for attribute i between two records will contribute $\frac{2}{|\Sigma_i|^2}$ to the square of the norm between the two vectors.

8.2 Kernels for Sequences: The Spectrum Kernel

Convolution kernels can be defined over arbitrary input spaces. We use a kernel defined over sequences to model sequences of system calls. The specific kernel we use is a *spectrum kernel* which has been successfully applied to modeling biological sequences Christina Leslie and Noble, 2002.

The spectrum kernel is defined over an input space of sequences. These sequences can be an arbitrary long sequence of elements from an alphabet Σ. For any $k > 0$, we define the feature space of the k-spectrum kernel as follows. The feature space is a $|\Sigma|^k$ dimensional space where each coordinate corresponds to a specific k length sub-sequence. For a given sequence, the value of a specific coordinate of the feature space is the count of the number of times the corresponding sub-sequence occurs in the sequence. These sub-sequences are extracted from the sequence by using a sliding window of length k.

The dimension of the feature space is exponential in k which makes it impractical to store the feature space explicitly. Note that the feature vectors corresponding to a sequence are extremely sparse. We can take advantage of this fact and efficiently compute the kernels between sequences using an efficient data structure as described in Christina Leslie and Noble, 2002. This is critical because in one of our experiments we consider 26 possible system calls and sub-sequences of length 4 which gives a dimension of the feature space of close to $500,000$.

9. Experiments

We perform experiments over two different types of data. We analyze network connection records, and system call traces.

9.1 Performance measures

To evaluate our system we were interested in two major indicators of performance: the *detection rate* and the *false positive* rate. The detection rate is defined as the number of intrusion instances detected by the system divided by the total number of intrusion instances present in the test set. The false positive rate is defined as the total number of normal instances that were (incorrectly) classified as intrusions divided by the total number of normal instances. These are good indicators of performance, since they measure what percentage of intrusions the sys-

tem is able to detect and how many incorrect classifications it makes in the process. We calculate these values over the labeled data to measure performance.

The trade-off between the false positive and detection rates is inherently present in many machine learning methods. By comparing these quantities against each other we can evaluate the performance invariant of the bias in the distribution of labels in the data. This is especially important in intrusion detection problems because the normal data outnumbers the intrusion data by a factor of 100 : 1. The classical accuracy measure is misleading because a system that always classifies all data as normal would have a 99% accuracy.

We plot ROC (Receiver Operating Characteristic) Provost et al., 1998 curves depicting the relationship between false positive and detection rates for one fixed training/test set combination. ROC curves are a way of visualizing the trade-offs between detection and false positive rates.

9.2 Data Set Descriptions

The network connection records we used was the KDD Cup 1999 Data KDD99-Cup, 1999, which contained a wide variety of intrusions simulated in a military network environment. It consisted of approximately 4,900,000 data instances, each of which is a vector of extracted feature values from a connection record obtained from the raw network data gathered during the simulated intrusions. A connection is a sequence of TCP packets to and from some IP addresses. The TCP packers were assembled into connection records using the Bro program Paxson, 1998 modified for use with MADAM/ID Lee et al., 1999. Each connection was labeled as either normal or as exactly one specific kind of attack. All labels are assumed to be correct.

The simulated attacks fell in one of the following four categories : DOS - Denial of Service (e.g. a syn flood), R2L - Unauthorized access from a remote machine (e.g. password guessing), U2R - unauthorized access to superuser or root functions (e.g. a buffer overflow attack), and Probing - surveillance and other probing for vulnerabilities (e.g. port scanning). There were a total of 24 attack types.

The extracted features included the basic features of an individual TCP connection such as its duration, protocol type, number of bytes transferred, and the flag indicating the normal or error status of the connection. Other features of an individual connection were obtained using some domain knowledge, and included the number of file creation operations, number of failed login attempts, whether root shell was obtained, and others. Finally, there were a number of features computed

Table 4.1. Lincoln Labs Data Summary

Program Name	Total # of Attacks	# Intrusion Traces	# Intrusion System Calls	# Normal Traces	# Normal System Calls	% Intrusion Traces
ps	3	21	996	208	35092	2.7%
eject	3	6	726	7	1278	36.3%

using a two-second time window. These included - the number of connections to the same host as the current connection within the past two seconds, percent of connections that have "SYN" and "REJ" errors, and the number of connections to the same service as the current connection within the past two seconds. In total, there are 41 features, with most of them taking on continuous values.

The KDD data set was obtained by simulating a large number of different types of attacks, with normal activity in the background. The goal was to produce a good training set for learning methods that use labeled data. As a result, the proportion of attack instances to normal ones in the KDD training data set is very large as compared to data that we would expect to observe in practice.

Unsupervised anomaly detection algorithms are sensitive to the ratio of intrusions in the data set. If the number of intrusions is too high, each intrusion will not show up as anomalous. In order to make the data set more realistic we filtered many of the attacks so that the resulting data set consisted of 1 to 1.5% attack and 98.5 to 99% normal instances.

The system call data is from the BSM (Basic Security Module) data portion of the 1999 DARPA Intrusion Detection Evaluation data created by MIT Lincoln Labs Lippmann et al., 1999. The data consists of 5 weeks of BSM data of all processes run on a Solaris machine. We examined three weeks of traces of the programs which were attacked during that time. The programs we examined were *eject*, and *ps*.

Each of the attacks that occurred correspond to one or more process traces. An attack can correspond to multiple process traces because a malicious process can spawn other processes. We consider the attack detected if one of the processes that correspond to the attack is detected.

Tables 4.1 summarize the system call trace data sets and list the number of system calls and traces for each program.

9.3 Experimental Setup

For each of our data sets, we split the data into two portions. One portion, the training set, was used to set parameters values for our algorithms and the second, the test set, was used for evaluation.

We set parameters based on the training set. Then for each of the methods over each of the data sets, we varied the detection threshold and at each threshold computed the detection rate and false positive rate. For each algorithm over each data set we obtained a ROC curve.

Our parameter settings are as follows. For the cluster-based algorithm presented in Section 5, when processing the network connection data, we set the width of the fixed-width clustering to be 40 in the feature space. For the *eject* system call traces, the width was set to be 5. For the *ps* traces, the width was set to be 10.

For the k-nearest neighbor-based algorithm presented in Section 6, for the KDD cup data, the value of k was set to $10,000$ of the data set. For the *eject* data set, $k = 2$ and for the *ps* data set, $k = 15$. The k is adjusted to the overall size of the data.

For the SVM-based algorithm in Section 7, for the KDD cup data, we set $v = .01$ and $\sigma^2 = 12$. For the system call data sets, we used a value of $v = .05$ and $\sigma^2 = 1$.

9.4 Experimental Results

Our approach to unsupervised anomaly detection performed very well over both types of data.

In the case of the system call data, the each of the algorithms performed perfectly. What this means is that at a certain threshold, there was at least one process trace from each of the attacks identified as being malicious without any false positives. An explanation for these results can be obtained by looking at exactly what the feature space is encoding. Each system call trace is mapped to a feature space using the spectrum kernel that contains a coordinate for each possible sub-sequence. Process traces that contain many of the same sub-sequences of system calls are closer together than process traces that contain fewer sub-sequences of system calls.

The results are not surprising when we consider previous approaches to anomaly detection over system call traces. In the original work in this area Forrest et al., 1996, sub-sequences of system calls were the basis of the detection algorithm. The supervised anomaly detection algorithm presented in Forrest et al., 1996 recorded a set of sub-sequences that occurred in a normal training set and then detected anomalies in a test set by counting the number of unknown sub-sequences that oc-

cur within a window of 20 consecutive sub-sequences. If the number of previously unseen sub-sequences is above a threshold, then the method would determine that the process corresponds to an intrusion. The results of their experiments suggest that there a many sub-sequences that occur in malicious processes that do not occur in normal processes. This explains why in our feature space defined by the spectrum kernel, the intrusion processes are significantly far away from the normal processes. Also, in our experiments, the normal processes clumped together in the feature space. This is why the intrusion processes were easily detected as outliers.

For the network connections, the data is not nearly as regular as the system call traces. From our experiments, we found that there were some types of attacks that we were able to detect well and other types of attacks that we were not able to detect. This is reasonable because some of the attacks using our feature space were in the same region as normal data. Although the detection rates are lower than what is typically obtained for either misuse or supervised anomaly detection, the problem of unsupervised anomaly detection is significantly harder because we do not have access to the labels or have a guaranteed clean training set.

Figure 4.1 shows the performance of the three algorithms over the KDD Cup 1999 data. Table 4.2 shows the Detection Rate and False Positive Rate for some selected points from the ROC curves. All three algorithms perform relatively close to each other.

10. Discussion

This paper presents a geometric framework for unsupervised anomaly detection. The framework maps data elements to a feature space and then detects anomalies by determining which points lie in sparse regions of the feature space. We presented two feature maps. The first feature map is a data-dependent normalization feature map which is applied to network connection records. The second feature map is a spectrum kernel which is applied to system call traces. We also presented three algorithms for detecting points in the sparse regions of the feature space. The first algorithm is a cluster-based approach which is a variant of Portnoy et al., 2001. The second algorithm is a k-nearest neighbor-based approach. The third algorithm is a SVM-based approach. Each of these algorithms can be applied to any feature space which when combined with the different feature maps, allows us to apply the same algorithms to model different kinds of data.

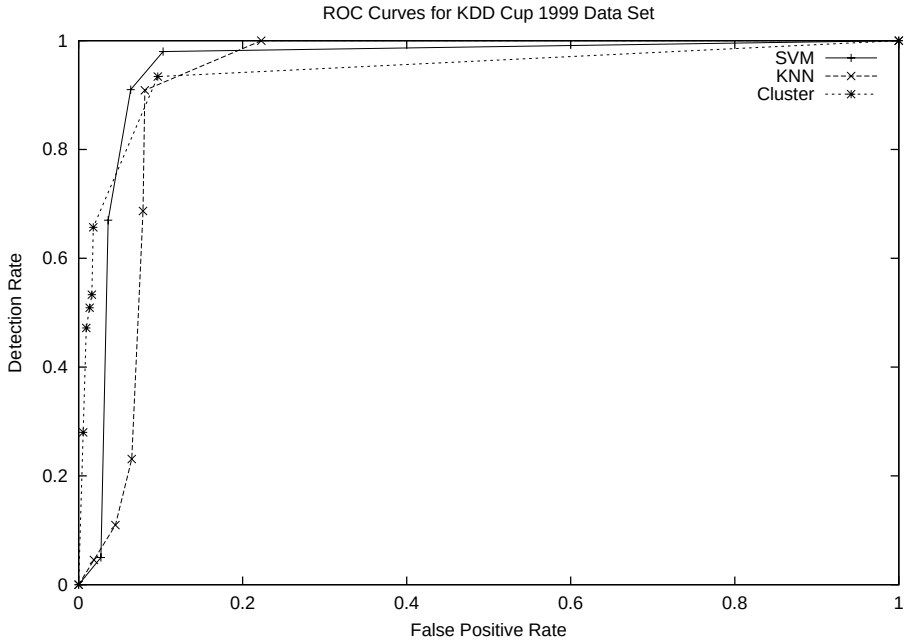


Figure 4.1. ROC curves showing the performance of the 3 algorithms over the KDD data set. The curves are obtained by varying the threshold.

We evaluated our methods over both a collection of network records and a set of system call traces. In both cases, our algorithms were able to detect intrusions over the unlabeled data. In practice, detecting intrusions over unlabeled data is important, because these algorithms can be applied to raw collected system data and do not need to be manually labeled which can be an expensive process.

Future work involves defining more feature maps over different kinds of data and performing more extensive experiments evaluating the methods presented in this paper.

References

Barnett, V. and Lewis, T. (1994). *Outliers in Statistical Data*. John Wiley and Sons.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). LOF: identifying density-based local outliers. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 93–104.

Table 4.2. Selected points from the ROC curves of the performance of each algorithm over the KDD Cup 1999 Data.

Algorithm	Detection rate	False positive rate
Cluster	93%	10%
Cluster	66%	2%
Cluster	47%	1%
Cluster	28%	.5%
K-NN	91%	8%
K-NN	23%	6%
K-NN	11%	4%
K-NN	5%	2%
SVM	98%	10%
SVM	91%	6%
SVM	67%	4%
SVM	5%	3%

Christina Leslie, E. E. and Noble, W. S. (2002). The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing (PSB-2002)*, Kaua'i, Hawaii.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK.

Denning, D. (1987). An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222–232.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the International Conference on Machine Learning*.

Eskin, E., Lee, W., and Stolfo, S. J. (2001). Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX II)*, Anaheim, CA.

Fan, W. and Stolfo, S. (2002). Ensemble-based adaptive intrusion detection. In *Proceedings of 2002 SIAM International Conference on Data Mining*, Arlington, VA.

Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for unix processes. In *1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society.

Ghosh, A. and Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium*.

Haussler, D. (1999). Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz.

Helman, P. and Bhangoo, J. (1997). A statistically base system for prioritizing information exploration under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(4):449–466.

Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion detect using sequences of system calls. *Journal of Computer Security*, 6:151–180.

Javitz, H. S. and Valdes, A. (1993). The NIDES statistical component: description and justification. In *Technical Report, Computer Science Laboratory, SRI International*.

KDD99-Cup (1999). The third international knowledge discovery and data mining tools competition dataset `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

Knorr, E. M. and Ng, R. T. (1998). Algorithms for mining distance-based outliers in large datasets. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 392–403.

Knorr, E. M. and Ng, R. T. (1999). Finding intentional knowledge of distance-based outliers. *The VLDB Journal*, pages 211–222.

Lane, T. and Brodley, C. E. (1997). Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49. AAAI Press.

Lee, W. and Stolfo, S. J. (1998). Data mining approaches for intrusion detection. In *Proceedings of the 1998 USENIX Security Symposium.*

Lee, W., Stolfo, S. J., and Chan, P. K. (1997). Learning patterns from unix processes execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. AAAI Press.

Lee, W., Stolfo, S. J., and Mok, K. (1999). Data mining in work flow environments: Experiences in intrusion detection. In *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99).*

Lippmann, R. P., Cunningham, R. K., Fried, D. J., Graf, I., Kendall, K. R., Webster, S. W., and Zissman, M. (1999). Results of the 1999 darpa off-line intrusion detection evaluation. In *Second International Workshop on Recent Advances in Intrusion Detection (RAID 1999)*, West Lafayette, IN.

McCallum, A., Nigam, K., and Ungar, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178.

Paxson, V. (1998). Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX.

Platt, J. (1999). Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA. MIT Press.

Portnoy, L., Eskin, E., and Stolfo, S. J. (2001). Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, Philadelphia, PA.

Provost, F., Fawcett, T., and Kohavi, R. (1998). The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*.

Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (1999). Estimating the support of a high-dimensional distribution. Technical Report 99-87, Microsoft Research. To appear in *Neural Computation*, 2001.

Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting intrusions using system calls: alternative data models. In *1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society.

Watkins, C. (2000). Dynamic alignment kernels. In Smola, A., Bartlett, P., Schölkopf, B., and Schuurmans, D., editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA. MIT Press.

Ye, N. (2000). A markov chain model of temporal behavior for anomaly detection,. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*.

Chapter 5

FUSING A HETEROGENEOUS ALERT STREAM INTO SCENARIOS *

Oliver Dain
Massachusetts Institute of Technology
Lincoln Laboratory
odain@ll.mit.edu

Robert K. Cunningham
Massachusetts Institute of Technology
Lincoln Laboratory
rkc@ll.mit.edu

Abstract An algorithm for fusing the alerts produced by multiple heterogeneous intrusion detection systems is presented. The algorithm runs in real-time, combining the alerts into scenarios; each is composed of a sequence of alerts produced by a single actor or organization. The software is capable of discovering scenarios even if stealthy attack methods, such as forged IP addresses or long attack latencies, are employed. The algorithm generates scenarios by estimating the probability that a new alert belongs to a given scenario. Alerts are then added to the most likely candidate scenario. Two alternative probability estimation techniques are compared to an algorithm that builds scenarios using a set of rules. Both probability estimate approaches make use of training data to learn the appropriate probability measures. Our algorithm can determine the scenario membership of a new alert in time proportional to the number of candidate scenarios.

Keywords: security, fusion, scenarios, correlation, intrusion detection

*This work was sponsored by the Department of Defense under Air Force contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

1. Introduction

Many security-conscious organizations protect their networks with multiple intrusion detection systems (IDSs). While this does improve attack detection rates, it also increases the work load on intrusion detection analysts; there are now more components to be monitored and false alarm rates tend to rise. Fusion systems attempt to alleviate these problems by collecting the alerts produced by all the systems on the network. The collected alerts can then be managed from a single location.

In this paper we propose a realtime algorithm to combine the alerts produced by multiple intrusion detection systems into scenarios. Each scenario is intended to indicate a sequence of actions performed by a single actor or organization. It is important to realize that a scenario constructed by this algorithm does not necessarily indicate malicious behavior. The purpose of the scenarios is simply to group alerts that share a common cause. The resulting scenarios give network defenders a more complete picture of the traffic on their network. Additionally we can assign false alarm probabilities to whole scenarios rather than individual alerts, and thus decrease false positive rates while increasing the sensitivity of the underlying sensors.

Each time a new alert is produced the algorithm must compare it to scenarios that have already been constructed. For each existing scenario the probability that the new alert belongs to this scenario must be calculated. We compare two techniques for producing the required probability estimates. Both approaches use training data to learn the appropriate probability measures. These techniques are compared to a naïve algorithm that builds scenarios using a set of hand coded rules. The algorithm runs in time proportional to the number of existing scenarios, and is capable of finding scenarios even if an intruder used stealthy attack methods such as forged source IP addresses or long latencies between attack components.

As Edward Amoroso points out, very little research has been done on correlation or fusion[2]. Much of the work in this area has focused on collecting alerts from multiple detectors at a single location where they can be displayed, queried, and correlated[17]. The GrIDS system developed at University of California, Davis uses rule sets to combine alerts and network data into a graph structure capable of discovering large scale coordinated attacks[15]. Clifton and Gengo use data mining techniques to discover common sequences of alerts[4]. These alert sequences can, at the discretion of human operators, be filtered so they are no longer displayed to an analyst. The discovered sequences are not intended to be scenarios; they are simply sequences that appear frequently. Valdes

and Skinner have studied correlation and scenario building from probabilistic detectors such as their EMERALD eBayes system[16]. Each alert and meta alert (scenario) in their system is represented by a vector of attributes. A similarity measure is applied to the vectors and those that exceed a threshold are joined together into a scenario. Our approach is similar to the one proposed by Valdes and Skinner but we use a different scenario construction algorithm and optimize our probability estimates so that the system forms scenarios like a human expert. Furthermore we have tested our system using output from both commercial and proprietary intrusion detection systems.

2. Fusion Approach

The fusion system develops scenarios as alerts are received from the component Intrusion Detection Systems. Thus, our task at any given instant is to determine the scenario membership of a new alert given the alerts we have already seen. The most flexible way to do this is to consider all possible ways of combining the alerts into scenarios each time a new alert is generated. Unfortunately this quickly becomes computationally intractable. To see this, consider the simpler problem of deciding which of n existing alerts belong in a scenario with a given new alert. Figure 5.1a shows possible scenario assignments for a new alert, C, given two existing alerts. We note that there are 4 possibilities. Figure 5.1b shows the 8 possible scenario assignments for a new alert, D, given three existing alerts. We note that we could add alert D to any of the scenarios enumerated in Figure 5.1a. Additionally D could be grouped with any of these scenarios with C removed. Thus the addition of each alert doubles the number of possible scenario constructions so that a new alert may be grouped with n existing alerts in 2^n ways. Since our task is not simply to construct a scenario in which to place a single new alert but rather to construct scenarios for all alerts the complexity of the problem is super-exponential. Clearly it is not possible to consider all such arrangements in realtime.

Our solution is to use an "atom model". Each time we receive a new alert from a sensor we compare it to the scenarios we have constructed thus far. For each existing scenario we calculate the probability that the new alert belongs to this scenario. The new alert is then assigned to the scenario that produced the highest probability score. If all the scores are below a threshold the new alert is not joined to any scenario and instead starts a new scenario. Once the alert has joined a scenario we do not consider changing this assignment. Hence the scenario is not divisible and is "an atom" - it may absorb other alerts and grow, but it can

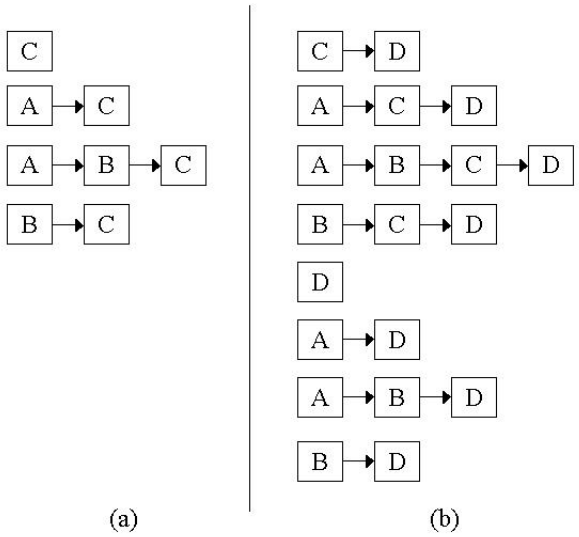


Figure 5.1. (a) Possible scenario assignments for a new alert, C, given 2 existing alerts, A, and B. (b) Possible scenario assignments for a new alert, D, given 3 existing alerts, A, B, and C.

not be split. Although it may cause errors in the assembled scenarios, this simplification reduces the computational complexity from super-exponential in the number of alerts to linear in the number of candidate scenarios. Despite this simplification the system still performs well (see Section 6). Informal testing indicates that the system is able to construct scenarios from over 16,000 alerts in less than two seconds.

3. Architecture

The fusion system consists of three distinct components: the IDSs that produce the alerts, a database where the alerts are stored and the fusion software itself. The IDSs currently used by the fusion system are Internet Security System's RealSecure network sensor[11] and MIT Lincoln Laboratory's Battlefield Intrusion Detection System (BIDS), a host based system capable of finding known and novel attacks[6]. The alerts produced by these IDSs are converted to a standard form and written to a SQL database. The fusion system reads from the database, determines the scenario membership of any new alerts, and then saves the results back to the database. Although our research has thus far focused on the two sensors mentioned, the algorithm would work with any sensor that could send its output to this database. Communication between the

component IDSs and the database currently relies on custom "bridging" software. As vendors begin to support the the Internet Engineering Task Force Intrusion Detection Working Group's emerging Intrusion Detection Message Exchange Format[8] we will consider supporting this as our communication mechanism.

4. Definitions

Table 5.1. r value calculation for addresses 172.16.112.20 and 172.16.112.40

	Decimal	Binary
Address 1	172.16.112.20	10101100.00010000.01110000.00010100
Address 2	172.16.112.40	10101100.00010000.01110000.00101000
Mask	255.255.255.192	11111111.11111111.11111111.11000000

$r = 26$

RealSecure version 5.0 is capable of identifying over 250 different attack types. Because attackers can use multiple techniques to accomplish the same result[18] each of these alerts is assigned to one of five categories: *discovery*, *scan*, *escalation*, *denial-of-service* (DoS), and *stealth*. Alerts in the *discovery* category are indicative of network discovery activity (IP sweeps, DNS zone transfers, etc.). Alerts in the *scan* category indicate port scanning activity. The *escalation* category consists of privilege escalation type attacks (password guessing, buffer overflows etc.). The *DoS* bucket consists of attacks that prevent access to services (SYN floods, Smurf attacks, etc.). The *stealth* bucket contains alerts that indicate attempts to conceal identity (forged IP addresses, etc.).

As an attacker may be able to launch attacks from several different, legitimate IP addresses (the attacker may have access to several machines on a single subnet or may be running DHCP), it is useful to be able to quantify the proximity of two IP addresses. To this end we have defined a quantity we denote r. r is the maximum number of 1 bits in an IPv4 subnet mask that could account for the two addresses. Thus $r = 32$ when the two IP addresses are identical, and $r = 0$ when there is no way the two addresses could be on the same subnet (see Table 5.1). Due to the prevalence of Classless Inter-Domain Routing (CIDR) [9] we allow two addresses that are on different class A, B or C subnets to have a value of r greater than 0 as long as some of their high order bits are the same. For example, the two class B addresses 130.114.5.20 and 130.115.100.8 differ in their second octet, thus they can't be on the same class B network. However, if an organization owned both the 130.114.x.x and 130.115.x.x networks they could treat them as one large subnet via CIDR. Therefore

we would give these two addresses an r value of 15 since the first 15 bits in both addresses are the same.

5. Probability Assignment

Thus far we have specified how scenarios are formed given a measure of the probability that an event belongs to a given scenario. The following sections describe different approaches to the probability estimation problem. Section 5.2 describes a naïve approach, Section 5.3 describes a heuristic method, and Section 5.4 describes the application of traditional data mining techniques to this problem. Both the heuristic and data mining approaches make use of training data to optimize the probability estimates. The process by which data is collected and prepared for use is described in Section 5.1.

5.1 Data Sources and Use

Given a set of network alerts and the corresponding scenarios we could use this data to train the free parameters in our probability estimate algorithms to reproduce the scenarios in the data set. Unfortunately, the authors know of only one dataset that contains realistic network traffic and attack scenarios for which ground truth is known[10] and this dataset does not contain enough scenarios to be used for parameter estimation. However, if the fusion system could match the performance of a human expert this would be a valuable contribution. We therefore find scenarios in real network traffic by hand and use this data for optimization.

Every year at the DEF CON conference participants play "capture the flag"[1]. There are two types of participants in this game: those who run servers and those who try to break into them. The former group gets points for running more services (as this makes the host more vulnerable) and the latter group gets points for each server they compromise. The rules stipulate that a host is considered compromised only if the attacker can put a file containing his or her name in the root of the file system. Additional points are awarded for "style".

This is an excellent data source because it contains a large amount of attacks and scenarios. It is, however, an unusual data source. The volume and frequency of attacks is far greater than that experienced on most operational networks. Additionally, all hosts participating in the game were on the same subnet. Ordinarily such a large volume of attacks in a small time window emanating from a single subnet would be indicative of a coordinated attack by a single organization. While that is not the the case here, this type of activity is similar to what one might see in a cyber terrorism attack. For example, recently many Israeli web

Figure 5.2. Alert occurrence times in the DEF CON data. Each horizontal line depicts a single scenario. The points on the line indicate when an alert occurred. The symbol indicates the class of the alert. Depicted are the twenty five scenarios containing the largest number of alerts.

sites have been the target of coordinated attacks by Palestinian groups [14]. The type of traffic experienced by these sites may be similar to the traffic in this data set.

We believe that different tactical situations would produce different alert stream characteristics. For example, a "cyber war" would produce a very different alert profile than would normal traffic. A good fusion approach should be flexible enough to work in all such environments. In our case, the optimal parameters would change with the tactical situation, but these could be easily swapped via a configuration file. Thus far we have only analyzed data from the DEF CON 2000 hacker conference's "capture the flag" game. We are investigating other data sources that contain more typical traffic patterns. Results on these data sources will help us determine how robust our approach is to different tactical situations.

Two and a half hours of tcpdump data from the DEF CON conference was replayed on a network using Lincoln Laboratory's Netpoke tool. All traffic was sniffed by ISS RealSecure's network sensor (in its "out of the box" configuration). This produced 16,250 alerts. The alerts were mapped by the first author into eighty nine scenarios. The largest scenario contained 10,912 alerts. Twenty seven scenarios contained only a single alert. The average number of alerts per scenario was 183. Twenty one scenarios contained alerts from more than one source address. All the remaining scenarios contained alerts from a single source IP address. The length of time covered by the alerts in a scenario varied widely (see Figure 5.2). Often two consecutive alerts in a scenario were separated by hundreds of alerts belonging to other scenarios making scenario reconstruction difficult.

When hand tagging scenarios some idiosyncrasies of ISS RealSecure version 5.0 were discovered. In particular we noticed that the source and destination addresses reported by RealSecure were often the reverse of what would be expected. We expect the attacker's address to be listed as the source and the victim's address to be listed as the destination. The majority of RealSecure alerts follow this pattern, but some do not.[1] Classification features designed to account for these idiosyncrasies are discussed in Section 5.4.

Tagging the scenarios provides us with examples of data which should cause our probability estimate algorithms to produce a "join the scenario" decision (a high probability that the alert belongs to the scenario). Optimization requires negative ("don't join the scenario" decision) training examples as well as positive ones. In order to produce negative training examples a program was written that reproduces the decisions the computer would need to make given a set of data. This program takes a file which contains the alerts in the order in which they were produced. Each alert is tagged with a number indicating its scenario membership. The program loops through this file and writes one line of output for each decision the fusion system would have had to make. The output contains the "correct" decision and the statistics on which this decision must be based (time difference between alerts, r value

[1] It appears that RealSecure version 5.0 always reports *windows_access_error*, *netbus*, and *dns_length_overflow* alerts backwards while *dns_all*, *iphalfscan* and *udp_port_scan* alerts are only sometimes reported backward. The *dns_all*, *iphalfscan* and *udp_port_scan* alerts appear to be reported backward as an "echo" effect. They are only reported with the source and destination address backward shortly after the same alert has been reported correctly. Not every occurrence of one of these alerts is followed by another with the source and destination reversed.

of the IP addresses in question, etc.). Pseudo-code for this algorithm is included below:

```
WHILE there are still alerts in the file
    Read the next alert from the file
    FOR each scenario in memory
        If ID of scenario matches ID of alert then
           positive training example
        else
           negative training example
        END IF
        Generate and write out features
    END FOR
    Add new alert to correct scenario
END WHILE
```

Running this algorithm on the alerts in this dataset produced 1,001,436 training examples.

Once the training data has been generated it must be separated into training, validation and test data sets. The validation and test data sets are used to ensure that we are learning a function which will generalize well to other data (see [13] for an discussion of overfitting). In order to ensure accurate results the proportion of "join" and "don't join" examples must be equal in the train, test and validation data files. A script was written to randomly split the data into these three files (50% for training, 25% for evaluation, and 25% for testing) while preserving the prior probabilities.

5.2 Naïve Technique

The most obvious way to group alerts together is by source IP address. This naïve approach assumes that all attacks belonging to a single scenario share a common source address. Thus scenarios are formed by comparing the new alert to the most recent alert in the scenario. The new alert is added to the scenario if and only if the source IP addresses are identical.This technique was tried, but due to the tendency of RealSecure to reverse source and destination IP addresses it did not perform very well.

To compensate for this we applied some simple heuristics to correct for alerts whose source and destination address might be reversed. To deal with alerts whose source and destination addresses are always reversed we simply replaced the source address with the destination address before comparing it to other alerts. Other alerts are only sometimes reversed. This usually occurs as an "echo" effect where an alert is reported

twice; first with the source and destination correct and then with the source and destination exactly reversed. We handled these alerts by producing a join decision when the new alert and the most recent alert in the scenario were of this type and their source and destination addresses were exactly opposite. Results of this approach are presented in Section 6.

5.3 Heuristic Technique

The technique discussed in this section extends work described earlier [7]. It attempts to model our intuitions about how to form scenarios. We believe that:

- Attacks often follow a logical progression. As such, certain sequences of alerts types would be more common than others.
- The time between alerts is a good predictor of scenario membership. Alert latencies vary with alert type.
- The range of source IP addresses in a group of alerts is a good predictor of scenario membership. The range of addresses one is likely to see is dependent on the alert type; some attacks lend themselves to IP spoofing and others do not.

We approximate these intuitions with a bigram model. Given a scenario and a new alert, probability is assigned by considering the most recent alert in the scenario and the new alert. Thus if our scenario contains three alerts, a_1, a_2, and a_3, and we get a new alert, a_{new}, we compare a_{new} to a_3 only. The probability that two alerts, from buckets i and j respectively, belong in the same scenario is the product of three quantities: l_{ij}, which accounts for the strength of the link between the two alerts, $\sigma_{ij}(\Delta t)$, which accounts for the time lag between the two alerts, and $R_{ij}(r)$, which accounts for the source IP addresses of the two alerts. All three quantities are between 0 and 1, thus their product is also between 0 and 1 (see Figure 5.3).

The first quantity, l_{ij}, is the probability that an alert from bucket i is followed by an alert from bucket j. For example the probability that a scan will be followed by an escalation might be higher than the probability than an escalation will be followed by a DoS because an attacker often needs to scan a system prior to gaining access to it while an attacker who has gained unauthorized access to a system is less likely to deny services to a network.

The second quantity, $\sigma_{ij}(\Delta t)$, accounts for the time between the alerts. It is a sigmoid function, defined by $\sigma_{ij}(\Delta t) = 1/\left(1 + e^{\alpha+\beta\Delta t}\right)$. The

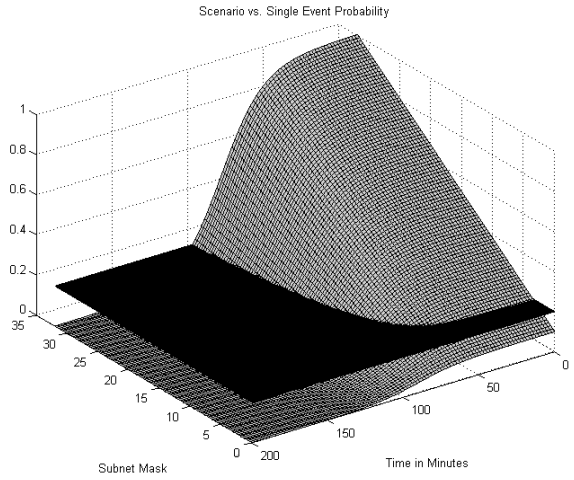


Figure 5.3. Decision surface for a single transition type. The height of the curve is the probability that two alerts in the given time span with the given r value belong in the same scenario. The black plane is the threshold below which the new alert would not join the scenario and would start a new scenario.

shape of the function (determined by the values of α and β) is different for each transition type (where transition type is defined by the buckets containing the two alerts). Thus the sigmoid representing the time between DoS attacks might fall off quickly (as we expect these alerts to occur in rapid succession) while the sigmoid representing the time between network discovery and privilege escalation might fall off rather slowly since an attacker might not launch an attack immediately after gathering information.

The final quantity, $R_{ij}(r)$, accounts for the IP address range of the two alerts. Let r be the maximum number of 1 bits in an IPv4 subnet mask that could account for the two IP addresses (as defined in section 4). $R_{ij}(r)$ is a function of r. The exact function depends on the buckets i and j. For each transition type values of $R_{ij}(r)$ are picked for $r = 0$, 8, 16, 24 and 32. The value of $R_{ij}(r)$ for other values of r is determined by linear interpolation. For example, since source addresses are often spoofed in a DoS attack we might expect $R_{ij}(r)$ to have a relatively high value for all values of r if $i = j =$ DOS.

To compensate for the idiosyncrasies of RealSecure the same heuristics used by the naïve approach were applied before r was calculated. $R_{ij}(r)$ is then calculated from this modified value of r. The following algorithm is used to calculate r:

- `new_a = new alert`

- `r_a = most recent alert in the sceanario`
- `if always_reversed(new_a) and not always_reversed(r_a) then r = calc_r(new_a.destination, r_a.source)`
- `else if always_reversed(r_a) and not always_reversed(new_a) then r = calc_r(new_a.source, r_a.destination)`
- `else if sometimes_reversed(new_a) and sometimes_reversed(r_a) and r_a.source = new_a.destination and r_a.destination = new_a.source then r = 32`
- `else r = calc_r(new_a.source, r_a.source)`

For each transition type the model has the eight parameters described above. Since there are five bucket types there are $5^2 = 25$ transition types. Thus there are $8 \times 25 = 200$ total parameters. To optimize the parameters we converted our fusion code to work from a training file as discussed in Section 5.1 and compiled it into a shared library. We can then call the fusion code passing it a matrix of parameters. The fusion code reads each line in the training file and, using the parameters passed, produces the probability that the example read from the file should produce a "join" decision. Since the training file encodes the "right" answer we can calculate the squared error for this example (the "correct" probability is 0 if the example should have produced a "don't join" decision and 1 otherwise). When all the examples in the training file have been scored the sum of squared errors is returned. Using the constrained optimization routine fmincon in MATLAB[5] we can find the parameter values that minimize the sum of squared errors. With each iteration the evaluation data set was also scored and the sum of squared errors on this data set was tracked. This would allow us to stop the training early if overfitting became an issue.

5.4 Data Mining Techniques

We have also assigned probabilities using data mining techniques. Here the data is prepared as described in Section 5.1 and then passed to a standard machine learning algorithm. The use of machine learning algorithms allowed us to use additional features which may provide better predictive power. As with the heuristic technique, features which indicate the bucket membership of the new alert, the bucket membership of the most recent alert in the scenario, the time between the new alert and the most recent alert in the scenario, and the value of r when the new alert is compared to the most recent alert in the scenario were used. In addition, the following features were used:

- To expand our model from a simple bi-gram model, features which indicate the bucket membership of the 3 most recent alerts in the scenario were used.
- Since attackers often use the same tools or attack types many times features were included to indicate if any previous alerts in the scenario are the exact same type as the current alert and if the most recent alert in the scenario is the exact same type as the new alert.
- Attackers often focus their efforts on a single host . We therefore include a feature that indicates if the destination address of the new alert was the target of any other alerts in the scenario.
- An attack scenario may contain components with spoofed source IP addresses while other components of the attack may use the attackers real source IP. As such a feature whose value is the maximum value of r when the source IP of the new alert is compared to the source IP addresses of all the alerts in the scenario is included.
- Once a machine has been compromised it can be used as the source of further attacks. A feature whose value is the maximum value of r when the source IP of the new alert is compared to the destination IPs of all the alerts in the scenario is therefore included.
- Since RealSecure sometimes reports the source and destination IP addresses reversed features to account for this were included. One indicates if the new alert is "backward". Another indicates if the most recent alert in the scenario is "backward". Both of these features vary between 0 and 1 where 1 indicates that the given alert type is always reported backward by RealSecure and 0 indicates that the given alert type is never reported backward.
- As RealSecure alerts sometimes suffer from an "echo effect" where an alert is sent twice with the source and destination IP addresses of the two alerts exactly switched we included a feature which indicates if the source IP of the new alert matches the destination IP of the most recent alert in the scenario and the destination IP of the new alert matches the source IP of the most recent alert in the scenario.

6. Experimental Results

All experiments were conducted on the DEF CON data described in Section 5.1. A test data set containing 500,717 patterns was used to

train the heuristic and data mining algorithms. An evaluation data set containing 250,358 patterns was used to monitor for over-fitting and to choose algorithm parameters. A test data set containing 250,361 patterns, which was not used for any other purpose, was used for final algorithm evaluation. All of the following results reflect performance on this data set.

6.1 Naïve Technique

Table 5.2. Confusion matrices from Naïve approach on test data set . Two hundred forty six of the 4,041 patterns that should have produced a "join" decision (6.09%) incorrectly produced a "don't join" decision, and ten of the 246,320 patterns that should have produced a "don't join" decision incorrectly produced a "join" decision.

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	246,310	10	246,320
	Join	246	3,795	4041
	Totals	246,556	3,805	250,361

(a)

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	100.00%	0.00%	100%
	Join	6.09%	93.91%	100%

(b)

The performance of the naïve approach is presented in Table 5.2. This simple approach does remarkably well on this data set making only 256 errors on 250,361 patterns. It might seem that this performance is adequate. However we think it is important to improve on this performance for two reasons. First, this data set contains very few attempts at stealth. The capture the flag game rewards speed but no points are awarded for stealth. We expect that more realistic data sets would not be as amenable to such a simple technique. Second, the six percent of attackers who are missed by this simple technique are precisely the individuals we need to worry about. If an attacker can be discovered by our Naïve algorithm that attacker probably can't do much damage anyway. This simple technique does help us group all of the "script kiddie" attacks together which results in significant data reduction, but we would prefer an approach that could discover more sophisticated attacks as well.

6.2 Heuristic Technique

The two hundred parameters used by our heuristic approach were optimized to the data described in Section 5.1. Monitoring the performance on the evaluation data set revealed that overfitting was not a problem; probably because this is not a very complex classifier. After optimizing the parameters needed by our heuristic approach we scored the test data set which contained 250,361 patterns. The results are depicted in Table 5.3.

Table 5.3. Confusion matrices from heuristic approach on test data set. The algorithm tries to match the human decision. For example, 4,041 test examples should have produced a "join" decision. The algorithm correctly produced the "join" decision 3,589 times (88.81% of the time).

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	246,315	5	246,320
	Join	452	3,589	4,041
	Totals	246,767	3,594	250,361

(a)

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	100.00%	0.00%	100%
	Join	11.19%	88.81%	100%

(b)

The error rates for this algorithm are slightly higher than the error rates for the naïve approach even though the same features used by the naïve algorithm are used by this classifier. The problem is that we have constrained the possible set of decision surfaces this classifier can produce and a decision surface equivalent to that used by the naïve approach is not in possible. Additional features or a different set of possible decision surfaces would need to be considered by the classifier in order to improve its performance enough to be useful. As it is not immediately clear which features would be most useful and how these features should be combined with those we are already considering we will replace this technique with the more powerful data mining algorithms discussed in Section 5.4 in our future work.

6.3 Data Mining Techniques

Radial basis function networks, multi-layer perceptrons and decision trees were applied to the training data using the features described in

Table 5.4. Confusion matrices from a decision tree on the test data set. The decision tree tries to match the human decision. For example, 4,041 examples should have produced a "join" decision. The decision tree produced the correct decision on 4,037 of these (99.90% of the time).

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	246,316	4	246,320
	Join	4	4,037	4,041
	Totals	246,320	4,041	250,361

(a)

		Predicted		
		Don't Join	Join	Totals
Target	Don't Join	100.00%	0.00%	100%
	Join	0.10%	99.90%	100%

(b)

Section 5.4. The LNKnet data mining package [12] was used for all experiments. A decision tree with 36 non-terminal nodes produced the best performance. Confusion matrices for this classifier appear in Table 5.4. The tree was trained using the CART algorithm [3] until no errors were produced on the training set. Nodes were ordered by their effect on the error rate on the training data and then pruned, removing nodes which least affect the error rate first. The validation data set was used to choose the optimal number of nodes.

The error rate of the decision tree is extremely low. It performs significantly better than either of the other two approaches considered, catching the six percent of attack scenario missed by our naïve approach. As this six percent of attackers are probably the ones we should be concerned about this performance increase is significant. We suspect the improved performance is largely a result of the additional features available to the algorithm. Those features used by the the tree as splitting criteria in non-terminal nodes indicate which features were most useful. These are listed below:

- Of the twenty features which indicate the bucket type of the new and three most recent alerts in the scenario only four were used as splitting criteria. The features used indicated if the new alert was a scan, if the new alert belonged to the stealth bucket, if the most recent alert in the scenario belonged to the stealth bucket, and if the second most recent alert was an escalation alert.

- Our measure r proved to be very useful. Recall from Section 4 that r is a number which indicates the similarity of two IP addresses. We can compare the new alert to the most recent alert in the scenario and produce r values four ways. We can compare the source of the new alert to the source of the most recent alert, the destination of the new alert to the source of the most recent alert, the source of the new alert to the destination of the most recent alert and the destinations of both alerts may be compared. All four features were used by the decision tree. Additionally a feature whose value is the maximum value of r when the source address of the new alert is compared to the source address of each alert in the scenario was used.

- A feature which indicates if the destination of the new alert was the destination of any previous alerts was used.

- The time between the new alert and the most recent alert in the scenario was used as a splitting criteria.

- A feature which indicates if the most recent alert was the same type as the new alert was used as was a feature which indicates if any alert in the scenario was the same type as the new alert.

- A feature which indicates if the new alert might have its source and destination addresses switched was used. This feature varies between 0 and 1 where 1 indicates that this type of alert is always reported reversed and 0.5 indicates that the alert has its addresses reversed about half of the time.

7. System Benefits

The system described provides great benefits to the security analyst. The value of the system was made clear when we began to examine the DEF CON 2000 data. Looking at the 16,250 alerts in this data set was initially overwhelming. It was extremely difficult to figure out what was happening on the network and which alerts represented activity that would be of concern to somebody trying to protect this network. It took the first author about fifteen hours to hand tag this data into scenarios. Once this was done there were only eighty seven scenarios to consider. The scenarios conveyed much more information that the original alert stream. A quick look at many scenarios was sufficient to infer the attackers intentions and tools. For example many scenarios consisted of scans for port 80 followed by a long string of HTTP specific exploits. These individuals were clearly running scripts that tried every

HTTP exploit they knew of. The exploits attempted on every host were identical. Thus a network administrator who determined that her hosts were not vulnerable to these exploits could quickly determine that the entire scenario containing several hundred alerts was not a security risk. It might also allow her to preemptively check the security of other web servers once the the beginning of the scenario was seen.

While the benefits of this type of analysis are clear, it is not feasible for administrators to spend fifteen hours analyzing two and a half hours of network data. It is necessary to automate the analysis. The use of the atom model makes it possible to reproduce these decisions in real-time. Our system, using the binary tree discussed in Section 6.3, was able to accurately reproduce the author's analsis of the data in under two seconds!

The system would be even more valuable if it were able to assess the security risk of a scenario and update that risk assessment as the scenario changed. To this end we have begun work on a declarative language that administrators can use to specify risk assignment rules. This language will be the subject of a future publication.

8. Discussion and Summary

We have presented a scenario building approach to IDS fusion. We believe that the scenarios produced by our software convey more information to an analyst than the individual alerts. The scenario building process requires us to be able to produce an estimate that an alert belongs to a scenario. We have examined three approaches to this problem. Our analysis indicates that the data mining approach is superior to either of the other two considered. The data set used for this analysis was fairly simple. This allowed our naïve approach to perform better than we expect it would on more realistic data. Even on this relatively simplistic data the improvement offered by machine learning techniques is significant. We are in the process of obtaining a large amount of data from an operational military network so that we can assess our technique on more realistic traffic.

We intend to improve the system by integrating the security risk assessment language and adding a graphical user interface for viewing the constructed scenarios. Furthermore we intend to create graphical tools to ease the burden of hand tagging scenarios to produce training data.

References

[1] DEF CON 8 conference. Las Vegas, NV, 2000. www.defcon.org.

[2] E. Amoroso. *Intrusion Detection.* Intrusion.Net Books, Sparta, New Jersey, 1999.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Inc., Belmont, California, 1984.

[4] C. Clifton and G. Gengo. Developing custom intrusion detection filters using data mining. In *2000 Military Communications International Symposium*, Los Angeles, CA, October 2000.

[5] T. Coleman, M. A. Branch, and A. Grace. *Optimization Toolbox for Use with MATLAB.* The MathWorks, Inc., 1999.

[6] R. K. Cunningham, R. P. Lippmann, D. Kassay, S. E. Webster, and M. A. Zissman. Host-based bottleneck verification efficiently detects novel computer attacks. In *IEEE Military Communications Conference Proceedings*, Atlantic City, NJ, 1999.

[7] O. M. Dain and R. K. Cunningham. Building scenarios from a heterogeneous alert stream. In *Proceedings of the IEEE SMC Information Assurance Workshop*, West Point, NY, June 2001.

[8] B. S. Feinstein and G. A. Matthews. The Intrusion Detection Exchange Protocol (IDXP), August 2001. www.ietf.org.

[9] P. S. Ford, Y. Rekhter, and H.-W. Braun. Improving the routing and addressing of IP. *IEEE Network*, 7(3):10–15, May 1993.

[10] J. Haines, L. Rossey, and R. Lippmann. Extending the 1999 evaluation. In *DISCEX Proceedings*, June 2001.

[11] Internet Security Systems. RealSecure console user guide. Atlanta, GA, 2000. www.iss.net.

[12] R. P. Lippman, L. Kukolich, et al. LNKnet: Neural network, machine learning, and statistical software for pattern classification. *Lincoln Laboratory Journal*, 6(2):249–268, 1993.

[13] T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[14] J. Schwartz. Hacker defaces pro-israel web site as the mideast conflict expands into cyberspace. *The New York Times*, November 3, 2000.

[15] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS-a graph based intrusion detection system for large networks. In *19th National Information Systems Security Conference Proceedings*, pages 361–370, October 1996.

[16] A. Valdes and K. Skinner. An approach to sensor correlation. In *Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, October 2000.

[17] S. Vasile. Automated intrusion detection environment (AIDE). In *Joint Aerospace Weapon Systems Support, Sensors, and Simulation Proceedings*, June 2000. http://www.adpansia.org/events/brochure/092/proceed.htm.

[18] B. J. Wood and R. A. Duggan. Red teaming of advanced information assurance concepts. In *DISCEX 2000*, Hilton Head, South Carolina, January 2000.

Chapter 6

USING MIB II VARIABLES FOR NETWORK INTRUSION DETECTION

Xinzhou Qin
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332 USA
xinzhou@cc.gatech.edu

Wenke Lee
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332 USA
wenke@cc.gatech.edu

Lundy Lewis
Aprisma Management Technologies
121 Technology Drive
Durham, NH 03824 USA
lewis@aprisma.com

João B. D. Cabrera
Scientific Systems Company
500 West Cummings Park, Suite 3000
Woburn, MA 01801 USA
cabrera@ssci.com

Abstract

Detecting and resolving security and performance problems in distributed systems have become increasingly important and challenging because of the tremendous growth in network-based services. Intrusion detection is an important security technique for networks and systems. In this paper, we propose a methodology for utilizing MIB II objects for network intrusion detection. We establish the normal profiles of network activities based on the information provided by the MIB II variables and use data mining techniques and information-theoretic measures to build an intrusion detection model. We test our MIB II-based intrusion detection model with several Denial of Service (DoS) and probing attacks. The results have shown that the model can detect these attacks effectively.

Keywords: Intrusion Detection, Anomaly Detection, Data Mining, MIB II, Network Security Management

1. Introduction

An intrusion detection system (IDS) is an important component of network security mechanisms. An IDS collects audit data from the system and network activities, e.g., BSM [24] and tcpdump [12], and analyzes the information if there is an occurrence of intrusion or not.

There are two major intrusion detection techniques, i.e., *misuse detection* and *anomaly detection*. Misuse detection is a signature-based technique that identifies attacks based on the patterns of intrusion behavior or effects. Anomaly detection uses the established normal profiles to identify any unacceptable deviation as the possible result of an intrusion such as [16]. Anomaly detection is intended to detect new attacks that have no known signatures yet. Misuse detection cannot catch new intrusions, however, misuse detection can be more accurate and efficient than anomaly detection.

Building a good anomaly detection model is very difficult. Anomaly detection technique is based on the understanding of the normal activities of systems and networks. However, it is difficult to construct comprehensive profiles that cover all normal activities. Another difficulty in anomaly detection is the feature construction process. Generally, when building an IDS, we need to determine the features in order to effectively represent the network and system activities. Some of the features can be extracted from audit data directly, while others need statistics computed by using the audit data, for example, the number of incoming TCP segments within a certain period of time. It is nontrivial to decide what features should be extracted or computed.

Our research aims to improve the efficiency and performance of intrusion detection. In this paper, we propose to take advantage of the information provided by the MIB II [18] in Network Management System (NMS) to establish the normal profiles of network activities. We use information-theoretic measures, namely *conditional entropy*, and classification techniques to construct our anomaly detection model.

The remainder of the paper is organized as follows. In Section 2, we introduce MIB II and information-theoretic measures. Section 3 presents model construction techniques. Experiments and performance evaluations are described in Section 4. In Section 5, we discuss the strengths and limitations of our model. Section 6 compares our work with related efforts. Finally, we summarize the paper and outline the future work in Section 7.

2. Background

In this section, we describe MIB II, one type of data source provided by network management systems. We also describe several information-theoretic measures and discuss how these measures relate to the anomaly detection performance.

2.1 MIB II

The purpose of a network management system (NMS) is to operate, administrate, maintain, and provision the network and services. The goal of NMS is to assign and control proper network resources to address service performance needs and the network's objective. An NMS has several key functions, i.e. fault, accounting, configuration, performance, and security management.

In a network management system, the network elements such as hosts, routers, switches, etc. can have a management process, namely *agent*, installed for collecting information. The *network manager* manages these network elements. The network manager and agents are linked by a network management protocol. In the TCP/IP network, this protocol is Simple Network Management Protocol (SNMP) [23]. The Management Information Base (MIB) [18] is a collection of objects that represent the resources in the network and hosts. MIB II defines the second version of the management information base in SNMP. Its structure is defined in the Structure of Management Information (SMI) [23].

In an NMS, the network manager queries the agents and retrieves the value of MIB objects to perform the monitoring functions. It can also change the settings of agents by modifying the value of specific

MIB variables. Agents can also send some information to the network manager unsolicited given the proper policy and configuration.

There are several groups in the standard MIB II that collect the information on different network layers and protocols. These are the IP group, ICMP group, TCP group, etc. In each group, the MIB II variables provide information on system configuration, network traffic, control and error statistics. Therefore, MIB II objects provide us an audit source from which we can have a comprehensive understanding about conditions and operations of the managed elements and network.

2.2 Entropy and Conditional Entropy

Entropy, or Shannon-Wiener Index [20], is an important concept in information theory and communication theory. It measures the uncertainty (or impurity) of a collection of data items.

Definition 1 *For a dataset X where each data item belongs to a class $x \in C_X$, the entropy of X relative to this $|C_X|$-wise classification is defined as*

$$H(X) = \sum_{x \in C_X} P(x) \log \frac{1}{P(x)}$$

where $P(x)$ is the probability of x in X.

The typical interpretation of entropy is that it specifies the number of bits required to encode (and transmit) the classification of a data item. The entropy value is smaller when the class distribution is skewer, i.e., when the data is more "pure".

For anomaly detection, we can use entropy as a measure of the regularity of audit data. Each *unique* record in an audit dataset represents a class. The smaller the entropy, the fewer the number of different records (i.e., the higher the redundancies), and we say that the more regular the audit dataset. High-regularity data contains redundancies that help predicting future events because the fact that many events are repeated (or redundant) in the current dataset suggests that they will appear in the future. Therefore, an anomaly detection model constructed using dataset with smaller entropy will likely be simpler and have better detection performance. For example, if the audit data contains a single event class, e.g., the MIB II object *icmpInEchos* has a constant value, then the entropy is 0 and a single rule can identify any other event, e.g., when the *icmpInEchos* value has changed, as an anomaly. If the audit data contains many event types, then the entropy is greater than 0 and a more complex model is needed.

Definition 2 *The conditional entropy of X given Y is the entropy of the probability distribution $P(x|y)$, that is,*

$$H(X|Y) = \sum_{x,y \in C_X, C_Y} P(x,y) \log \frac{1}{P(x|y)}$$

$P(x,y)$ is the joint probability of x and y and $P(x|y)$ is the conditional probability of x given y.

Because of the temporal nature of user, program, and network activities, we need to measure the temporal or sequential characteristic of audit data. Using the definition above, let X be a collection of sequences where each is denoted as $(e_1, e_2, \ldots, e_{n-1}, e_n)$, and each e_i is an audit event; and let Y be the collection of subsequences where each is $(e_1, e_2, \ldots, e_k)$, and $k < n$, then the conditional entropy $H(X|Y)$ tells us how much uncertainty remains for the rest of audit events in a sequence x after we have seen y, i.e., the first k events of x (note that since y is always a subsequence of x here, we have $P(x,y) = P(x)$). For anomaly detection, we can use conditional entropy as a measure of regularity of sequential dependencies. And as the case of entropy, the smaller the conditional entropy, the better. For example, if each audit trail is a sequence of events of the same type, e.g., $X = \{aaaaa, bbbbb, \ldots\}$, then the conditional entropy is 0 and the event sequences are very deterministic. Conversely, a large conditional entropy indicates that the sequences are not as deterministic and hence much harder to model.

3. Model Construction

In this section, we discuss the architecture of our MIB II-based intrusion detection model. We then describe how we use conditional entropy in the construction of anomaly detection model, more specifically, in the selection of optimal window size. We also discuss how to evaluate the deviation from thresholds as an anomaly.

3.1 Model Architecture

As discussed in Section 1, there are two major intrusion detection techniques, i.e., *misuse detection* and *anomaly detection*. Misuse detection is a signature-based technique that identifies the attacks based on the patterns of intrusion behavior or effects. Anomaly detection uses the established normal profiles to identify any unacceptable deviation as the possible result of an intrusion.

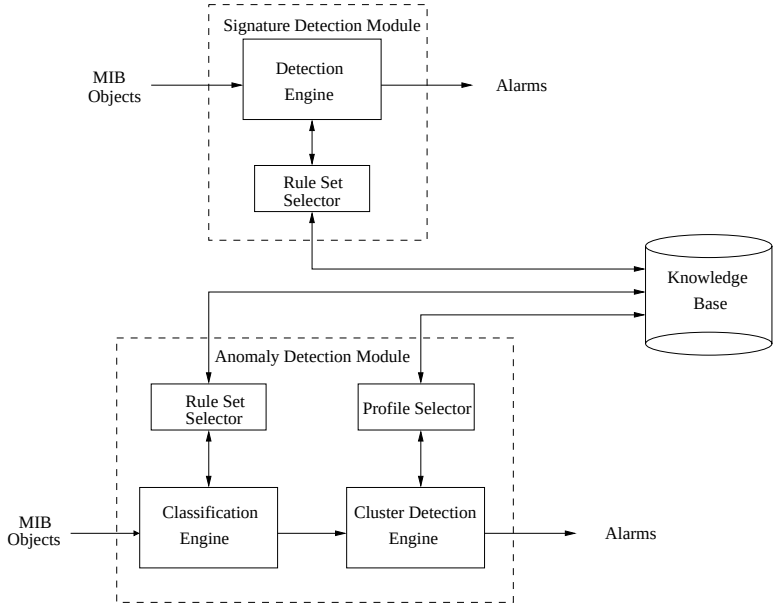


Figure 6.1. MIB II-based ID Architecture

In our MIB II-based ID architecture, as shown in the Figure 6.1, we have both misuse detection module and anomaly detection module in order to combine the strengths of these two ID techniques. These two modules work parallel for detecting intrusions.

The major components in this MIB II-based ID model are:

Knowledge Base: The Knowledge Base stores the normal profiles of key MIB variables and the corresponding rules that represent signatures of known intrusions that are retrieved by the misuse detection engine. For anomaly detection module, Knowledge Base provides the classification rules and normal profiles of clusters.

Signature Detection Module: It detects the intrusions based on the patterns of the attacks. When the MIB objects are input, *Detection Engine* requests *Rule Set Selector* to retrieve the rule sets from the Knowledge Base and match them with the corresponding MIB variables for the intrusion detection.

Anomaly Detection Module: It is responsible for detecting the "new" intrusions that are unknown in the Knowledge Base. In this module, there are two core correlation parts:

- *Classification Engine*: It classifies the input MIB variables based on the established classification rules stored in the Knowledge Base. These rules are learned by using normal data.

- *Cluster Detection Engine*: This engine is used for the final step of anomaly detection. The data fed to this engine is not from the raw MIB variables but some cluster information generated by *Classification Engine*. The *Profile Selector* selects the cluster profile from the Knowledge Base.

3.2 Anomaly Detection Module

3.2.1 Anomaly Detection Model Design Overview. Since MIB II can provide information on different network layers and protocols, in our MIB-based anomaly ID model, we use 91 MIB II objects related to network traffic, error and configuration information as our monitoring variables. These variables are in the groups of IP, ICMP, UDP, TCP and SNMP.

In order to have a comprehensive understanding of network activities, as shown in Figure 6.2, we partition the 91 MIB II variables into 18 sub-modules based on their definitions as well as the network layers and protocols that they belong to. For example, in the sub-module of *ICMP_In_Error*, we include the MIB II variable of *icmpInErrors*. In the sub-module of *UDP_Out*, the MIB II variable of *udpOutDatagrams* is included. The architecture is designed based on the protocol-hierarchy property of MIB II and objects' definitions.

When applying our anomaly detection model, all sub-modules are used first, and if there are any potential anomalies, the corresponding sub-module will send out the alarms. When an alarm is fired by any of the protocol ID sub-modules, we consider an anomaly in that protocol. In anomaly detection, since we do not assume to have prior knowledge about the intrusion, such as what protocol or which layer it will manifest itself, we need to further examine alarms from (higher) layer(s) and protocol(s). With ID sub-module in each protocol, we can detect an anomaly more precisely and locate the protocol and layer where the anomaly has happened.

3.2.2 Anomaly Detection Module Construction. When building anomaly detection models, we use classification techniques to generate a rule set for each object to predict its normal values. In our study, we use RIPPER [5], a rule learning algorithm, as our classification engine because of its good generalization accuracy and concise rule conditions.

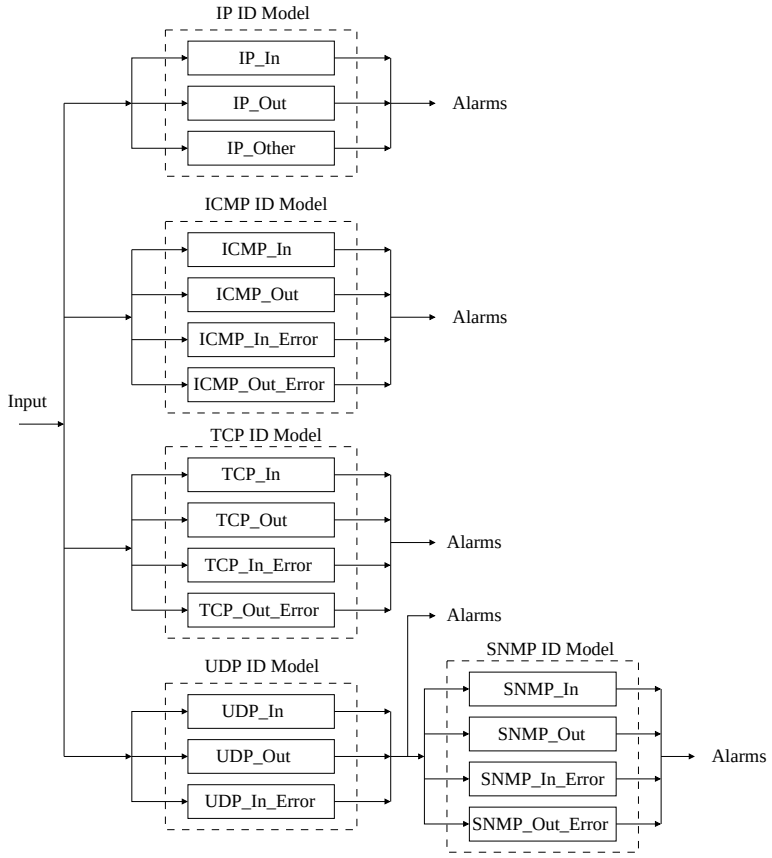


Figure 6.2. MIB II-based Anomaly ID Model

We use an *object-time* approach in constructing the anomaly detection model of each MIB II object. Specifically, for each MIB object, we use the previous values of this object to predict the coming value, i.e., we use $O_{t-n}, O_{t-(n-1)}, ..., O_{t-2}, O_{t-1}$ to predict the value O_t. The intuition is that there exists a predictable temporal relationship under normal conditions for each object, however, such temporal relationship will be broken or disturbed by the abnormal conditions such as the occurrence of intrusions. We train the ID model to learn such normal temporal relationship and detect the potential intrusions. In this approach, the main challenge is how to select a good window size.

As described in Section 2, in anomaly detection, we can use conditional entropy as a measure of regularity of sequential dependencies. The smaller the conditional entropy, the better we can predict the sequence. In addition, there is a direct connection between conditional entropy and information gain when the regularity of sequential dependencies is used directly in the anomaly detection model [15]. The lower the conditional entropy, the higher the information gain, the better the detection performance. As Figure 6.3 shows, for the object of *ipInReceives*, the conditional entropy decreases with the increase of the sequence length. Figure 6.4 shows the changes of the misclassification rate (i.e., error rate) of the model for object *ipInReceives* while the sequence length increases. We see the trends in the two figures agree with each other. In the model construction, for those MIB II objects whose conditional entropy is larger than zero, we do further modeling analysis such as the sequence length optimization.

Intuitively, the more information we have, the better detection performance we will achieve. For example, from Figure 6.4, it shows that the misclassification rate is the lowest when the sequence length equals 21. However, there is always a cost for any gain. With a longer sequence length, though we have included more information, we also increase both the processing time of data sets and the complexity of the model. Therefore, we should balance the trade off between the detection performance and the cost. As in [15], we also use the running time as the measurement of the cost. Here, we define the time cost as the running time when the test data sets are fed into the model for processing and calculation. We use the $Accuracy/Cost$ for selecting the optimal sequence length for each object model.

From Figure 6.5, it shows that, for the MIB II object *ipInReceives*, we should select sequence length equal to three as the final window size. That is we use the O_{t-2}, O_{t-1} to predict the value of O_t of this object.

Having determined the optimal window size for each object, we use three phases to construct model profiles. We first create two normal datasets for each object where each record is an observation sequence, e.g., O_{t-2}, O_{t-1}, O_t. One of these datasets is used for training in phase one, and the other is used for testing in phase two. In phase one, we apply RIPPER to the normal training data sets to get a set of rules for each of object. For example, for MIB II object *udpInDatagrams*, we have two rules: $C6\ 1534\ 0\ IF\ O1 \geq 4\ O2 \leq 0$, $C0\ 3633\ 570\ IF$. The first rule means if $O_{t-1} \geq 4$ and $O_{t-2} \leq 0$, then the value of O_t is predicted to be 6. In the training data sets, there are totally 1534 records matched with this rule and none contradicts. The second rule says that the default rule is $O_t = 0$, and in the training data sets, 3633

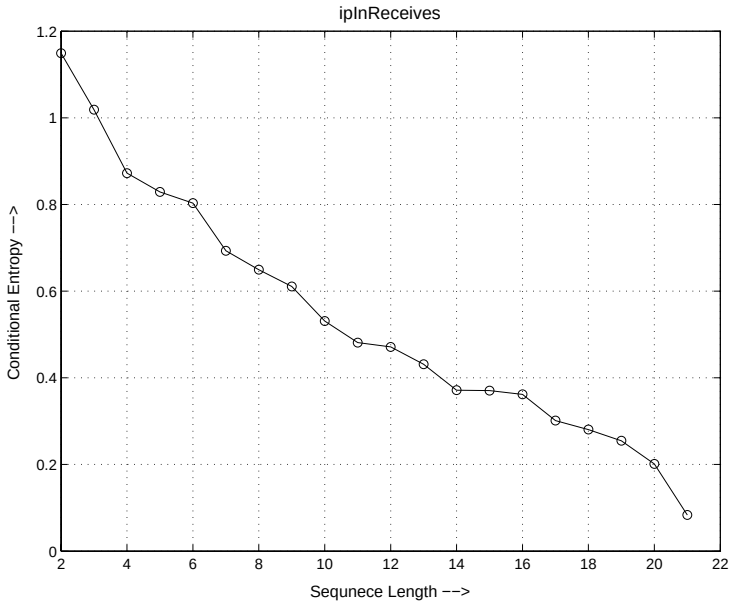


Figure 6.3. Conditional Entropy of MIB II object *ipInReceives*

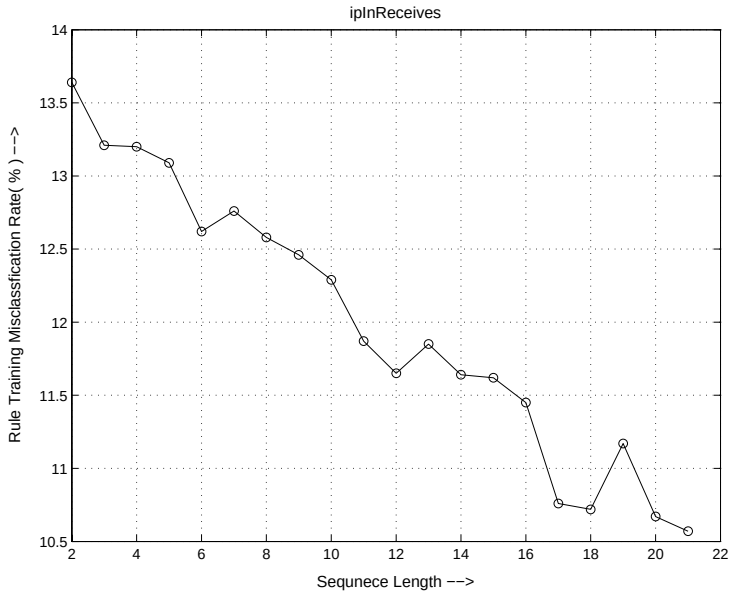


Figure 6.4. Misclassification rate of MIB II object *ipInReceives*

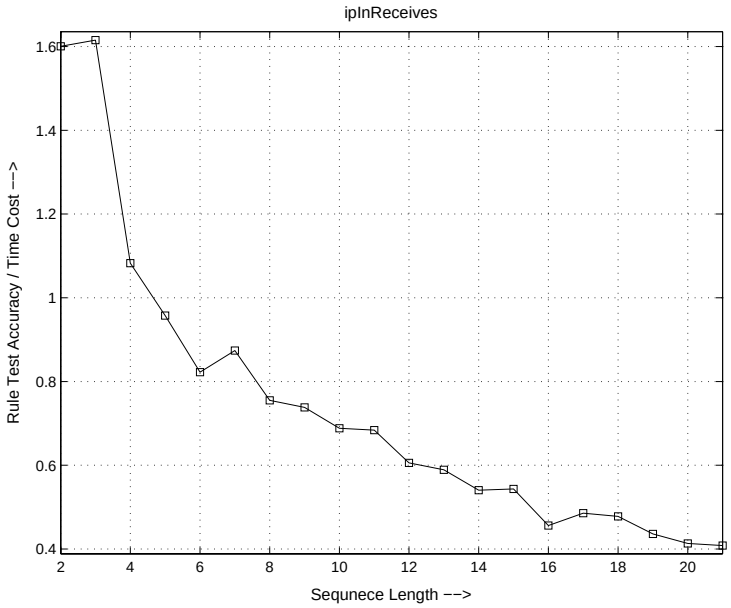


Figure 6.5. Accuracy over Cost of MIB II object *ipInReceives*

records match with this prediction rule and 570 records contradict the prediction. At the second phase, for each object, we apply the rules to the test data sets. For each prediction by RIPPER rules, we can get the "violation score" of the record. We define the "violation score" as either 0 when the prediction rule can predict correctly, or the *confidence* of the prediction rule, which is the number of matched examples divided by the sum of matched and unmatched examples when the prediction does not match with actual value. In phase three, based on the ID sub-module in each different protocol group as described in Section 3.2.1, we combine the corresponding objects' violation scores and form the ID sub-module *score vectors*. For example, in the ID sub-module "UDP-In", since it consists of two MIB II objects, namely *udpInDatagrams* and *udpNoPorts*, the score vector of the ID sub-module "UDP-In" has two dimensions, one is the violation score of *udpInDatagrams*, the other is the violation score of *udpNoPorts*.

In phase one, we apply data mining technology on the normal data sets in order to have the knowledge of normal behaviors of network activities, and this phase is a machine learning process. In phase two, the intuition is that normal data sets should match the predictions more precisely by the rules learned in phase one, i.e., with lower violation scores. These lower violation scores make up the dimensions of the

normal score vectors in phase three. The vectors in phase three serve the purposes of combining together the relevant scores, i.e., those from MIB II objects of the same modules. When an intrusion occurs, the abnormal data will violate the prediction rules and generate a large violation score which affects the corresponding score vector directly.

After establishing the normal score vectors for each ID sub-module, clustering techniques are applied for anomaly detection. With the normal score vector data sets, at each sampling time point, we compute the centroid and calculate the maximum and minimum Euclidean distances from each score vector to the centroid. For example, we have three normal score vectors, namely $\mathbf{V_1}$, $\mathbf{V_2}$, $\mathbf{V_3}$, from three different normal score vector data sets and all refer to the same sampling time point T. The centroid at the sampling point T is computed as the mean of these three vectors, i.e.: $(\mathbf{V_1} + \mathbf{V_2} + \mathbf{V_3})/3$, and Euclidean distance is from these 3 score vectors to the centroid respectively. We denote the maximum and minimum Euclidean distance as $\text{Euclidean}_{\text{max}}$ and $\text{Euclidean}_{\text{min}}$. A tolerance deviation margin α in percentage is introduced and

$$Euclidean_{max} \times (1 + \alpha)$$

and

$$Euclidean_{min} \times (1 - \alpha)$$

act as the performance upper and lower threshold respectively. In real-time monitoring, we feed the sampled real-time MIB data into the ID model, and get the score vectors of each ID sub-module. Anomaly detection is accomplished by comparing Euclidean distance of the input score vectors to the centroid with the thresholds. If the Euclidean distance is either over or below the thresholds, we regard it as an anomaly.

4. Experiments and Performance Evaluation

We deployed our ID model on a research testbed in Emulab Net (http://www.emulab.net) and conducted a series of experiments. We describe some results here.

4.1 Normal Data Sets

The testbed we configured has two LANs with a router in the middle. The background traffic is simulated based on the results and reference from [21, 4, 3]. Specifically speaking, there is around 90% tcp traffic, 10% udp traffic and the rest (around 0.01%) is icmp traffic. Http traffic occupies the major part of the tcp traffic which is approximately 90%. The packet size is distributed as the following, 30% packets have 1000-1200 bytes, 10% are 450-600 bytes, and 60% are below 100 bytes.

We monitored a host, acting as the victim in the attack experiments, for two hours and sampled the MIB II objects under normal conditions every three seconds. There are total 2400 sample points for each data set. The normal data sets are divided into two parts, namely *normal training data sets* and *normal test data sets*. *Normal training data sets* is 40% of the total normal data sets and the rest 60% is the *normal test data sets*. The *normal training data sets* are used to train the RIPPER to get the rules for each object under different window size. The *normal test data sets* are used by the RIPPER rules and calculate the test accuracy and time cost under different window size in order to get the optimal window size for each object. The *normal test data sets* are also used to establish the violation score vectors, the normal temporal centroids, and Euclidean distances for each ID sub-module.

4.2 Evaluation under Attacks

In this section, we present the evaluation results on the MIB II-based ID model under different intrusions.

4.2.1 Misuse Detection. As discussed in Section 6.1, misuse detection has the advantages of high accuracy and efficiency, however the domain knowledge of the nature of the intrusions are needed. In MIB II variables, there are many objects that can directly indicate the nature of some intrusions, therefore, such MIB II variables can be used directly for setting up the signature-based detection rules. For example, MIB II object *icmpInEchos* counts the total number of inbound icmp ping packets. Since the syntax of this MIB II object is *counter*, we need to use the delta value, i.e., $X_{t+1} - X_t$, to show the changes of the object. Therefore, by using the delta value of object *counter*, we can detect the Ping Flood attack[1]. As shown in the Figure 6.6 and Figure 6.7, we can see that the Ping Flood attack can increase the delta value of MIB II object *icmpInEchos* dramatically. Therefore, we can set up a rule that if the delta value of *icmpInEchos* has been increased over the normal value by a certain percentage, then Ping Flood attack occurs. Similarly, for example, we can also use the MIB II object *icmpInEchoReps* to detect the SMURF[2] [6] attack. With the knowledge of the pattern of SMURF

[1] Ping flood attack is that the attacker sends a high amount of icmp echo request packets to the victim in order to make the victim inoperable.

[2] The attacker sends a stream of icmp echo requests to the broadcast address of many subnets with the source address spoofed to be that of the intended victim. The attack results in a large, continuous stream of echo replies that flood the victim.

attack, we can select object *icmpInEchoReps* as the key MIB II variable for detecting the SMURF attack. Object *icmpInEchoReps* represents the total number of inbound icmp echo replies. Therefore, the pattern of SMURF attack by using this object can be phrased as: If the delta value of *icmpInEchoReps* is more than the normal maximum delta value by a certain percentage, e.g., 30%, then SMURF attack occurs.

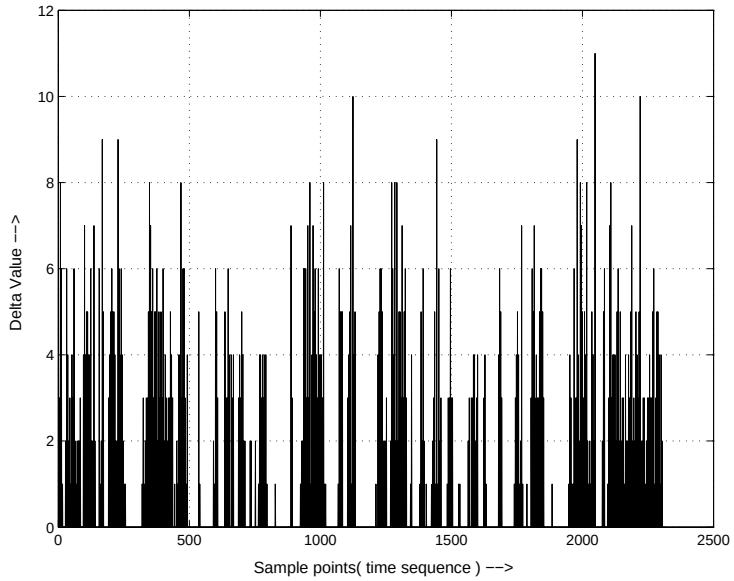


Figure 6.6. *icmpInEchos* under normal condition

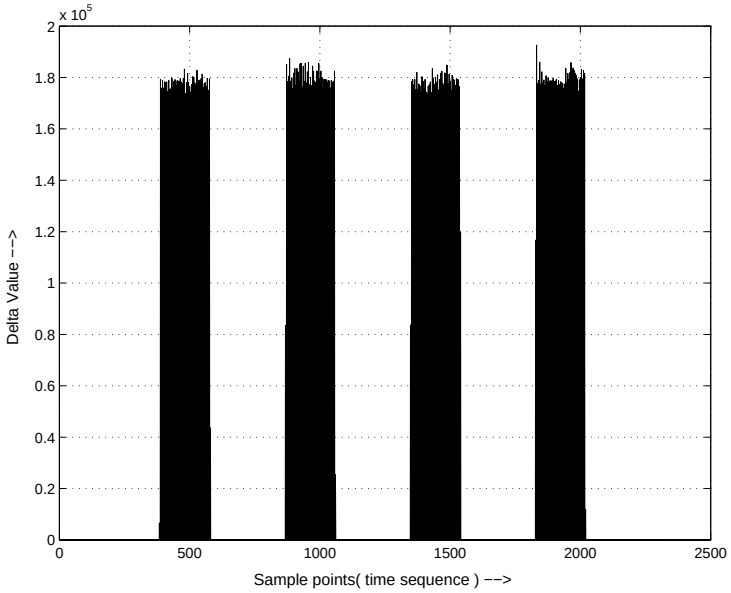


Figure 6.7. *icmpInEchos* under Ping Flood attack

Figure 6.8 to Figure 6.11 show the changes of two MIB II objects in UDP group, i.e., *udpInDatagrams* and *udpInErrors*, observed under the normal condition and the intrusion of UDP Flood[3] respectively. *udpInDatagrams* counts the total number of UDP datagrams delivered to UDP users. *udpInErrors* refers to the number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination host. Figure 6.8 and Figure 6.9 show that the delta value of MIB II object *udpInDatagrams* changes a lot during the intrusion period, comparing normal and intrusion conditions. From Figure 6.10, we can see that the delta value of object *udpInErrors* is zero in the normal condition. Figure 6.11 shows the dramatic changes of the delta value of object *udpInErrors* under the UDP Flood. It clearly indicates a huge amount of UDP packets were sent to the target during the intrusion. The reason why the *udpInErrors* instead of *udpInDatagrams* has a more drastic value changes is that the attacking tool generated many incorrect UDP packets which were counted by *udpInErrors*. From these figures, we can see that some MIB II objects can indicate the corresponding abnormal events clearly according to their definitions. Therefore, these objects can act as an audit source for misuse detection.

[3] The attacker sends a huge amount of UDP traffic to the UDP port(s) of the victim in order to make the corresponding port inoperable.

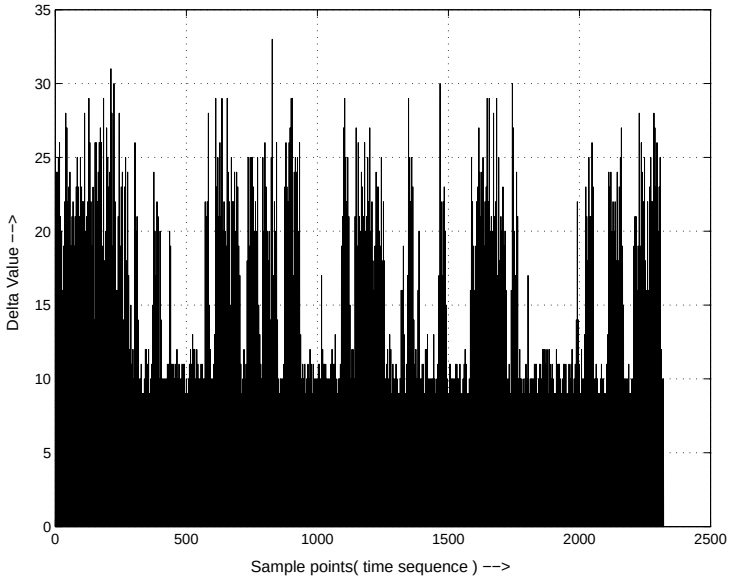


Figure 6.8. *udpInDatagrams* under normal condition

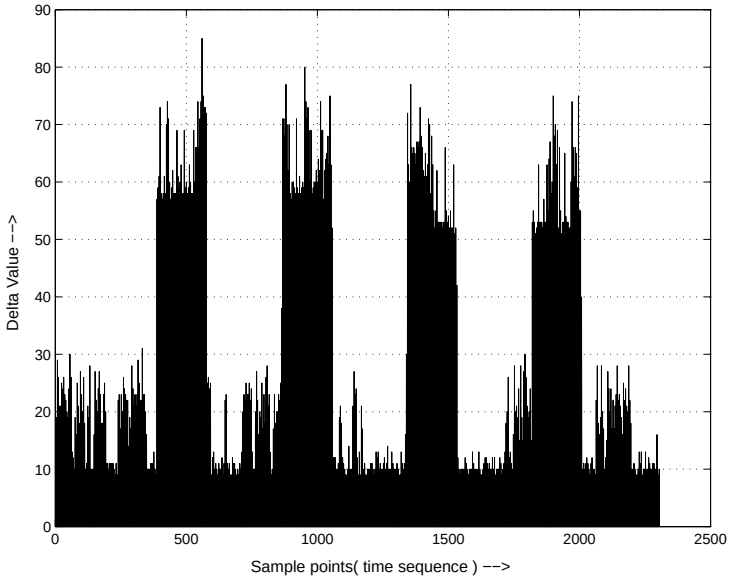


Figure 6.9. *udpInDatagrams* under UDP Flood attack

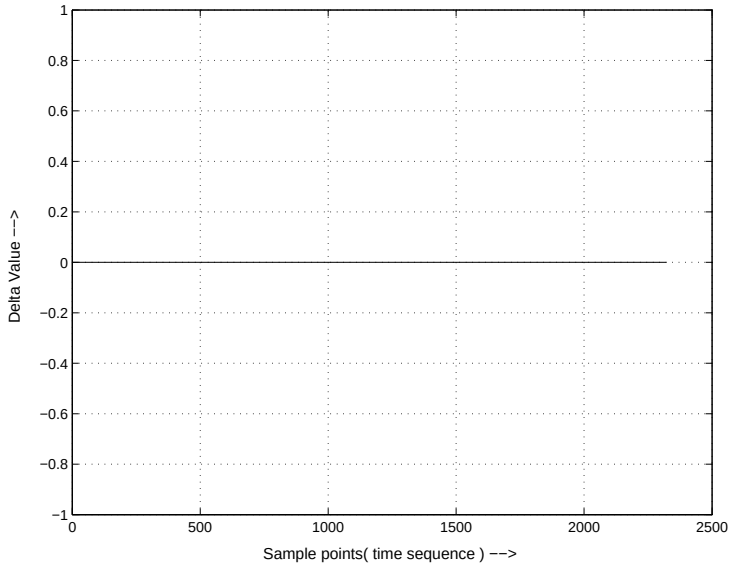


Figure 6.10. *udpInErrors* under normal condition

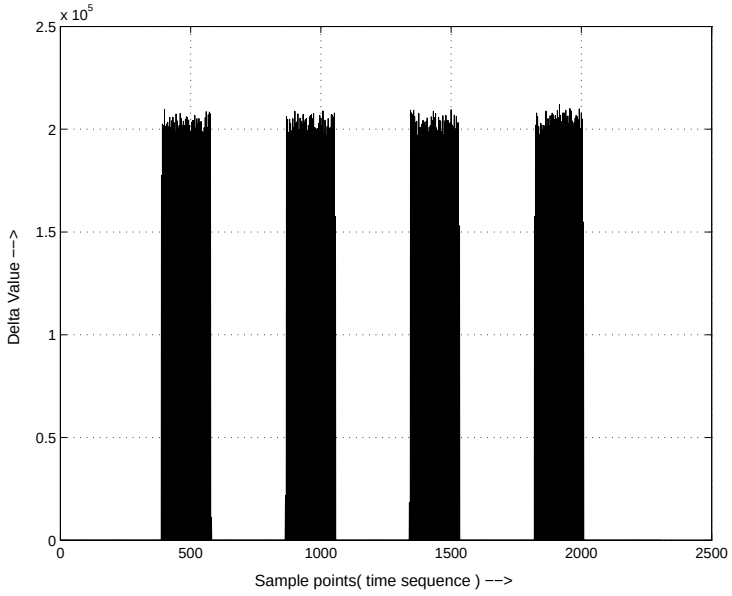


Figure 6.11. *udpInErrors* under UDP Flood attack

In the experiments, we launched several denial-of-service(DoS) attacks, e.g., Syn Flood[4] and probing attacks, e.g., NMAP[5] [11], and evaluated the performance of Misuse Detection Module as shown in the Table 6.1

Table 6.1. Misuse Detections by MIB II-based ID Model

Type of Intrusions	TP[1]	FP[2]	Key MIB II Objects
Ping Flood	98.86%	0.039%	*icmpInEchos*
Syn Flood	97.83%	0.535%	*tcpInSegs* *tcpActiveOpens* *tcpAttemptFails*
UDP Flood	98.51%	0.348%	*udpInDatagrams* *udpInErrors*
NMAP(TCP & Syn)	97.59%	0.372%	*tcpInSegs*,*tcpOutRsts*
NMAP(UDP)	99.89%	0.273%	*udpInErrors* *udpNoPorts* *icmpOutDestUnreachs*

[1] TP: True Positive Rate, [2] FP: False Positive Rate

4.2.2 Anomaly Detection. Although we can directly use some MIB II variables to represent some certain known intrusion patterns and compose the corresponding detection rules, an anomaly detection module is still desired in order to detect the new intrusions whose signatures are unknown yet. Therefore, it is necessary to combine the corresponding MIB II objects according to network layers and protocols, e.g., using the sub-modules, in order to detect intrusions more precisely.

In evaluating the ID model, we ran several DoS attacks using the TFN2K attacking tool [6]. The MIB II variables were collected at the target, and the sample rate and period were the same as that in the normal runs. We use Mix Flood, i.e., a combination of Ping Flood, UDP Flood and Syn Flood, as an example here to discuss our results.

[4]The attacker sends a huge stream of SYN packets to a port(s) of the victim without finishing the TCP connections. Such a big amount of half open TCP connections can block the service(s) associated with the port(s) of the victim.

[5]A port scanning tool that can find out which port is opening and what the service is running on the port on the victim host.

Figure 6.12 to Figure 6.18 show some information on the Euclidean distance of four ID sub-modules, i.e., *ICMP_In*, *ICMP_Out*, *UDP_In_Error* and *TCP_In*, under the normal condition and the intrusion respectively. Figure 6.12 shows the normal maximum and minimum Euclidean distance of *ICMP_In* sub-module. From Figure 6.13, we can see that the Mix Flood attack has caused the Euclidean distance of *ICMP_In* to increase a lot over its maximum normal profile. The same results can be seen by comparing the Euclidean distance of sub ID model *ICMP_Out* under normal condition and under the attack respectively, i.e., Figure 6.14 and Figure 6.15. Figure 6.16 shows the normal profile of maximum and minimum Euclidean distance of sub-module *UDP_In_Error*. Figure 6.17 shows clearly that the Euclidean distance of *UDP_In_Error* has been increased dramatically as the Mix flood attack occurred. Similarly, Figure 6.18 shows the normal maximum and minimum Euclidean distance of module *TCP_In*. Figure 6.19 shows that the module *TCP_In* has very large Euclidean distance under the intrusion. Comparing the figures of the normal maximum and minimum Euclidean distance with the figures of Euclidean distance under the intrusion, we can see that, these three sub-ID modules can detect the anomaly based on the pre-set thresholds. In this experiment, we set tolerance deviation margin α as 30%.

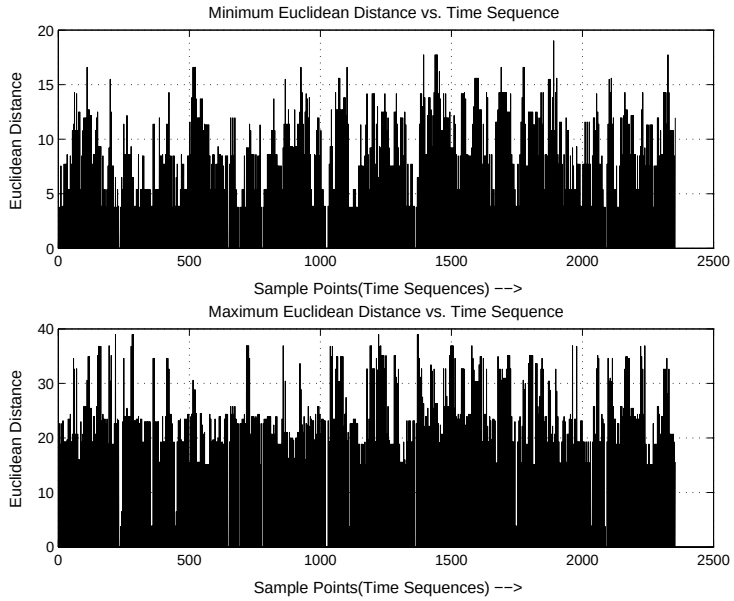


Figure 6.12. *ICMP_In* ID sub-module under normal condition

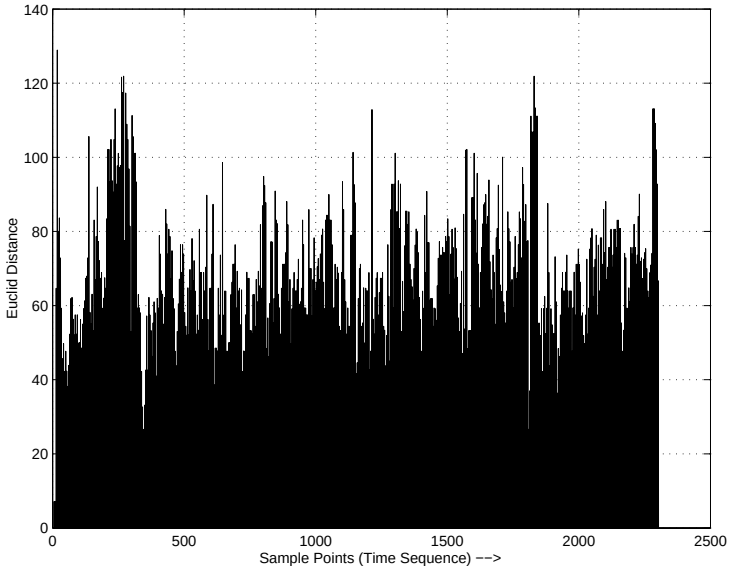


Figure 6.13. *ICMP_In* ID sub-module under Mix Flood attack

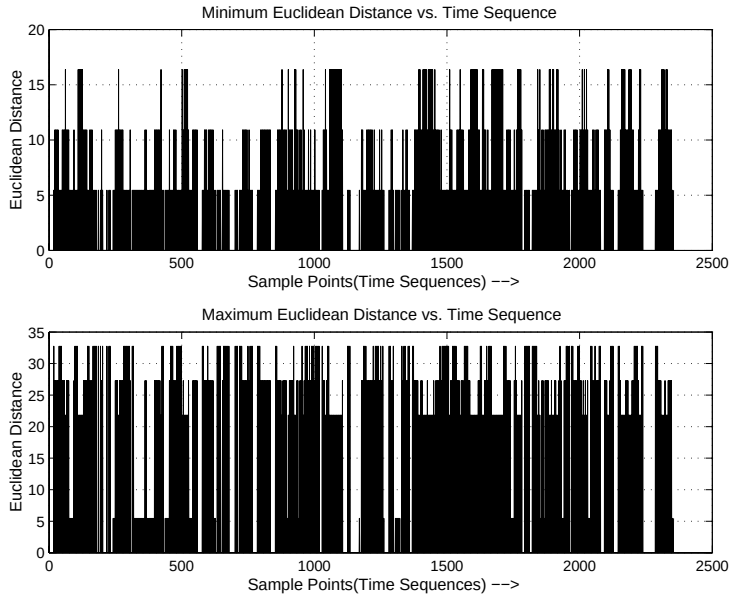


Figure 6.14. *ICMP_Out* ID sub-module under normal condition

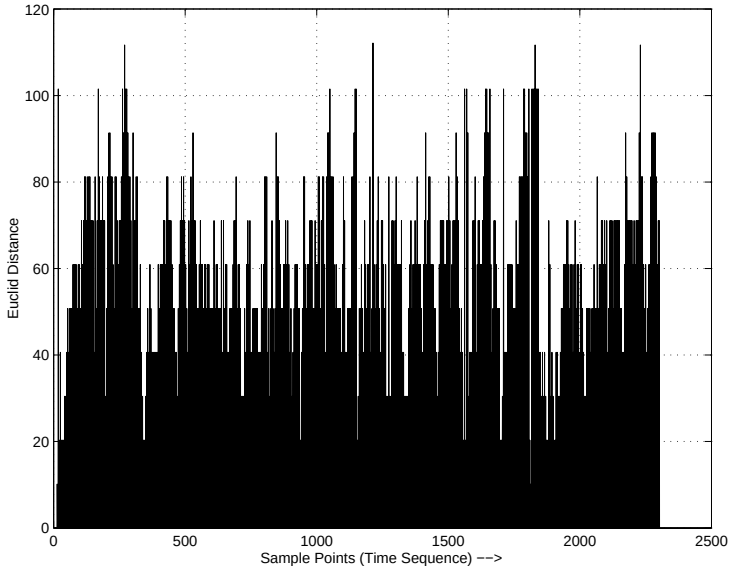


Figure 6.15. *ICMP_Out* ID sub-module under Mix Flood attack

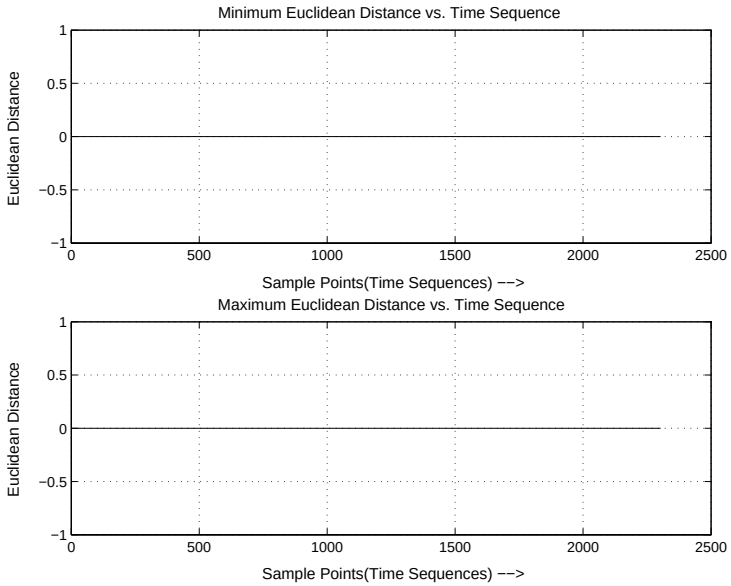


Figure 6.16. *UDP_In_Error* ID sub-module under normal condition

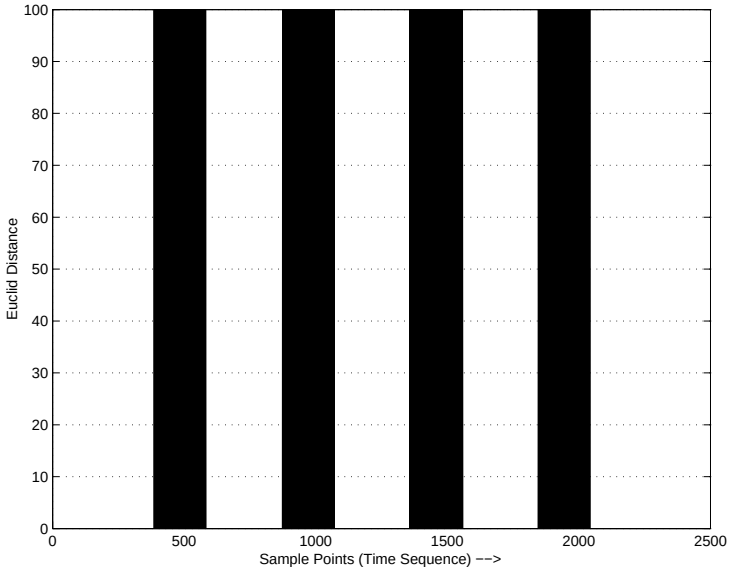


Figure 6.17. *UDP_In_Error* ID sub-module under Mix Flood attack

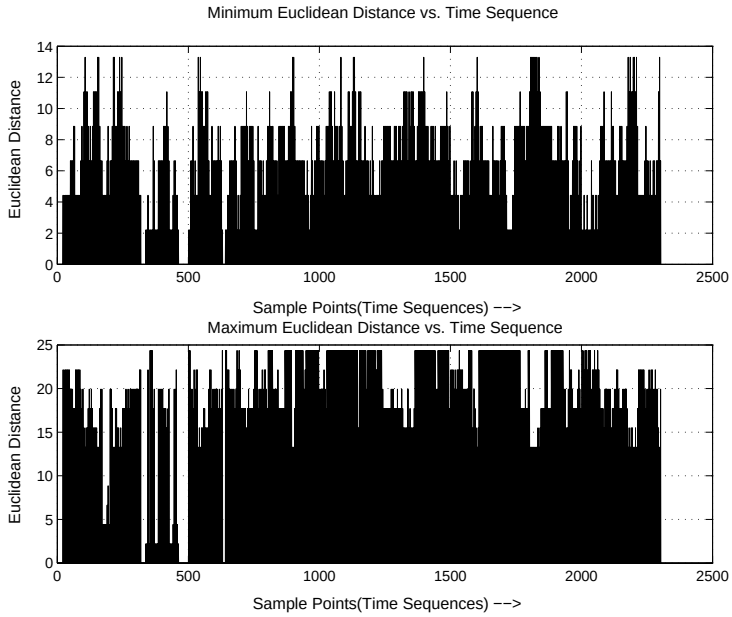


Figure 6.18. *TCP_In* ID sub-module under normal condition

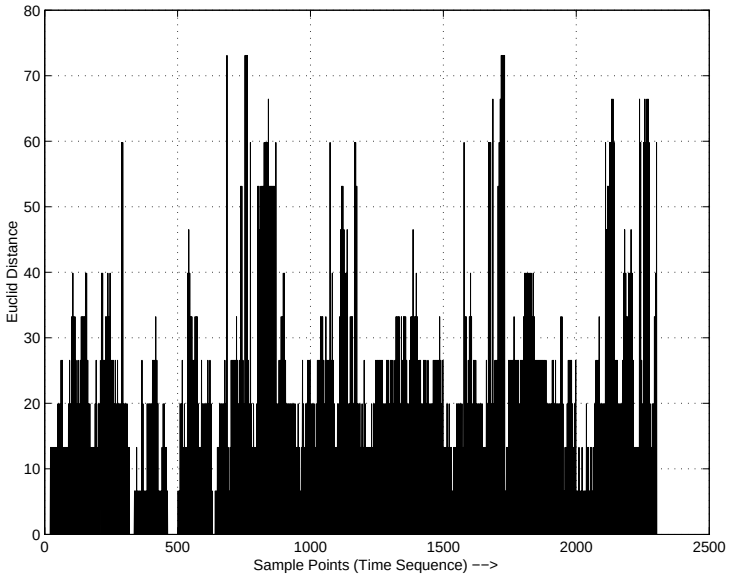


Figure 6.19. *TCP_In* ID sub-module under Mix Flood attack

In each experiment, we have 2400 samples with 3-second sample interval. In the performance evaluation, we use our ID model to identify the anomaly sample point. Table 6.2 shows the performance results of our ID model in the experiments. It also shows which ID sub-module is used to detect each intrusion. From the table, we can see that our ID model has a very high detection accuracy and relatively low false alarm rate on some intrusions such as Ping Flood, Syn Flood, Targa3[6], UDP Flood and Mix Flood. It also shows that the right sub-modules were able to detect the intrusions that had influences on the corresponding protocols. For example, because Ping Flood is to send a huge amount of *pings*, i.e., *icmpechos* to the victim host, the victim host will not only receive a large number of *icmp echo requests*, but sends out many *icmp echo replies* as well. Therefore, we would expect the sub-modules *ICMP_In* and *ICMP_Out* to detect such anomaly. As shown in the results, these two sub-modules *ICMP_In* and *ICMP_Out* have detected the Ping Flood successfully. In TearDrop[7], the detection rate is not as high as the others. This is due to the sample rate of the ID model. Generally, when an intrusion occurs in the network, the deviation on the MIB data measures will last for a while due to the fact that the intrusion effects have the "build-up" and "residual" stages. However, the relatively large sample interval with regard to the short duration of TearDrop makes these stages less visible and affects the detection performance.

5. Discussion

From our experiments, we can see that our MIB II-based ID anomaly detection module can detect some traffic-based attacks such as distributed denial-of-service (DDoS) and DoS very well. In addition, with the help of Misuse Detection module, the probing intrusion can also be detected accurately. These two modules are actually complementary to each other. While misuse detection module is intended to detect the known intrusions efficiently, MIB II objects, due to their definitions, cannot represent all the known intrusion signatures. Therefore, there is a need to find another approach, i.e., using Anomaly Detection module, to comprehensively detect the intrusions based on layers and protocols.

[6]The attacker sends out the incorrect and uncommon packets, such as the invalid fragmentation, packet size, header values, etc. When the TCP/IP stack receives these weird packets, the kernel has to allocate the sources to handle these packets. A huge amount of malformed packets can make the system crash because of the exhausted resources.

[7]The attacker sends out many IP packets with overlapping IP fragments. If the implementations of the IP fragmentation re-assembly code cannot properly handle with overlapping IP fragments, the system will be crashed.

Table 6.2. Anomaly Detections by MIB II-based ID Model

Type of Intrusions	TP[1]	FP[2]	Detected by ID Sub-Module
Ping Flood	95.48%	0.217%	$IP\_In, ICMP\_In$ $ICMP\_Out$
Syn Flood	96.15%	0.767%	IP_In, TCP_In
Targa3	96.78%	0.375%	IP_In, IP_Other TCP_In_Error UDP_In_Error
UDP Flood	97.95%	0.412%	IP_In, UDP_In UDP_In_Error
MixFlood	97.85%	0.492%	$ICMP\_In, ICMP\_Out$ TCP_In, UDP_In_Error
TearDrop[8]	60.78%	0.327%	IP_In, IP_Other

[1] TP: True Positive Rate, [2] FP: False Positive Rate

However, due to the limitation of MIB II objects, our ID model currently cannot detect those non-traffic based attacks such as illegal-remote-root-access attacks since these attacks generally do not depend on the large traffic to compromise the target, yet, most of MIB II objects are related to the traffic information. In addition, due to the size of our test bed, we were unable to test our MIB-based ID model under more realistic and complicated network traffic conditions. This is a limitation in our experiments.

Our ID model has shown some potential benefits. First of all, our ID model can improve the efficiency of the IDS. Since most of the current IDSs use tcpdump [12] for network intrusion detection, for those traffic-based attacks, such as DoS, the IDS engine needs to calculate the related statistics based on the raw tcpdump data in order to detect the abnormal traffic. Such approach is inefficient in terms of space and time. In addition, to select and construct the statistical features to detect a wide range of attacks is very challenging. Network Management System (NMS) has already provided us with a comprehensive set of variables about network activities that are very valuable for anomaly detection. For example, MIB II has many objects representing the traffic information and configuration of the network and hosts. As our ID model shows, by taking advantage of the information from the MIB II, we can do anomaly detection on the traffic-based intrusions directly. We believe that using the information in NMS can ease the workload of the IDS in

terms of information collection and calculation. One of the potential integration is that the ID agents based on our MIB-based ID model can be deployed in the network and collaborate with the tcpdump-based IDS for intrusion detection. A security manager is necessary for correlating the alarms sent by the ID agents and the IDS. We believe that such integration will provide a more comprehensive solution to network intrusion detection.

Another potential application of our MIB-based ID model is to do proactive detection of the attacks such as DDoS [2]. We can install security agents based on our MIB-based ID model on the slave machines which monitor the anomaly activities on the host. When the the slave machine sends out attacking packets, the anomalous outbound attacking traffic can be detected by the ID model and alarms can be sent out before the attacking traffic reaches the breaking point to compromise the target.

On the whole, through this study, we have learned that, although MIB II objects have some limitations, the MIB II-based ID model can provide us an effective approach in detecting a certain category of attacks, i.e., traffic-based intrusions. In addition, we can also take advantage of such MIB-based ID model in the integration of the NMS and IDS in order to achieve a more comprehensive and efficient security system.

6. Related Work

Anomaly detection has been an important subject in the IDS. Many different approaches have been proposed and implemented. In the early approaches [22, 13, 1], a normal profile is usually based on statistical measures of the system features, e.g., the CPU usage, etc. Recently, there have been several studies in using learning-based approach to build anomaly detection models for privileged programs [7, 26, 14, 8]. Warrender et al. [26] showed that a number of machine-learning approaches, e.g., rule induction, Hidden Markov Model, etc., can be used to learn concise and generalizable representation of the "self" identity of a system program [7]. EMERALD [19] has an enterprise-wide correlator which combines the alarms from the distributed detectors. Its anomaly detection model is based on the statistical approach for monitoring the network traffic. While these systems all have some degree of success, most of them are focused on the host-based anomaly detection by investigating the system programs. In addition, few of the above systems has taken advantage of MIB II objects as an audit source for the network-based anomaly detection.

In NMS, the work concentrates on network performance modeling for the purpose of detecting degradation and faults [9, 10, 25, 17]. How-

ever, the context of intrusion detection is more subtle than the general fault detection in NMS. In [25], MIB II variables are used for the fault detection. Their approach uses only five MIB II objects in the Interface group and IP group and focuses on the faults happened on the IP layer. However, this approach is not appropriate for the intrusion detection since such a few MIBs are not comprehensive enough to represent the behavior of normal network activities.

7. Conclusions and Future Work

In this paper, we proposed an approach for intrusion detection using MIB II-based ID model. We apply data mining, information theory, and clustering technique to develop our ID model. The results of experiments showed that our ID model performs well in detecting probing attacks and some traffic-based intrusions such as DoS and DDoS. The promising results also show that our ID model can play an important role in the the integration of the IDS and NMS.

Further work is needed to validate our methodology under more realistic and complicated traffic conditions. Future efforts will be also focused on anomaly detection on intrusions with slower traffic and stealth attacks. We will refine our modeling techniques in selecting the appropriate MIB variables and time window as well as correlation techniques between ID sub-modules at different protocol layers. We will also further study the integration of the IDS and NMS.

Acknowledgments

This research is supported in part by a grant from DARPA (F30602-00-1-0603) and a graduate fellowship sponsored by Aprisma Management Technologies).

References

[1] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (NIDES): A summary. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, California, May 1995.

[2] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, and R. K. Mehra. Proactive detection of distributed denial of service attacks using MIB traffic variables - a feasibility study. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, May 2001.

[3] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun. On the nonstationarity of internet traffic. In *Proceedings of ACM SIGMETRICS '01*, pages 102–112, 2001.

[4] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: Recent traffic measurements from an internet b ackbone. In *Proceedings of Inet '98*. The Internet Society, July 1998.

[5] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[6] P.J. Criscuolo. Distributed denial of service - trin00, tribe flood network, tribe flood network 2000, and stacheldraht. Technical Report CIAC-2319, Department of Energy - CIAC (Computer Incident Advisory Capability), February 2000.

[7] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[8] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

[9] J. L. Hellerstein, F. Zhang, and P. Shahabuddin. An approach to predictive detection for service management. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, May 1999.

[10] L. L. Ho, D. J. Cavuto, S. Papavassiliou, M. Z. Hasan, F. E. Feather, and A. G. Zawadzki. Adaptive network/service fault detection in transaction-oriented wide area networks. In *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management*, May 1999.

[11] NMAP Homepage. http://www.insecure.org/nmap/index.html, 2001.

[12] V. Jacobson, C. Leres, and S. McCanne. *tcpdump*. available via anonymous ftp to ftp.ee.lbl.gov, June 1989.

[13] Los Alamos National Laboratory. Wisdom and sense guidebook. Los Alamos National Laboratory.

[14] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, January 1998.

[15] W. Lee and D. Xiang. Information-theroetic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, May 2001.

[16] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real-time intrusion detection expert system (IDES) - final technical report. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

[17] R. A. Maxion. A case study of ethernet anomalies in a distributed computing environment. *IEEE Transactions on Reliability*, 39(4), October 1990.

[18] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets: MIB-ii. RFC1213, 1991.

[19] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference*, Baltimore MD, October 1997.

[20] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[21] G. Shipley and P. Mueller. Dragon claws its way to the top. In *Network Computing. TechWeb*, August 2001.

[22] S. E. Smaha. Haystack: An intrusion detection system. In *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, 1988.

[23] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addison-Wesley, 1999.

[24] SunSoft. *SunSHIELD Basic Security Module Guide.* SunSoft, Mountain View, CA, 1995.

[25] M. Thottan and C. Ji. Proactive anomaly detection using distributed intelligent agents. *IEEE Network, Special Issue on Network Management*, April 1998.

[26] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.

Chapter 7

ADAPTIVE MODEL GENERATION

An Architecture for Deployment of Data Mining-based Intrusion Detection Systems

Andrew Honig
arh@cs.columbia.edu
Department of Computer Science
Columbia University

Andrew Howard
Department of Computer Science
Columbia University
ahoward@cs.columbia.edu

Eleazar Eskin
Department of Computer Science
Columbia University
eeskin@cs.columbia.edu

Sal Stolfo
Department of Computer Science
Columbia University
sal@cs.columbia.edu

Abstract Data mining-based intrusion detection systems (IDSs) have significant advantages over signature-based IDSs since they are designed to generalize models of network audit data to detect new attacks. However, data mining-based IDSs are difficult to deploy in practice due to the complexity of collecting and managing audit data to train the data mining-based detection models. In this paper, we present Adaptive Model Generation (AMG), a real time architecture for implementing

data mining-based intrusion detection systems. This architecture solves the problems associated with data mining-based IDSs by automating the collection of data, the generation and deployment of detection models, and the real-time evaluation of data. It is a distributed system with general classes of components that can be easily changed within the framework of the system. We also present specific examples of system components including auditing sub-systems, model generators for misuse detection and anomaly detection, and support for visualization and correlation of multiple audit sources.

Keywords: Intrusion Detection, Data Mining, Anomaly Detection.

1. Introduction

As sensitive information is increasingly being stored and manipulated on networked systems, the security of these networks and systems has become an extremely important issue. Intrusion detection systems (IDSs) are an integral part of any complete security package of a modern, well managed network system. An IDS detects intrusions by monitoring a network or system and analyzing an audit stream collected from the network or system to look for clues of malicious behavior.

The most widely used and commercially available IDSs are signature-based systems. A signature-based system matches features observed from the audit stream to a set of signatures hand crafted by experts and stored in a signature database. Signature-based methods have some inherent limitations. The most significant is that a signature-based method is designed to only detect attacks for which it contains a signature in the database. In addition to the expense in time and human expertise of manually encoding a signature for each and every known attack, the signature-based methods therefore can not detect unknown attacks since there is no signature in the database for them. It is these unknown attacks that are typically the most dangerous because the system is completely vulnerable to them.

Data mining-based methods are another paradigm for building intrusion detection systems. The main advantage of these methods is that they leverage the generalization ability of data mining methods and in order to detect new and unknown attacks. A data mining-based IDS uses machine learning and data mining algorithms on a large set of system audit data to build detection models. These models have been proven to be very effective Lee et al., 1999; Warrender et al., 1999. These algorithms are generally classified as either misuse detection or anomaly detection. Misuse detection algorithms learn how to classify normal and attack data from a set of training data which contains both

labeled attack and normal data Lee et al., 1999. Anomaly detection algorithms learn a model of normal activity by training on a set of normal data. Anomaly detection algorithms then classify as an attack activity that diverges from this normal pattern based on the assumption that attacks have much different patterns than do normal activity. In this way new unknown attacks can be detected Denning, 1987; Forrest et al., 1996; Warrender et al., 1999; Ghosh and Schwartzbard, 1999.

However, data mining-based IDSs have their own disadvantages. Data to train the models is costly to generate. The data must be collected from a raw audit stream and translated into a form suitable for training. In addition, for misuse detection, each instance of data must be labeled either normal or attack. In the case of anomaly detection each instance of data must be verified to be normal network activity. Since data mining-based IDSs in general do not perform well when trained in one environment and deployed in another, this process of preparing the data must be repeated at every deployment of data mining-based IDS system. Furthermore, for each type of audit data that is to be examined (network packets, host event logs, process traces, etc.) the process of preparing the data needs to be repeated as well. Because of the large volumes of data that needs to be prepared, the deployment of a data mining-based IDS system involves a tremendous amount of manual effort.

Many of parts of these manual processes can be automated, including the collection and aggregation of the data and translating it into a form appropriate for training the data mining-based detection models. In addition, many of these processes are the same across types of audit data. Some of the processes still require some manual intervention such as labeling the data, but even these can be semi-automated.

In this paper we present Adaptive Model Generation (AMG), an architecture to automate the processes of data collection, model generation and data analysis. The AMG system solves many of the practical problems associated with the deployment of data mining-based IDSs.

AMG is an architecture to automate many of the critical processes in the deployment and operation of real time data mining-based intrusion detection systems. AMG abstracts various parts of the data mining-based IDS process and formalizes the automation of this process. This includes abstracting the processes of data collection, data aggregation, detection model evaluation, detection model generation, and model distribution. AMG uses a general XML-based data and model representation scheme that facilitates the translation of data from what is collected at the audit stream to the form necessary for generation of detection models. AMG provides a tool for semi-automating the pro-

cess of labeling training data and also leverages a new class of intrusion detection algorithms called unsupervised anomaly detection (UAD) algorithms which permits the training of detection models over unlabeled data. These algorithms detect intrusions buried within unlabeled data and can tolerate a small amount of intrusion data (noise) in the training data Eskin, 2000; Portnoy et al., 2001; Eskin et al., 2002.

The AMG architecture consists of several different types of component sub-systems. Real time components such as sensors and detectors collect information from the monitored system and detect intrusions in real time. The centerpiece of the data management capabilities of AMG is a data warehouse which stores the data collected by all of the sensors in a network. Model generators access this data and train data mining-based detection models using this data. Model distributors transfer the models to the detectors. Finally analysis engines provide data analysis capabilities such as visualization and forensic analysis.

More specifically, AMG has the following major capabilities:

- Automated Data Collection: Data is collected by sensors and automatically sent to detectors for classification and to the data warehouse for aggregation and use in training models.
- Data Warehousing: AMG includes a data warehouse component that stores data from all sensors. This data is used for training detection models but can used to support various types of data analysis such as forensic analysis.
- Automated Model Generation: Detection models are trained from data stored in the data warehouse. The process for converting the data into the appropriate form for training is fully automated.
- Heterogeneous Data Support: Data from different types of audit sources is easily handled in the framework of our system. Any form of data can be represented using a general XML-based language.
- Automated Model Distribution: Once a new model is generated it is deployed to all of the detectors that subscribe to the particular detection models that have been generated.
- Data Analysis Capabilities (Including Forensics): AMG enables evaluation of archival records stored within the data warehouse to search for intrusions.
- Visualization Support: AMG includes generic visualization components that allow in a consistent fashion the visualization of data from different sources.

- Correlation Support: Since data from multiple sensors is stored in the data warehouse AMG can perform analysis over the data from multiple sources and to train detection models which examine audit streams from multiple sensors.

The paper is organized as follows. We first describe in detail the components that make up the AMG system. We then describe the main capabilities of the AMG system. Then we describe the kinds of data mining algorithms that the AMG system supports. Finally we give examples of model generation algorithms and sensors that are implemented through the AMG framework. Note that in most cases, the sensors and detectors presented in this paper have been described in detail in other prior publications. In this paper we present the details of how the sensor or model is easily integrated into the AMG system in a "plug and play" fashion.

2. Components of Adaptive Model Generation

The adaptive model generation system automates the processes of data collection, data management, detection model generation and distribution, and provides various capabilities for data analysis. The challenge in automating these processes is the need to support different types of data and different types of detection models. In a typical network environment there are many different audit streams that are useful for detecting intrusions. For example, such data includes the packets passing over the network (header values, payload features, or combinations thereof), the system logs on the hosts in the network, and the system calls of processes on these machines. These types of data have fundamentally different properties. In addition, detection models can also vary greatly. The most widely used detection model is a signature-based system while data mining-based approaches use methods such as probabilistic models Eskin et al., 2001 and support vector machines Cristianini and Shawe-Taylor, 2000. The methods for building these detection models as well as executing them in real time vary greatly for each type of detection model.

AMG is a system framework and architecture that can handle virtually any data type and detection model. The AMG system consists of four types of components: real time components which include sensors and detectors, a data warehouse component, detection model management components which include adaptive model generators and detection model distributors and data analysis components which includes components for visualization, forensics, labeling data, correlation and extracting information from multiple records.

Sensors gather information from an environment and send that data to the data warehouse. The data in the data warehouse is accessed by the detection model generators which generate models that classify activity as either malicious or normal. Once a model is created it is stored in the data warehouse. A model distributor deploys that model to a real-time detector. The detector receives the detection model from the detection model distributor and also receives the audit data from the sensor. It then uses the model to evaluate the audit data from the sensor to detect intrusions.

The data analysis engines retrieve data from the data warehouse. The use of the data varies depending on the particular data analysis engine. The results of an analysis are either stored in the data warehouse, or displayed directly to the user. Data analysis engines allow the adaptive model generation system to implement many systems that are helpful in the deployment of an intrusion detection system. New types of data analysis engines can easily be incorporated into the AMG system. The complete architecture of the adaptive model generation system is displayed in Figure 7.1.

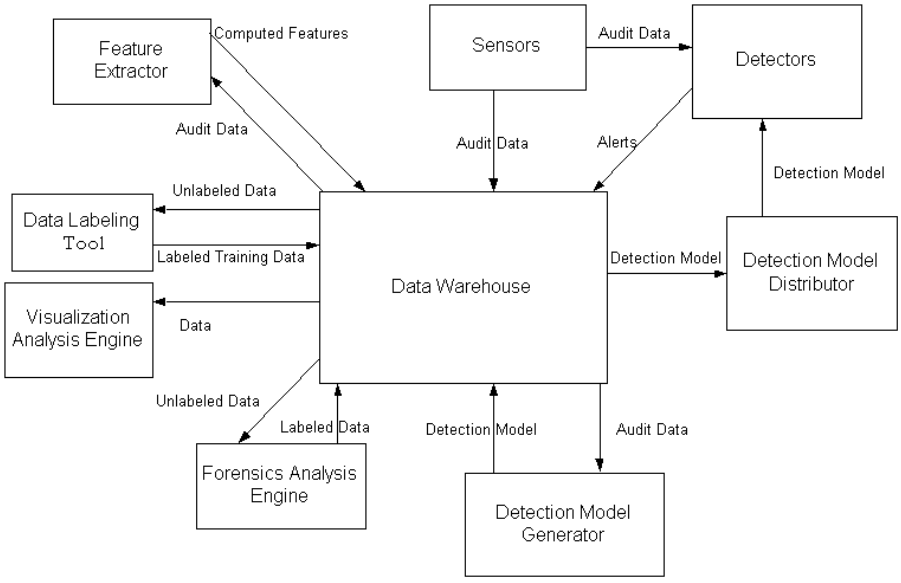


Figure 7.1. The AMG System Architecture

The adaptive model generation system uses a distributed architecture that consists of autonomous components. By making these components independent, linking them only with a communication protocol with very

loose format restrictions, we allow any component conforming to that protocol to interface with our system. The communication protocol uses an XML markup language which is similar to the IDMEF specification in the IETF Internet Engineering Task Force, 2000. The system can easily be adapted to the IDMEF format or any other language format such as CIDF Staniford-Chen et al., 1998.

2.1 Real Time Components

The AMG system uses two basic components to provide real time detection capabilities: sensors and detectors. A sensor is a system that collects data from an audit stream and formats the data into the appropriate form using the XML markup language. This data is then sent to a detector which uses a detection model to determine whether or not the data corresponds to an attack. If the data corresponds to an attack, the detector generates an alert.

In this framework, a traditional IDS system consists of a sensor and a detector which in most cases are integrated together. For example, in our framework, the Snort Roesch, 1999 system contains a sensor which monitors packets and a detector which evaluates signatures over the features extracted from the packets. Our system separates these two components providing a means of managing the overall computation of IDS in reconfigurable architectures. For example, if the detection model evaluation for a particular audit stream is a computational bottleneck, it can be easily distributed to multiple machines.

The advantage of the AMG system over traditional system architectures is the integration of the sensors and detectors with other analysis and distribution sub-systems. In addition to sending the data to a detector, the sensor also sends the data to a data warehouse which aggregates and stores the data. The detector retrieves its detection models from the model distributors. These detection models are created by other components of the AMG system.

Sensors Sensors are lightweight processes that gather information from a real-time audit stream, format that information into an XML representation and then send that to the detectors for real time intrusion detection and to the data warehouse for aggregation of data and storage. The sensors can gather information from any source. For every different audit stream of data a sensor can be implemented to monitor and collect the data.

Typically, there are two major classes of sensors, network sensors and host sensors. Network sensors monitor the packets moving through the network. We have implemented a network sensor HAUNTDios et al.,

2001 which is used to monitor network activity. The system listens to network traffic and reassembles the packets in order to create data records which summarize connections. A network sensor in AMG can be created from existing network IDS systems such as Snort Roesch, 1999 and Bro Paxson, 1998 simply by wrapping the component in the XML-based form required for AMG.

In most circumstances there are multiple sensors for each host. This is because a host usually has several different streams of information that are useful for detecting intrusions. AMG has been implemented with a variety of sensors for both Windows and Linux systems Hershkop et al., 2001. On the Windows system we have developed Windows event log sensors, software wrappers, netstat sensors, and registry sensors. There are three Windows event log sensors which take information from the application, security, and event logs on the Windows system respectively. Software wrappers are sensors that gather information about system calls. Netstat sensors use the netstat tool that gathers information about network connections on the host. data. Registry sensors monitor the activity of the windows registry when applications are run on the host.

The Linux sensors built into AMG include process sensors, network connection sensors, resource sensors, and software wrappers. Process sensors use the `/proc` virtual file-system to gather information about each running process. Network connection sensors collect information about network connections being made to and from the host machine. Resource sensors gather information about CPU and memory usage on a machine. The software wrappers for Linux systems are the same as for Windows, monitoring system calls made by a process. Details of all the host based sensors can be found in Hershkop et al., 2001.

The sensors themselves are constructed from two major components, the basic auditing module (BAM) and the communication engine. The BAM is the component that extracts the information from the audited source. The communication engine encodes the data and sends it to the data warehouse for storage.

The BAM needs a mechanism to gather the data. This is done differently for each stream of data. Therefore a separate BAM is needed for each source of data that the system monitors. Packet sniffers and Win32 hooks are two examples of ways to tap into the data stream and gather data. The BAM can be seen as an interface to the system being monitored. It hooks into the data stream and has the ability to transfer the information from that data stream to the communication engine. Therefore this system can function without any knowledge of how the sensor work. This makes the system very flexible with respect to sensors.

The communication engine takes the data from the BAM, encodes the data into the AMG XML format and then sends that data to the data warehouse. Along with the data itself the BAM sends the meta data, such as variable types, to the communication engine. This is the information that the communication engine needs to encode the information. The communication engine also needs to know the type of sensor in order to send the data to the right place in the data warehouse. This is specified when the connection is made to the communication engine. An example of a record being generated from the RAD sensor, a sensor that monitors accesses to the Windows registry, can be seen below.

Raw data being read by sensor:

```
Process:  IEXPLORE
Key:  HKCR\Applications\notepad.exe\shell
Query:  Openkey
Result:  Success
Response:  0xE22FC4C0
```

The sensor sends the following Data sent to Communication Engine:

```
Process:  IEXPLORE
Key:  HKCR\Applications\notepad.exe\shell
Query:  Openkey
Result:  Success
Response:  0xE22FC4C0
Time:  Tue Jul 31 14:43:20 EDT 2001
ProcQuery:  1263.4353
KeyProc:  6784.9363
QueryKey:  6621.3521
KeyResponse:  4510.2431
KeyResVal:  8743.3245
```

Note that the combination features are stored as hashes, not the actual values. This is done for efficiency and convenience purposes and implemented in the sensor. The communication engine then encodes the record as the following.

```
<rec> <process> IEXPLORE </process>
<key> HKCR\Applications\notepad.exe\shell </key>
<query> Openkey </query>
<result> Success </result>
<response> 0xE22FC4C0 </response>
<procQuery> 1263.4353 </procQuery>
<keyProc> 6784.9363 <keyProc>
<queryKey> 6621.3521 </queryKey>
```

```
<keyResponse> 4510.2431 </keyResponse>
<keyResVal> 8743.3245 </keyResVal>
</rec>
```

Detectors Detectors analyze audit stream data collected from a sensor and detect intrusions by using a detection model. A detector performs *model evaluation* over each record from the sensor. The way a specific detector works depends on the type of model being evaluated. Each different model type has a different detector that implements model evaluation for that model type.

The detector can be viewed as a function that takes as input a data record and outputs an alert if the data is determined to correspond to an intrusion. An example of a detection model type is a signature-based model, which is the algorithm most widely used in commercial intrusion detection systems. A signature-based detection model simply contains a set of "signatures" which correspond to known attack types. Model evaluation consists of matching each signature in the model to a data record. If any of the signatures match, the detector generates an alert.

In our framework, more sophisticated model types can be used as well including data mining-based models that use decision trees, probabilistic models and support vector machines. In the case of a decision tree, the detection model would contain an encoding of a decision tree. The detector would take this model and evaluate the detection model on a record by following the relevant branches of the tree. In the case of a probabilistic model, the detection model would contain a parametrization of the probabilistic model and the detector would compute a probability associated with each record. In the case of a support vector machine, the model would contain a set of support vectors which correspond to a decision boundary in a high dimensional feature space. The detector would effectively map a record to this high dimensional feature space and computes which side of the decision boundary the record falls on to determine whether or not to generate an alert. We give more details on the support vector machine model generation and detection below.

Detectors receive detection models from model distributors which distribute models stored in the data warehouse originally created by model generators. The detectors receive real time updates from the model distributors. This keep the detection models updated as soon as new models are available. Below is an example of the a model that the detector for the RAD system uses to make a classification.

```
<model>
<type> RAD </type>
<target> registrydb </target>
```

```
<version> 2.11 </version>
<encode>
<feature> <name> process </name> <n> 52000 </n> <r>
31 </r>
<values> iexplore.exe, aim.exe, explore.exe, msaccess.exe,
pinball.exe, .....  </values> </feature>
<feature> <name> keyval </name> <n> 52000 </n> <r>
1800 </r>
<values> HKLM, HKLM\Appications, ......  </values>
</feature>
..........
</encode>
</model>
```

Note that the encoding of this model is explained later in section 2.3. The evaluation of the record shown in the previous section with this model would result in a normal label for the record.

2.2 Data Warehouse

The data warehouse is the centerpiece of the AMG system. It serves as the central storage repository for all of the data collected by the sensors. The model generators access the data in the data warehouse and create detection models using this data. The data warehouse also stores these detection models. The analysis engines also access the data stored in the data warehouse. These components give the AMG system visualization and forensics capabilities as described below.

The core of the data warehouse is a SQL database. This allows for easy manipulation of the data, critical for creating training data sets to build data mining-based detection models. Since we can retrieve an arbitrary subset of data using a SQL query, the data warehouse automates the tedious process of manually creating these data sets. This flexibility is very important in practical deployments of the system.

For example, if there are 40 Windows hosts in a network and we wish to create an anomaly detection model over the Application Event logs for each of the hosts in the AMG framework, we perform the following. We first install a sensor on each of the hosts. This will collect the data and store it in the data warehouse. If each host is typically used in the same way, we may want to create a large data set containing the combined event logs from each of the hosts. On the other hand, if each host is used differently, we may create a separate training set for each individual host. Since the data warehouse uses a SQL database, we can create these different data sets by issuing different queries.

Storing the data in a single repository has several other advantages for correlating sensor outputs. Since the data is stored in a SQL database, we can use "join" statements to facilitate the linking of records from different sensors together into single records. In addition, we can obtain data from two sensors relatively easily because all of the data is stored in the same database.

The data warehouse uses an XML markup language for communication with all of the modules. The communication is specified in a specific XML markup language defined for this purpose. This markup language was influenced by the IDMEF specification Internet Engineering Task Force, 2000. The format for an insert transaction is displayed below.

```
<command>
<tablename>
<begin>
<rec>
<var1 var1type> valueA </var1>
<var2 var2type> valueA </var2>
<var3 var3type> valueA </var3>
.........
<varN varNtype> valueA </varN>
</rec>
<rec>
<var1 var1type> valueB </var1>
<var2 var2type> valueB </var2>
<var3 var3type> valueB </var3>
.........
<varN varNtype> valueB </varN>
</rec>
<end>
```

The transaction begins with a <command> to direct the data warehouse operation appropriately. The name of the table to be operated upon is then provided via <tablename>, where the pertinent information is stored. Then the information is sent in XML format. The data starts with a <begin> tag. Each record is started with a <rec> tag. Within each record all of the data is sent for that record, variable by variable. The variable name is sent along with its type as the tag, and between the tag is the value for that variable. Any number of records can be sent at a given time using this protocol. This greatly reduces the cost in many cases when there are many records being sent to the database by a sensor. When the data warehouse decodes the XML format, it checks to see if each variable has a column in the table where the

data is being inserted. If that column does not exist then it is created on the fly.

Below is a sample transaction. It is a single record being inserted into the nfr1 database by the HAUNT network sensor.

```
<insert>
<nfr1>
<begin>
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
<src-serror-rate f> 0.16265 </src-serror-rate>
<label str> normal </label>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
<end>
```

The HAUNT sensor connects to the data warehouse to transfer a record. It begins by sending an insert command to let the data warehouse know that it wants to insert data. Then it specifies the table nfr1 where the data is to be stored. Then it opens its first record with an opening <rec> tag. Then in order each variable is sent to the data warehouse. First the ID of the transaction which is an integer is sent over and that is 96. Next the destination numbers of bytes, also an integer, is sent. Then each variable is sent sequentially until the entire record is sent to the data warehouse. For convenience we abbreviated the types int, float, and string with i, f, and str respectively.

2.3 Detection Model Management

The AMG system manages the creation and distribution of detection models. The detection models are generated using data collected stored in the data warehouse. They are distributed to the detectors by the model distributors.

Detection Model Generation Our adaptive model generation system is designed to work with any model generation algorithm. Thus, the model generator components can be viewed as black boxes that are

"plugged" into the architecture with relative ease. These components take the training set of data as input and output a model of malicious activity. Different types of model building algorithms require different types of data. In our architecture we allow the model generators to select any data that it wants through the use of general or specific queries. This means that the architecture is robust enough to handle any type of model generation algorithm.

The model generation modules request the data from the data warehouse when they want to create a model. They form their request based on the information that the model needs to train on. The generator then runs and creates a model. This model is then encoded into XML and sent to the data warehouse. Model generators also signal the model distributor to let it know that a new model is available. A sample XML encoding of a model generated by the RAD system is shown below.

```
<model>
<type> RAD </type>
<target> registrydb </target>
<version> 2.11 </version>
<encode>
<feature> <name> process </name> <n> 52000 </n> <r>
31 </r>
<values> iexplore.exe, aim.exe, explore.exe, msaccess.exe,
pinball.exe, .....  </values> </feature>
<feature> <name> keyval </name> <n> 52000 </n> <r>
1800 </r>
<values> HKLM, HKLM\Appications, ......  </values>
</feature>
..........
</encode>
</model>
```

The model encoding begins with some meta-data about the model itself. The type field is used to notify the detector how to decode the rest of the model. The target specifies which table in the database this model applies to. The version information is used to coordinate with the model distributor in order to ensure that detectors have the most recent detection model. The model specifies information for evaluating the model follows the version information. This particular algorithm requires information and statistics about each feature in the data, and the values observed for that feature. This information is sent over one feature at a time. The encoding is specific to the type of model. All of the data between the <encode> and </encode> is specific to the model

type, and needs to be defined for each new detection model generation algorithm. This flexibility is what allows the adaptive model generation system to work with any types of models.

Detection Model Distribution Many model generation algorithms can be used in real-time environments. This creates the need for model distribution ensuring that all detectors have the most recent models. Detectors do not continuously check for updates in the data warehouse because this would be inefficient, and the real-time requirements of the system as a whole depends on the detectors being lightweight components. The model distributors are used to automatically send model updates to the detectors whenever the model generators create them.

2.4 Data Analysis Engines

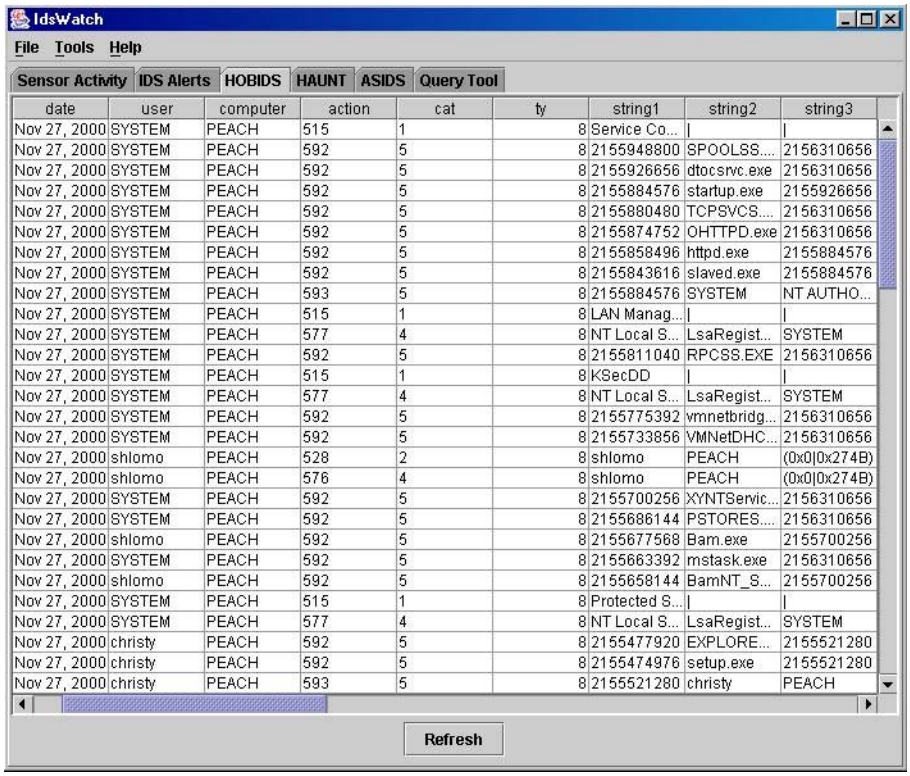
An analysis engine is any component that takes as its input a set of data from the database and performs some kind of analysis over this data. The analysis engines have the capability of inserting the results of the analysis into the database. In our system the analysis engine queries the data warehouse for a set of data and then inserts new information into the data warehouse using the SQL interface. This can be useful for several purposes. The data analysis engine uses the same XML format that the rest of the system uses with some specific tags designed specifically for data analysis.

Currently, we have implemented four types of analysis engines: a visualization client, a forensics tool, a data labeling tool and a feature extractor.

Visualization Analysis Engine The visualization analysis engine provides a system administrator with a mechanism to view all of the data in the data warehouse. An example of a visualization agent implemented by the adaptive model generation system is displayed in Figure 7.2.

The visualization analysis engine is integrated with the database which allows the use of SQL queries to filter the data to be viewed. An example of the interface and a SQL query is shown in Figure 7.3.

Forensics Analysis Engine One of the more important types of data analysis is forensic analysis. A forensic system retrieves a set of historical data from the data warehouse, typically the data of interest is a set of data which we suspect contains intrusions. The tool must retrieve a specific set type of data appropriate to the algorithm in question. Once the data set is retrieved the forensics analysis engine can apply a detection algorithm algorithm to find suspicious activity in the data set.



date	user	computer	action	cat	ty	string1	string2	string3
Nov 27, 2000	SYSTEM	PEACH	515	1	8	Service Co...	|	|
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155948800	SPOOLSS....	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155926656	dtocsrvc.exe	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155884576	startup.exe	2155926656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155880480	TCPSVCS....	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155874752	OHTTPD.exe	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155858496	httpd.exe	2155884576
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155843616	slaved.exe	2155884576
Nov 27, 2000	SYSTEM	PEACH	593	5	8	2155884576	SYSTEM	NT AUTHO...
Nov 27, 2000	SYSTEM	PEACH	515	1	8	LAN Manag...	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155811040	RPCSS.EXE	2156310656
Nov 27, 2000	SYSTEM	PEACH	515	1	8	KSecDD	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155775392	vmnetbridg...	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155733856	VMNetDHC...	2156310656
Nov 27, 2000	shlomo	PEACH	528	2	8	shlomo	PEACH	(0x0|0x274B)
Nov 27, 2000	shlomo	PEACH	576	4	8	shlomo	PEACH	(0x0|0x274B)
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155700256	XYNTServic...	2156310656
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155686144	PSTORES....	2156310656
Nov 27, 2000	shlomo	PEACH	592	5	8	2155677568	Bam.exe	2155700256
Nov 27, 2000	SYSTEM	PEACH	592	5	8	2155663392	mstask.exe	2156310656
Nov 27, 2000	shlomo	PEACH	592	5	8	2155658144	BamNT_S...	2155700256
Nov 27, 2000	SYSTEM	PEACH	515	1	8	Protected S...	|	|
Nov 27, 2000	SYSTEM	PEACH	577	4	8	NT Local S...	LsaRegist...	SYSTEM
Nov 27, 2000	christy	PEACH	592	5	8	2155477920	EXPLORE...	2155521280
Nov 27, 2000	christy	PEACH	592	5	8	2155474976	setup.exe	2155521280
Nov 27, 2000	christy	PEACH	593	5	8	2155521280	christy	PEACH

Figure 7.2. Visualization of Data in Database

The suspicious activity is then labeled (either anomalous or normal) using SQL statements to mark the appropriate data. Note that this requires that a column be added to the table in the database in order to store the label. The data warehouse has the capability to do this on the fly.

A sample input and output of a forensics analysis tool being used on RAD data can be seen below.

Input data from the data warehouse:

```
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
```

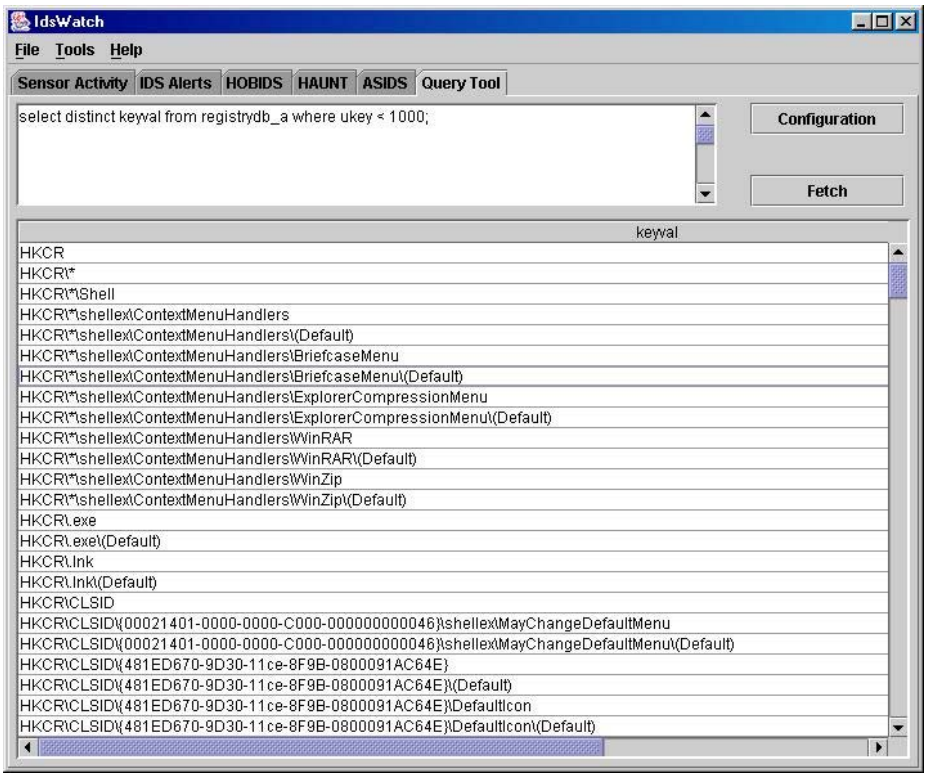


Figure 7.3. Visualization of SQL Query

```
<src-serror-rate f> 0.16265 </src-serror-rate>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
```

```
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
.....
```

The output sent back to the data warehouse contains the same data with a label appended to the end. In this example, the first record was labeled as an attack and the second record was labeled as normal.

```
<rec>
<ID i> 96 </ID>
<dst-bytes i> 490 </dst-bytes>
<rerror-rate f> 0.18786 </rerror-rate>
<sensor-rate f> 0.09760 </sensor-rate>
<src-bytes i> 1381 </src-bytes>
<src-count i> 151 </src-count>
<src-serror-rate f> 0.16265 </src-serror-rate>
<src str> 128.59.22.66 </src>
<dst str> 12.59.22.87 </dst>
<ip-overlap str> 0 </ip-overlap>
<label str> attack </label>
</rec>

<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<label str> normal </label>
</rec>
........
```

Data Labeling Tool Another data analysis engine is the data labeling tool. The data labeling tool takes the list of known attacks and uses that information to label all of the records in the database which corresponds to these attacks. The labeling tool is used to create labeled training data. The list of known attacks could be process names, time

stamps, or anything else that is also contained in the data records and can be matched to the known attacks. The labeling tool is a significant improvement over the difficult manual labeling of records in a database. The manual labeling of data is the greatest cost for deploying a data mining-based intrusion detection system. This cost is cut significantly through the use of this data labeling tool.

The data labeling tool is implemented using SQL joins with the sensor data in the data warehouse and the attack list. For example, let us assume we have a table full of Windows host based information from the application log. All actions in the application log are stored in the data warehouse with all available information from the log, including process name. Now assume that we have an attack list that is a list of all process names corresponding to attacks. We can automatically insert that attack list into the data warehouse in a temporary table. This temporary table could then be joined with the table of sensor data and the resulting table would be the sensor data labeled with its attack classification. This is a labeled set of training data that was created automatically from an attack list and a large set of sensor data. An example of the data labeling tool being used on the RAD data is seen below.

Input from the two tables in the data warehouse:

```
Raw data:
<rec> <process> iexplore.exe </process> <query>
queryKay </query> ...  </rec>
<rec> <process> happy99.exe </process> </query>
createKey </query> ...  </rec>
<rec> <process> outlook.exe </process> </query> openKey
</query> ...  </rec>
.....
```

Attack List of process name:

```
<process> happy99.exe </process>
<process> bo2k.exe </process>
.....
```

Labeled Data:

```
<rec> <process> iexplore.exe </process> <query>
queryKay </query> ...  <label> normal </label> </rec>
<rec> <process> happy99.exe </process> </query>
createKey </query> ...  <label> attack </label> </rec>
<rec> <process> outlook.exe </process> </query> openKey
</query> ...  <label> normal </label> </rec>
.....
```

Feature Extraction Features are important discriminating attributes derived from raw audit data that are employed in detection models. A feature extractor is any module that takes as input raw audit data and outputs additional pieces of information that were computed from the raw data. These new features are augmented to the original record. This can be thought of as a more general version of the forensic analysis engine. Many features are computed by using information that spans several individual records. This is because many times records by themselves are not meaningful, but in combination with other records they could represent an attack. The data warehouse has the capability to provide the feature extractor with any subset of data necessary. This could be the past n records for use with algorithms based on sequences, or those that compute temporal statistical features of connections or sessions. The flexibility of this system allows any group of record to be used to create a feature.

Features can also be created from a single record. In this case the feature extractor needs only to retrieve a single record and perform any calculations necessary to compute the feature.

Once the feature or features have been calculated they must be appended to the data in the data warehouse. A column is added to the table using the SQL interface to store the values of the new feature. An example of extracting some features gathered from the HAUNT sensor is shown below.

This shows features extracted from three records. In reality features could be extracted from any number of records. This example shows only the calculation of the number of http connections seen by the sensor thus far.

```
<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 100 </ID>
```

```
<dst-bytes i> 325 </dst-bytes>
<rerror-rate f> 0.13426 </rerror-rate>
<sensor-rate f> 0.12450 </sensor-rate>
<src-bytes i> 1341 </src-bytes>
<src-count i> 242 </src-count>
<src-serror-rate f> 0.12435 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>

<rec>
<ID i> 101 </ID>
<dst-bytes i> 425 </dst-bytes>
<rerror-rate f> 0.12456 </rerror-rate>
<sensor-rate f> 0.12654 </sensor-rate>
<src-bytes i> 1311 </src-bytes>
<src-count i> 102 </src-count>
<src-serror-rate f> 0.21325 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
</rec>
```

The updated records contain a new feature `num_http` which stores the new information.

```
<rec>
<ID i> 99 </ID>
<dst-bytes i> 420 </dst-bytes>
<rerror-rate f> 0.12786 </rerror-rate>
<sensor-rate f> 0.16760 </sensor-rate>
<src-bytes i> 1281 </src-bytes>
<src-count i> 132 </src-count>
<src-serror-rate f> 0.19325 </src-serror-rate>
<src str> 128.59.22.69 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 1 </num_http>
</rec>

<rec>
<ID i> 100 </ID>
```

```
<dst-bytes i> 325 </dst-bytes>
<rerror-rate f> 0.13426 </rerror-rate>
<sensor-rate f> 0.12450 </sensor-rate>
<src-bytes i> 1341 </src-bytes>
<src-count i> 242 </src-count>
<src-serror-rate f> 0.12435 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 2 </num_http>
</rec>

<rec>
<ID i> 101 </ID>
<dst-bytes i> 425 </dst-bytes>
<rerror-rate f> 0.12456 </rerror-rate>
<sensor-rate f> 0.12654 </sensor-rate>
<src-bytes i> 1311 </src-bytes>
<src-count i> 102 </src-count>
<src-serror-rate f> 0.21325 </src-serror-rate>
<src str> 128.59.22.63 </src>
<dst str> 12.59.22.121 </dst>
<ip-overlap str> 0 </ip-overlap>
<num_http> 3 </num_http>
</rec>
```

2.5 Efficiency consideration

An important consideration when designing an intrusion detection system is efficiency. A real-time system must be able to respond to intrusions in a timely manner so that action can be taken, without utilizing too many of the resources of the system it is intended to protect. This is especially important in the case of host based systems. The adaptive model generation framework emphasizes light components and a distributed architecture. Resource heavy components can be separate from the system that the IDS is trying to protect. The only component that needs to be run on the system being protected is the lightweight sensor. This greatly minimizes the amount of computational resources taken by the IDS.

An example of where this advantage is useful is in the HAUNT Dios et al., 2001 system which is a network intrusion detection system. In section 7 we describe the deployment of the HAUNT system in the AMG framework.

3. Capabilities of Adaptive Model Generation

The adaptive model generation system has many advantages over a traditional intrusion detection system. AMG facilitates real time detection, data collection and storage, model generation and distribution, and various types of analysis of the data. It also facilitates correlation of various data streams.

3.1 Real Time Detection Capabilities

The sensor and detector provide real time detection capabilities to the AMG system. Both are as light weight as possible and the main advantage of the distributed framework is the ability to isolate the sensor and detector from the remaining components of the system to maximize their real-time performance.

3.2 Automatic Data Collection and Data Warehousing

The distributed architecture of the adaptive model generation system allows for the automation of the data collection and data warehousing. In the AMG framework, simply deploying a sensor will automatically collect and aggregate that sensors data in the data warehouse. There are many reasons why it is desirable to aggregate the date. For example, we may be interested in performing forensic analysis of archival data. It may also be useful to look back at errors made by the intrusion detection system in order to improve performance and study weaknesses.

Heterogeneous Data Support The distributed architecture of adaptive model generation allow the intrusion detection system to gather data from heterogeneous systems. A set of standard guidelines in a flexible format are placed on sensor data. There are many different types of information that IDSs use such as network packets, application logs, Windows registry accesses, etc. The ability to accommodate these different sources of information in a consistent way is a large advantage of the adaptive model generation system.

This is easily accomplished because all of the data gathered by the system is transmitted to the data warehouse using our XML mark up

language. The data warehouse system is flexible enough to store all types of information.

Labeling collected data Labeling collected data is necessary to create training data for data mining-based detection models. To accomplish this without a tool an administrator would have to manually go through the data record by record and label each piece of data individually. The adaptive model generation system provides a tool that automates the process of labeling training data.

3.3 Model Generation and Management

In a typical network environment, we can conceivably have hundreds of models deployed throughout the network. These models can also become out of data. The management of the models quickly can become a very difficult task.

The adaptive model generation solves this problem by creating a mechanism for the creating and management of detection models. The models are created using the detection model generators. They are then stored in the data warehouse. The data warehouse is robust enough to handle any types of models and therefore the system can be used with any types of models. The data warehouse is also stable enough that failure of model storage is not a concern while the protected machine is under attack. The use of model distributors allows the system to update and alter models on the fly with a minimal computational overhead. This is very advantageous because it allows the system to be deployed for a long period of time without the need for maintenance by an administrator.

3.4 Data Analysis Capabilities

The adaptive model generation provides users with the functionality to implement different data analysis tools. A data analysis tool is anything that retrieves the data from the data warehouse, performs some sort of computation with the data, and then either sends new information back into the data warehouse or uses the new information in some other way. There are many cases where this can be a very useful and important tool. The system is designed so that any data analysis tool can be created and work with the system. We have implemented three types of data analysis tools. They are forensics, feature extraction, and visualization.

Forensics Forensics is a very important field in computer security. Currently forensic analysis of data is done manually. Computer experts have to search through large amounts of data, sometimes millions of records, individually and look for suspicious behavior. This is an extremely inefficient and expensive process. The adaptive model generation framework contains components that can automate the forensic analysis process.

Forensics can be done with misuse detection models if there is a learned detection model for that data. If a learned detection model exists if can be run over the data and we could find the intrusions in the data after the data has already been collected. The method can be applied with signature-based models which are used by commercial systems today. We can also use anomaly detection models if there exists a normal model for the data set.

In some cases, we do not have an appropriate model of any kind to perform forensics. In these cases, we can use an unsupervised anomaly detection algorithm over the data. Unsupervised anomaly detection algorithms can be used to perform forensic analysis on unlabeled data. The adaptive model generation framework enables this process. Unsupervised anomaly detection algorithms detect intrusions buried within an unlabeled data set. Unsupervised anomaly detection algorithms are described in section 4.3.

Visualization The adaptive model generation system provides a visualization tool so that an administrator can examine all of the data in the data warehouse. This can also provide an administrator or researcher with information about the strengths and weaknesses of a particular intrusion detection system. If an administrator recognizes activity as an attack but the system does not, he can act and the system can be protected even though the system missed the attack. In addition by seeing the activity during an intrusion this can provide insight into the vulnerabilities of the host as well, and better explain how attacks work. This will help to more accurate detection models in the future and provide security experts with the knowledge they need to improve security systems.

Feature Extraction The success of a model generation algorithm depends largely on the quality and correlation of the data. Feature extractors are components that transform the basic features gathered by the sensors into more meaningful ones, often referred to as advanced features. For example the time stamp on a packet is not a very important feature when considered alone. However using the time stamp to

compute the number of packets within the last two seconds can be a crucial piece of information in determining certain types of network attacks Lee and Stolfo, 1998. Models learned over well-computed features are generally far superior to those computed over raw pieces of information Lee, 1999.

Feature extractors can be seen as data analysis engines by the adaptive model generation system. They retrieve data from the data warehouse and then perform computations on that data. Once these computations are completed the new data is sent back to the warehouse and appended with the new information.

In many cases the feature extractors are built into the sensors. This makes the number of components smaller and easier to manage. However this means that a specialized feature extractor must be made for each sensor. This is not a drawback in many cases because features are carefully selected based on analysis of the data gathered by that sensor. In those cases using a feature extractor for two or more different sensors doesn't make any sense. However there exist feature extraction algorithms that can work on multiple types of data, even ones that can work on any data. It is in these cases where feature extraction is particularly useful. This is because the feature extractor can be viewed as an autonomous unit that can be used by any sensor to alter the data and improve performance, with almost no additional production overhead.

Another concern with combining the feature extraction with the sensor is that many feature extraction algorithms can be very computationally expensive. The sensor is the only component that must be run on the system it is protecting. It is therefore crucial that the sensor is very lightweight. Separate feature extraction modules can be extremely helpful in keeping the sensors lightweight.

3.5 Correlation of Multiple Sensors

Distributed models are models that are trained and evaluated over multiple sets of data from multiple sources. Traditional intrusion detection systems would have difficulty combining data from multiple different sources, especially across different networks. Intuitively if a machine learning algorithm has more data from more sources then it will perform better. By eliminating dependencies between sensors, model generators, and detectors the adaptive model generation system has enabled correlation algorithms to be constructed the same as any other algorithm.

The distributed architecture and the data warehouse allow us to implement correlation algorithms with no additional implementation overhead. The data warehouse will allow us to retrieve any subset of the

data in the database with a single query. This means that data from multiple sources can be retrieved just as easily as data from a single source. This data can be aggregated and used more efficiently than if it was stored individually.

4. Model Generation Algorithms

There are three types of model generation algorithms that the AMG system supports. The first is misuse detection which trains on labeled normal and attack data. The second is supervised (traditional) anomaly detection which trains on normal data. The third is unsupervised anomaly detection which trains on unlabeled data.

4.1 Misuse Detection

Misuse detection algorithms train over normal and attack data. Using this data, these algorithms build a model that can discriminate between attack records and normal records. These models can then classify new records as either attack or normal. This approach has been very successful in the past Lee et al., 1997. The only major disadvantage of this type of system is that it requires labeled training data that contains labeled normal activity and labeled attacks. This data is very expensive to obtain, and it may not be portable from one system to another or from one network to another.

Misuse detection algorithms can be used as model generation algorithms in the adaptive model generation framework. The training data for misuse detection algorithms must consist of labeled normal and attack data often making the training data for this algorithm very expensive.

Using the AMG system, we can help minimize the cost of labeling the data. Once we deploy the sensors into a network, we can run simulated attacks and record the time stamps and other information about the attack. Since the sensors will be automatically sending the data to the data warehouse, the data for labeling is already aggregated into one location. Using the data labeling tool, we can label the attack data. This labeled data is now stored in the data warehouse and can be retrieved by the model generators. These models can also be distributed in to the detectors using the model distributors.

4.2 Anomaly Detection

Anomaly detection algorithms train over normal data to create a model of normal activity. These algorithms need to train over data that contains no intrusions. The training data needed for these algorithms

is expensive because it is difficult to ensure that the data contains no intrusions. This can be done by either having an expert manually clean the data or by somehow ensuring that the data contains no intrusions to begin with. In general this is not as expensive as the training data necessary for misuse detection algorithm. However many anomaly detection algorithms require a very large amount of training data which can increase the cost.

Once an anomaly detection model is trained, it can then classify new data as normal or anomalous. These algorithms rely on the assumption that attacks are cause behavior that is different from normal.

The adaptive model generation framework supports the creation of anomaly detection models. Since sensors send data to the data warehouse, it is easy to aggregate the data for collection. Using the forensics analysis engine, we can help check to see if the data is clean and contains no intrusions. This can greatly decrease the cost of creating the training set since it speeds the process of verifying that the data is clean. The model generators can automatically generate anomaly detection models using the data from the data warehouse and deploy the detection models using the model distributor.

4.3 Unsupervised Anomaly Detection

Unsupervised anomaly detection algorithms examine unlabeled data and attempt to detect intrusions buried within the unlabeled data. Unsupervised anomaly detection algorithms makes the assumptions that intrusions are very rare compared to the normal data and they are also quantitative different. Because of this, intrusions are outliers in the data and can be detected. Unsupervised anomaly detection is discussed in depth in Eskin, 2000; Portnoy et al., 2001; Eskin et al., 2002.

Since unsupervised anomaly detection can detect intrusions in an unlabeled data set, they are used inside the forensics analysis engines. Data from the data warehouse is sent to a forensics analysis engine where a unsupervised anomaly detection algorithm is applied. The forensics analysis engine can label the data which it determines to be an outlier.

Unsupervised anomaly detection algorithms can also be used to help label data that is collected by the system. This labeled data can then be used to train a misuse or anomaly detection model.

5. Model Generation Example: SVM

One specific type of model generation algorithm used by AMG is Support Vector Machines (SVMs). SVMs were first introduced by Vap-

nik Vapnik and Chervonenkis, 1974 as a supervised machine learning algorithm. Vapnik's SVM algorithm is used in AMG for misuse detection. An unsupervised variant of the SVM algorithm was put forth by Schölkopf et. al. Schölkopf et al., 1999. This algorithm can be used for both Unsupervised Anomaly Detection and normal Anomaly Detection.

5.1 SVM Algorithm

Vapnik's SVM algorithm is a binary classifier. The idea behind an SVM approach to intrusion detection is that we map our data to a *feature space*. Inside this feature space, use use the SVM and a set of labeled training data to determine a linear decision surface (hyperplane). This surface is then used to classify future instances of data. Data is classified based upon which side of the decision surface it falls.

Given a training set S consisting of m vectors and their labels (x_i, y_i) where $x_i \in \Re^n$ and $y_i \in \{\pm 1\}$, the algorithm generates a decision surface. The decision surface is a hyperplane of the form $\langle w, x\rangle + b = 0$ where w is normal to the hyperplane and b scalar that shifts the hyperplane. The decision surface that is chosen is determined by solving an optimization problem that determines the "best" hyperplane under a set of criteria which is described fully in Cristianini and Shawe-Taylor, 2000.

The classification of a future instance $x \in \Re^n$ is made by the function

$$f(x) = \text{sgn}(\langle w, x\rangle + b)$$

It is shown in Cristianini and Shawe-Taylor, 2000 that solving the following optimization problem results in a solution the solution to the SVM optimization.

$$\text{maximize: } \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle$$

$$\text{subject to } 0 \leq \alpha_i, \sum_i \alpha_i y_i = 0$$

Setting $b = 0$, the solution is then:

$$w = \sum_i \alpha_i y_i x_i$$

All x_i with $\alpha_i \neq 0$ are called the support vectors. These are the vectors on the boarder of each class that determine the unique solution. If a support vector were removed it would change the resulting hyperplane.

However, all non-support vectors are irrelevant to the solution. They can all be removed and the solution would not change.

This algorithm performs best when the data is linearly separable data. However in order to work for the non-linearly separable case, data must be mapped into a higher dimension feature space where it does become linearly separable. In addition, often intrusion detection data are not all vectors in $\Re^n$ so there is no natural definition of the dot products between the data elements.

Since the SVM algorithm is defined in terms of dot products, we can use kernel functions to define both the mappings of the elements to the feature space and the dot product within these space simultaneously. This fact can be exploited and a kernel function can be used in place of the dot product.

Let Φ be a feature map $\Phi : X \to F$. Φ maps the input space X into a dot product space called the feature space F. A kernel function K implicitly maps data into this feature space and takes the dot product in that space.

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

An example of a kernel function is the Gaussian kernel.

$$K(x_i, x_j) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$$

Now the support vector machine optimization equation and classification equation can be rewritten in terms of kernels.

$$\text{maximize} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to: } 0 \leq \alpha_i, \sum_i \alpha_i y_i = 0$$

Substituting the formula for w into the classifier equation we get another dot product that can be converted to a kernel. Setting $b = 0$ the solution is then:

$$f(x) = \text{sgn} \left(\sum_{i=1}^{N_s} \alpha_i y_i K(s_i, x) + b \right)$$

where N_s is the number of support vectors and s_i is the i^{th} support vector

5.2 SVM for Misuse Detection in AMG

The standard support vector machine algorithm is used for misuse detection by AMG. Data in the form of vectors of real numbers are sent from the sensors to detectors. The detectors use a SVM model to differentiates between normal data and intrusion data.

To implement this in the AMG system, training data must first be generated. A system is monitored by sensors that send their observations to the data warehouse in the form of XML tagged data. Sporadically, different attacks are launched against the system. After enough training data has been generated, data is labeled in the data warehouse as either normal or attack. This labeled data is then sent via XML to the model generator. The model generator uses the SVM algorithm to create a model for misuse detection. A model, in this case, is the set of support vectors and their weights. This model is automatically sent to the data warehouse for storage and to all of the detectors that use this kind of model. Once the model is in place, sensors send data that they are monitoring to the for classification by the SVM classification rule.

SVMs could also be used in misuse detection to determine what kind of attack is being carried out against the system. This would require labeling training data with a more specific attack type label. Then a set of support vector machines can be training with each one trying to detect a specific attack. This basically equates to taking the intersection of these support vector machines. This would not add much additional complexity but it might interfere with the accuracy of the classification.

5.3 Unsupervised SVM Algorithm

The standard SVM algorithm is a supervised learning algorithm. It requires labeled training data to create its classification rule. Schölkopf adapted the SVM algorithm into an unsupervised learning algorithm. This unsupervised variant does not require its training set to be labeled to determine a decision surface.

The algorithm is similar to the standard SVM algorithm in that it uses kernel functions to perform implicit mappings and dot products. It also uses the same kind of hyperplane for the decision surface. The solution is only dependent on the support vectors as well. However, the support vectors are determined in a different way. This algorithm attempts to find a small region where most of the data lives and label it as class $+1$. Everywhere else is labeled as class -1. This is accomplished by finding the hyperplane that maximizes the distance from the origin while still capturing the majority of the data. The support vectors define that hyperplane.

Given a training set S consisting of m vectors $x_i \in \Re^l$

$$\text{minimize: } \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j K(x_i, x_j)$$

$$\text{subject to: } 0 \leq \alpha_i \leq \frac{1}{vl}, \sum_i \alpha_i = 1$$

where $0 < v < 1$ is a parameter that controls the trade off between maximizing the distance from the origin and containing most of the data in the region created by the hyperplane.

The classification equation is:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{N_s}\alpha_i K(s_i, x) - b\right)$$

where N_s is the number of support vectors and s_i is the i^{th} support vector

For this algorithm b cannot be set to 0, it must be found explicitly.

$$b = \sum_{j=1}^{N_s}\alpha_j K(s_j, s_i)$$

5.4 Unsupervised SVM for Unsupervised Anomaly Detection

The standard SVM algorithm is not useful for Unsupervised Anomaly Detection as it requires labeled data. The unsupervised SVM variant proposed by Schölkopf can be used for UAD. This approach was used in Eskin et al., 2002 to perform UAD using the AMG framework. The algorithm differentiates between normal data and anomalous data. Anomalous data is thought to be intrusion data because intrusions are mush different than normal system use.

Like the misuse detection algorithm, UAD requires training data. During a training period a system is run normally with no attacks. Sensors monitoring the system send their observations via XML to the data warehouse. Although no attacks are intentionally run, if some unknown attacks were to occur, there would be no problem. The algorithm can tolerate some noise (unknown attacks) and still generalize well. Once enough training data has been accumulated it is sent from the data warehouse to the model generator via XML. There is no need to label data. The model generator then uses the unsupervised SVM algorithm

to generate a model of normal activity. This model is made up of the set of support vectors and their associated weights. The model is then sent to the data warehouse for storage and to the appropriate detectors. Once the detection model is in place, sensors send data to the detector for classification.

Supervised Anomaly Detection can also be performed using the same method. However, all of the training data would have to be guaranteed to be attack free.

6. System Example 1: Registry Anomaly Detection

The AMG framework can support a variety of different intrusion detection systems. One example of an IDS system that is integrated into the AMG system is the *Registry Anomaly Detection* (RAD) system. The RAD system is a host-based IDS system which runs on the Microsoft Windows platform. RAD monitors the accesses to the windows registry on a host and detects anomalous registry accesses that correspond to attacks. It uses an anomaly detection algorithm to make models of normal registry accesses and compares in real time, monitored accesses to that model. Details on the system as well as details on experiments showing the effectiveness of the RAD system are presented in Apap et al., 2001.

6.1 The RAD Data Model

The RAD system uses 10 features to characterize each registry access. Five of these are basic features that come directly from the registry accesses, and five are composite features which are made from the combination of two of the basic features. The basic features are `Key`, `Process`, `Query`, `Response`, and `ResultValue`. The advanced features are `Process/Query`, `Key/Process`, `Query/Key`, `Response/Key`, and `ResultValue/Key`. The first five features are derived directly from the registry accesses.

The registry is stored in a tree structure, and all information is stored in a data structured called a key. The name of the location where the information is stored is the '`Key` basic feature. The `Process` feature is the name of the process that is performing the registry access. The `Query` feature represents the type of access being made, such as `QueryValue`, `CreateKey`, `SetValue`, etc. The `Response` feature is the outcome of the query, such as `success`, `not found`, `access denied`, etc. The `Result` feature is the value of the key being accessed. These five features provide all the necessary information about any single registry access.

6.2 The RAD Sensor

The RAD sensor consists of two parts. The first part connects to the Windows operating system and monitors the accesses to the registry. This part is implemented as a basic auditing module(BAM). The BAM includes a hook into the audit stream which is the windows registry. The architecture of the system is taken from the commercial software Regmon produced by SysInternalsSysInternals, 2000. The BAM uses Win32 hooks to listen for all reads and writes to registry.

The second part of the RAD sensor is the communication component which translates this data into our XML format and sends it to the data warehouse. The communication module can supports multiple BAMs at the same time. This is done so that all sensors running on a host can be sent through a single source. Then the communication engine can send the data from all these sources to the data warehouse for storage.

The RAD BAM is not trivial to implement. The most direct method to implement the RAD BAM is through the Windows Event Log. However, the Windows Event Log is unable to handle the amount of traffic generated by the registry which required the use of more sophisticated Win32 hooks.

The five composite features that are used by the RAD system are examples of feature extraction. This is the simplest type of feature extraction possible, the combination of two basic fields. The RAD system uses these composite features in order to better classify activity. This is an example of the feature extraction capabilities of the adaptive model generation system. This is one of the cases where the feature extractor is very lightweight and therefore a part of the sensor.

6.3 The RAD Classification Algorithm

These ten features are used to classify each registry access as either normal or anomalous. In order to do this we implemented an anomaly detection algorithm first introduced by Phil Chan and Mathew Mahoney in the PHAD systemMahoney and Chan, 2001. This algorithm was first used to detect anomalies within packet headers in order to look for malicious behavior. The adaptive model generation algorithm allows us to use this algorithm even though it was created for Packet Headers. The algorithm trains over normal data to develop models of what normal activity is like and how to classify abnormal activity. The algorithm is based on keeping statistics about the data and using those statistics to determine anomalous features.

Each feature is individually evaluated to be either normal of anomalous. Then the statistics we gathered are used to score these anomalies.

This score is based on how likely we think it is that the value of this feature will be different then values seen in the past. These scores are then added together and if they are over a threshold then the access is considered to be malicious, otherwise it is classified as normal. This algorithm is however not important to the adaptive model generation system, in reality any algorithm could have been used and it would not have effected the overall architecture. Also from the point of view of the classification algorithm the sensor is not important. This algorithm could have been used on any data without any changes in architecture.

6.4 The RAD Detector

In order to detect anomalies in real time, a detector was implemented for the RAD system. This detector was implemented specifically for the RAD system but it could be used to evaluate any model that was created by the classification algorithm described in the previous section. The first requirement of the detector is that is must receive data from the sensor in real time. This is necessary to evaluate models in real time. The detector must also decode the model and have the capability to receive real time updates to this model. The RAD detector would retrieve the model from the data warehouse, decode it, and then evaluate each record that it was sent from the sensor. This is all done in real time and consequently the system is successful in detecting malicious activity in real time.

The multiplicity of this system can easily be increased with the adaptive model generation system. With no changes in architecture the system can support any number of host machines and sensors. Without the adaptive model generation architecture increasing the multiplicity would require major changes in the structure of a system. This is because the central data collection is automated in AMG. This means that data from multiple machines is gathered in the same place and can be analyzed from that central location.

Since the focus of this paper to highlight the RAD system as an instance of AMG we are not reporting the results in this paper. The results of this system can be found in Apap et al., 2001.

7. System Example 2: HAUNT

The *Heuristic Audit of Network Traffic* (HAUNT) system is a network based intrusion detection system that classifies network data as either normal or attack. Previous research has shown that network packet information can be useful in detecting intrusions. The majority of com-

mercial intrusion detection systems use network data to detect attacks. This is because many attacks are remote attacks and they can be seen in the network data. However these commercial systems are signature-based due to the high cost of deploying a data mining based network intrusion detection system.

Again, the focus of this description of the HAUNT system is to describe how it is integrated into the AMG system. Details as well as experiments evaluating the performance of HAUNT are presented in detail in Dios et al., 2001.

7.1 HAUNT Sensor

The HAUNT sensor is designed to gather information from a network stream. It listens to all network data, formats it, and sends that data directly to the data warehouse. The network sensor does not use a communication engine because it does not run on a host, so there is no need to aggregate information before it is sent to the data warehouse. The HAUNT sensor is implemented by utilizing the commercial products NFR Inc., 1997 and Snort Roesch, 1999. They use an abstract feature definition structure and a feature exchange protocol to extract information from the NFR and Snort systems. The HAUNT system only uses packet header information to extract features. This is done for efficiency purposes and because the system can be effective and inexpensive using just this information.

7.2 HAUNT Classification Algorithm

The HAUNT system uses a multiple model cost-sensitive approach to improve efficiency. The system is designed to minimize the computational cost of an intrusion detection system. The system first attempts to make a classification based on a simple rule and the basic data gathered from the sensor. If the system can not confidently make a classification at that point the system will perform more calculations in order to make a better decision. The system accomplishes this by implementing multiple models to classify the data. The difference between the models is that some are more accurate at the price of being more computationally expensive. The system does not evaluate the more expensive models unless it has to in order to make a classification. The more expensive models are more expensive in large part due to the fact that they require more data. These expensive models require derived features from the packet information. Some of these features are very expensive to calculate and therefore they are only calculated when needed by the more expensive models.

7.3 HAUNT Detector

The HAUNT system uses a special type of detector called JUDGE that implements multiple model evaluation.

The JUDGE system was implemented as a part of the HAUNT system in order to accomplish the evaluation of the multiple models. The JUDGE system is the system that decides whether to calculate more expensive features and evaluate more expensive models. The JUDGE models are models generated from the RIPPERCohen, 1995 model generation program. RIPPER generates rule sets for evaluation by the JUDGE system. These rule sets come in one of two different types. The first type is ordered rule sets. When evaluating ordered rule sets JUDGE goes through each rule one by one until one of the rules can make a classification and then that rule makes the decision. The second type of rule set is unordered rule sets. When evaluating unordered rule sets each rule in the set is evaluated and the rule with the most precise ruling makes the ruling. The unordered rule sets are more precise because they are always labeled by most precise classifying rule. However ordered rule sets are faster because in many cases JUDGE does not have to evaluate every rule in the rule set.

7.4 HAUNT Feature Extraction

The HAUNT system uses a feature extractor to discover features that are useful for detecting attacks. The algorithms for performing this feature discovery are described in Lee, 1999.

The HAUNT system uses feature descriptor in order to define the features that it uses for classification. These features are defined using arithmetic and logic expressions to combine primitive features. The logic expressions implemented by this system are SUM, AND, and UNIQUE. These features can be used to create a wide variety of important features. The sum feature could be used to calculate the total number of times something has happened. For example if we wanted to calculated the total number of tcp connections we could use

num_tcp_connections = SUM(protocol==tcp)

The SUM(protocol==tcp) which return the total of number of records of which the condition service==http is true. If we wanted to calculate the total number of tcp connections going to port 2301 we could use.

num_tcp_connections_to_port_2301 = SUM((protocol==tcp) AND destination_port==2301))

The AND operator is used to take the AND of two conditional expressions the same way it is normal used. The final logical operator is the UNIQUE operator. The UNIQUE operations takes in two parameters, a conditional, and a feature. The operator will return the number of unique values that feature has had when the condition is true. For example to get the number of different ports accessed by tcp protocol we would use.

num_tcp_ports = UNIQUE(protocol==tcp, destination_port)

These logical functions along with arithmetic functions such as multiplication and addition are all the HAUNT system needs to define all of the features it uses. The feature extraction provided by these tools can be seen as a data analysis engine by the adaptive model generation system. Feature extraction is an important part of many intrusion detection systems. The HAUNT system is an example of the feature extraction capabilities of the adaptive model generation system can be extremely useful in enabling an intrusion detection system. Without feature extraction capabilities the HAUNT system would not be realizable.

Since the focus of this paper to highlight the HAUNT system as an instance of AMG we are not reporting the results in this paper. The results of this system as well as further implementation details can be found in Dios et al., 2001.

8. Conclusion

In this paper we presented adaptive model generation, a method for automatically and efficiently creating models for an intrusion detection system. The system uses data collected by sensors to create a model and to search for attacks. It uses an anomaly detection algorithm to create models to classify data. The system updates models in real time so that over time performance can be improved. Adaptive model generation can significantly reduce the cost of deploying an intrusion detection system because it streamlines the process of gathering training data, creating models, and evaluating data.

The system uses a distributed architecture with autonomous components to increase the flexibility of the system. This additional flexibility has allowed us to create many modules and tools that greatly reduce the cost of deploying an intrusion detection system. Automated data collection and storage saves the time and effort to manually gather data. The data labeling tool streamlines the process of labeling the data. This greatly reduces cost because labeling data requires a security expert to check each record individually, which in some cases could be millions of records. Automated model generation and distribution saves the cost of

manually updating the models as they become outdated. Visualization capabilities allow an administrator to be involved with the intrusion detection in a manner that would not be possible with traditional black box intrusion detection. This reduces the deployment cost dramatically by circumventing the slow and lengthy process of gather and labeling training data.

The support of heterogeneous data and central storage of data enables the system to combine data from different sources very easily. In traditional systems combining data between sensors is not possible. The distributed architecture of the adaptive model generation system allows machine learning algorithms use data from multiple sources just as easily as data from one source. This allows correlation algorithms to be integrated into the adaptive model generation framework which could potentially increase the performance of an intrusion detection system.

Unsupervised anomaly detection algorithms eliminate the need for specialized training data. A system using an unsupervised anomaly detection algorithm could be deployed without labeled training data. This system can train over raw audit data gathered from sensors. This substantially reduces the deployment cost of an IDS. This property of unsupervised anomaly detection algorithm makes them very effective in the deployment of forensic analysis systems. The adaptive model generation implements data analysis components which can analyze data in the data warehouse and append information to the data. This in combination with an unsupervised anomaly detection will yield a forensic analysis system that can detect attacks within a data set without prior knowledge of the data set.

Future work includes expanding the correlation abilities of the system. Since all data of the system is stored in a central location we can combine data from different sensors in order to obtain a better picture of activity on a host machine or network. This approach will allow for the building of more complicated and more effective models. This approach will also allow us to study the relationship between data about the same attack that is gathered from different sources. An understanding of this relationship would aid greatly in future work on intrusion detection systems.

We can also extend the data representation to take advantage of linking capabilities of(or associated with) XML such as links among models and the data sets used to generate them. The adaptive model generation framework allows us to do this linking at a very small cost.

References

Apap, F., Honig, A., Hershkop, S., Eskin, E., and Stoflo, S. (2001). Detecting malicious software by monitoring anomalous windows registry accesses. Technical report, CUCS Technical Report.

Cohen, W. W. (1995). Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK.

Denning, D. (1987). An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13:222–232.

Dios, P. D., El-Khalil, R., Sarantakos, K., Miller, M., Eskin, E., Lee, W., and Stolfo, S. (2001). Heuristic audit of network traffic: A data mining-based approach to network intrusion detection. Technical report, CUCS Technical Report.

Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. Technical report, CUCS Technical Report.

Eskin, E., Lee, W., and Stolfo, S. J. (2001). Modeling system calls for intrusion detection with dynamic window sizes. In *Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX II)*, Anaheim, CA.

Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society.

Ghosh, A. and Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Proceedings of the Eighth USENIX Security Symposium*.

Hershkop, S., Apap, F., Glanz, E., D'alberti, T., Eskin, E., Stolfo, S., and Lee, J. (2001). Hobids: A data mining approach to host based intrusion detection. Technical report, CUCS Technical Report.

Inc., N. F. R. (1997). Network flight recorder. http://www.nfr.com.

Internet Engineering Task Force (2000). Intrusion detection exchange format. In *http://www.ietf.org/html.charters/idwg-charter.html*.

Lee, W. (1999). *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University.

Lee, W. and Stolfo, S. J. (1998). Data mining approaches for intrusion detection. In *In Proceedings of the Seventh USENIX Security Symposium.*

Lee, W., Stolfo, S. J., and Chan, P. K. (1997). Learning patterns from unix processes execution traces for intrusion detection. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. Menlo Park, CA: AAAI Press.

Lee, W., Stolfo, S. J., and Mok, K. (1999). Data mining in work flow environments: Experiences in intrusion detection. In *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99).*

Mahoney, M. and Chan, P. (2001). Detecting novel attacks by identifying anomalous network packet headers. Technical Report CS-2001-2, Florida Institute of Technology, Melbourne, FL.

Paxson, V. (1998). Bro: A system for detecting network intruders in real time. In *7th Annual USENIX Security Symposium.*

Portnoy, L., Eskin, E., and Stolfo, S. J. (2001). Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001).*

Roesch, M. (1999). Snort - lightweight intrustion detection for networks. In *Proceedings of Lisa '99.*

Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (1999). Estimating the support of a high-dimensional distribution. Technical Report 99-87, Microsoft Research. To appear in *Neural Computation*, 2001.

Staniford-Chen, S., Tung, B., and Schnackenberg, D. (1998). The common intrusion detection framework (cidf). In *Proceedings of the Information Survivability Workshop.*

SysInternals (2000). Regmon for Windows NT/9x. *Online publication.* http://www.sysinternals.com/ntw2k/source/regmon.shtml.

Vapnik, V. and Chervonenkis, A. (1974). *Theory of Pattern Recognition [in Russian].* Nauka, Moscow. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society.

Chapter 8

PROACTIVE INTRUSION DETECTION

A Study on Temporal Data Mining

João B. D. Cabrera
Scientific Systems Company
500 West Cummings Park, Suite 3000 Woburn, MA 01801 USA
cabrera@ssci.com

Lundy Lewis
Aprisma Management Technologies
121 Technology Drive Durham, NH 03824 USA
lewis@aprisma.com

Xinzhou Qin
Georgia Institute of Technology - College of Computing
801 Atlantic Drive Atlanta, GA 30332 USA
xinzhou@cc.gatech.edu

Wenke Lee
Georgia Institute of Technology - College of Computing
801 Atlantic Drive Atlanta, GA 30332 USA
wenke@cc.gatech.edu

Raman K. Mehra
Scientific Systems Company
500 West Cummings Park, Suite 3000 Woburn, MA 01801 USA
rkm@ssci.com

Abstract This chapter describes a principled approach for discovering precursors to security violations in databases recorded from multiple domains in networked information systems. These precursors can be used by security analysts to better understand the evolution of complex computer attacks, and also to trigger alarms indicating that an attack is imminent. We call Proactive Intrusion Detection the utilization of these temporal rules as part of an overall Information Assurance Infrastructure, including Prevention, Detection, Response and Tolerance. The approach is rooted in time series quantization, and in the application of the Granger Causality Test of classical statistics for selecting variables that are likely to contain precursors. A methodology is proposed for discovering Precursor Rules from databases containing time series related to different regimes of a system. These Precursor Rules relate precursor events extracted from input time series with phenomenon events extracted from output time series. Given a fixed output time series containing one or more Phenomenon events, it is shown under idealized conditions that the Granger Causality Test is effective for ranking candidate time series according to the likelihood that Precursor Rules exist. Using MIB (Management Information Base) datasets collected from real experiments involving Distributed Denial of Service Attacks, it is shown that Precursor Rules relating activities at attacking machines with traffic floods at target machines can be extracted by the methodology.

Keywords: Intrusion Detection, Distributed Denial of Service Attacks, Temporal Data Mining, Statistical Methods, Time Series Analysis, Network Management Systems

1. Introduction

The past fifteen years have witnessed a dramatic increase in society's dependence on networked information systems, and also the growing connectivity of these systems. These two factors have substantially increased both the potential damage which can be inflicted to critical resources, and the opportunities to inflict these damages through attacks on computer networks ([34]).

The failure of authenticators and other protection mechanisms to provide adequate defense to attacks against Information Systems, and the resulting mistrust in these mechanisms are the most important driving forces behind the development of Intrusion Detection Systems in the past twenty years - [16], [40]. IDSs are a relatively new development in Information Assurance; prevention has dominated the field for a long time (eg. [38]). The recent realization that prevention alone is not sufficient to protect complex Information Systems led academic researchers and manufacturers alike to also consider Detection and Response as integral parts of a working framework for Information Assurance (eg. [36]). As

noted in [2] and [23], the reasoning behind this shift was that responses triggered by detection would ultimately result in more secure systems, *even in the absence of better preventive security measures.* As such, current IDSs are not preventive security measures, and are most often used in conjunction with protection mechanisms, such as firewalls, smart cards and virtual private networks ([23]). The combination of several disparate mechanisms into a common security architecture constitutes the Defense-in-Depth Paradigm discussed in detail in ([39]), which is currently in vogue.

The rules produced by current IDSs are passive in the sense that a security violation has to occur in order to be detected. If we are fortunate to have detection happening early enough, it may be possible to minimize, or even eliminate the deleterious effects of the security violation. But early detection in current IDSs is usually the result of incidental circumstances, not of systematic design. On the other hand, almost all security violations encountered in practice evolve in multiple stages, and some of the preliminary stages may not be destructive per se, but merely preparatory steps in the Attack Scenario. If one could detect indicators of these preparatory steps, or attack precursors, and take immediate action, the resulting attack would be prevented. We call this capability Proactive, or Anticipatory Intrusion Detection, to distinguish it from the passive detection enabled by current IDSs. If successful, Proactive Intrusion Detection would be an invaluable enabling capability for Response, since enough time would be available to respond to the destructive phase of the attack, ideally preventing it to ever take place. A reasonable military analogy would be a preemptive strike against an enemy showing signs of increasing combat readiness. Another - less obvious - application of Proactive Intrusion Detection is in reducing the False Alarm Rates of Passive Intrusion Detectors through Event Correlation. Intuitively, an alarm raised by a passive IDS which is preceded by an identified precursor should be given more weight than an alarm which is not preceded by precursors.

The chapter is organized as follows: in section 2 we introduce Proactive Intrusion Detection through a Thought Experiment ([32]). We attempt to present the general ideas, contrasting Proactive Intrusion Detection with classical Intrusion Detection and protection techniques. This section also describes the possible roles that can be played by the techniques in computer forensics and in the response to security violations. Section 3 presents a methodology for designing Proactive Intrusion Detectors using data-based techniques. Section 4 presents a real example in which the methodology was applied for the extraction of

precursors to Distributed Denial of Service attacks. Section 5 closes the chapter, with our conclusions.

2. Information Assurance, Data Mining, and Proactive Intrusion Detection

2.1 Intrusion Detection Systems

Several taxonomies have been proposed for IDSs (eg. [4], [15]), but in this section we only distinguish between two classes of IDSs: IDSs relying on probabilistic reasoning applied to data collected from an Information System, and IDSs relying of domain knowledge concerning the operation of the Information System and the attacks against it. Data-based IDSs (the former type) are usually associated with Anomaly Detectors, designed on basis of statistical models of normal operation of a system, while Domain Knowledge-based IDSs are usually associated with Pattern Matching, designed on basis of the signatures of the attacks. It is worth pointing out however that data-based techniques can be applied to obtain IDS rules associated with both normal and abnormal behavior (eg.[7], [28]), while the specified normal behavior of a system can be encoded and utilized for detecting abnormalities (eg. [24], [45]).

We focus this chapter on data-based IDSs. Data-based techniques have been applied to IDS design since the inception of the field - [16], [31]. A relatively complete survey can be found in [4]. As noted earlier, mainstream efforts have focussed on Anomaly Detection schemes (eg. [11], [20], [21], [26], [35]), where a model of normal operation is extracted from the data, and deviations from the model are flagged as security violations. Regardless of the specific data-based method utilized in the design of an IDS, it is always possible to express its operation in the form of a concatenation of if-then rules of the form $X \Rightarrow Y$, where X is the antecedent of the rule, and Y is the consequent of the rule ([1]). Following [44], we view Data Mining as the "discovery of useful summaries of data"; as such, classical applied statistics, machine learning and pattern recognition are also manifestations of Data Mining. When time stamps are associated with the data items, we have ordered data sets. The extraction of rules in this case is studied under the heading of Temporal Data Mining (eg. [5], [33]).

2.2 A Thought Experiment

In the course of our research, we have been able to verify the validity of the concept of Proactive Intrusion Detection on a limited basis - [8], [9] - as summarized in section 4. The purpose of this section is

to introduce the reader to the concept of Proactive Intrusion Detection, emphasizing the unique aspects of the concept in contrast to classical (Passive) Intrusion Detection. To fix ideas, we will describe a thought experiment ([32]) in which we hope the roles and limitations of Authenticators, Passive Intrusion Detectors and Proactive Intrusion Detectors will become clear. We will also use the example to illustrate the general design issues in Proactive Intrusion Detection. The analogies between the various agents in the experiment and the typical environment encountered in the defense of Information Systems are mostly self-evident; we will make use of these analogies in the remainder of the discussion.

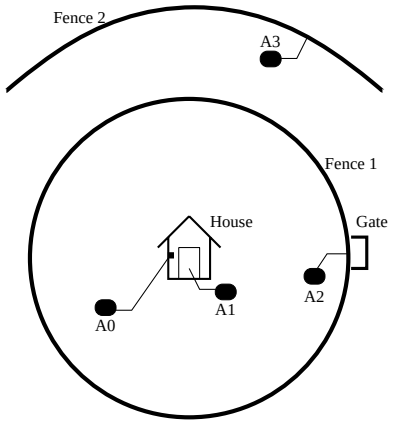


Figure 8.1. Authentication, Passive Intrusion Detection and Proactive Intrusion Detection. Alarm $A0$ is connected to an Authenticator. Alarm $A1$ enables Passive Intrusion Detection, while alarms $A2$ and $A3$ enable Proactive Intrusion Detection.

The residents of the house in Figure 8.1 and their guests are authorized to enter the house at any time. The residents agree on a password, and inform their guests of the password, as needed. We call the residents and their guests the Authorized Personnel. Upon arrival in the front door, incoming subjects enter the password on a keypad. If the password is entered correctly, the door opens. If not, the door remains closed, and the incoming subject is denied entry into the house. We say that an incoming subject passed the Authentication Test (eg. [38]) if she or he was able to key-in the right password. Once inside the house, the subject can perform regular activities or malicious activities. It is assumed that time elapsed between the entry of a malicious subject in the house and malicious activity is negligible. The objective of the authentication mechanism described above is to eliminate malicious activities, while assuring access to the house to all Authorized Personnel. This objective will be met under the following assumptions:

- **Assumption 1**: Authorized Personnel always key-in the right password.
- **Assumption 2**: The Password Checker cannot be tampered, neither the password be guessed by an automated trial-and-error procedure.
- **Assumption 3**: Authorized Personnel does not disclose the password to undesirable subjects.
- **Assumption 4**: Authorized Personnel does not perform malicious activities.

Assumption 1 implies that Authorized Personnel will always be able to enter the house. Assumptions 2-3 identify Authorized Personnel with subjects that passed the Authentication Test, and Assumption 4 identifies Authorized Personnel with subjects that only perform normal activities. In an ideal world, the authentication mechanism described above is sufficient to protect the house against malicious activities. This is a prevention mechanism; failure of such mechanisms lead to their mistrust, and the consequent search for Detection/Response schemes. Assume now that malicious activity was found to occur while the authentication mechanism is in place. There are three possible explanations:

- **Assumption 2 was violated**: The Password Checker was bypassed by a malicious agent - **Hacking**.
- **Assumption 3 was violated**: A member of Authorized Personnel, intentionally or non-intentionally, let the password be known

to a malicious agent, who entered the house - **Social Engineering**.

- **Assumption 4 was violated**: A member of Authorized Personnel performed malicious activities - **Insider Misuse**.

Hacking, Social Engineering and Insider Misuse are very prevalent in Information Systems. Intense market competition leads systems and application software to be introduced at a fast pace, and this leads to software containing unknown vulnerabilities which can be exploited by malicious intruders. Hacking is the layman term to describe this phenomenon. Social Engineering and Insider Misuse are human phenomena, very hard to describe analytically, and extremely harmful. The National Center for Computer Crime Data reports that in 1998 only about three percent of serious incidents in a typical business environment came from "outside" in the form of Intrusions ([41]); Insider Misuses on the other hand accounted for nineteen percent of these incidents[1]. Insider Misuse is also known to lead almost invariably to larger losses, specially when privileged users are involved ([41]).

One first step towards a defense mechanism is to install an alarm system connected to the Authenticator. We call this alarm $A0$. It goes off whenever an incoming subject is denied access to the house after entering the password. Figure 8.2 shows the evolution of $A0$ with time. As shown in the Figure, malicious activity is verified at time instants $T1$, $T2$, $T3$ and $T4$. $A0$ goes off a little before $T1$ and $T4$, which *may* indicate that the incoming subject was trying out the password before getting it right. But there is no logical basis for this inference; it may just be a coincidence. $A0$ is a rudimentary IDS, but incapable - by definition - of detecting malicious activity, since its behavior is predicated on Assumptions 2-4 that identify authorization to enter the house with normal activity. Detection of malicious activities is only possible if one mistrusts Assumptions 2-4 and accept the possibility of Hacking, Social Engineering or Insider Misuse. The key step in the design of a data-based IDS is the construction of a statistical model of the behavior of incoming subjects. In Anomaly Detection, one attempts to construct models of the behavior of Authorized Personnel. If variables are monitored during a period when malicious activity was verified, it is also possible to use the records to construct a model of malicious behavior, or, alternately, to design a classifier to distinguish normal from malicious behavior. In this simple example, the variables for such models could be the time of

[1] It is sobering to note that the remaining 78% of the incidents came from "natural causes", i.e. Bugs & Errors, as well as Disasters, such as power failures, etc.

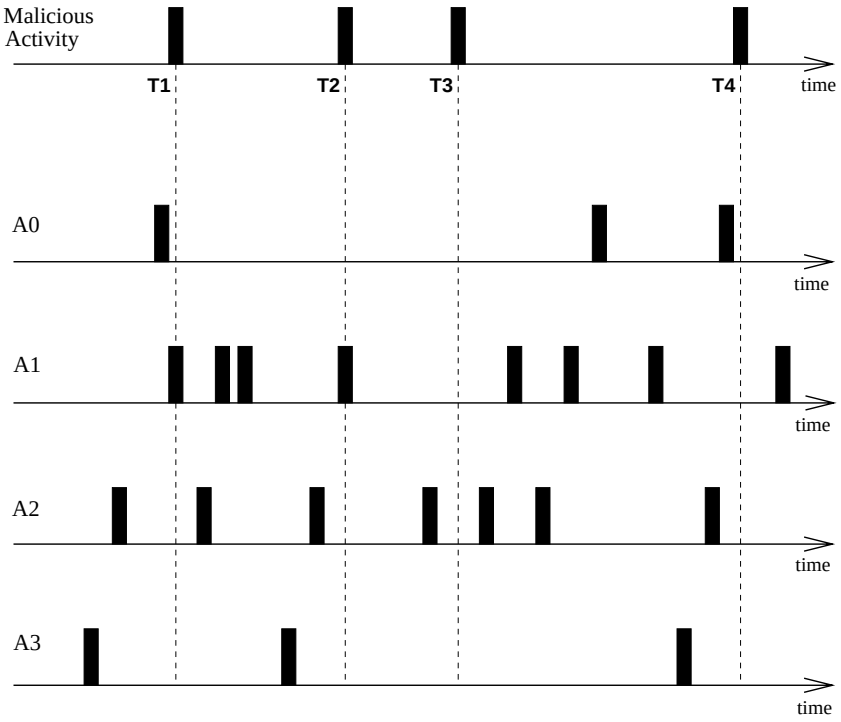


Figure 8.2. Timeline of the outputs of the alarms. It is assumed that the time elapsed between the entry of a malicious agent in the house and malicious activity is negligible.

entry (day, evening or night) and the typing speed in the keypad. The nature of the statistical model used for characterizing the behavior of incoming subjects, the nature of the variables used for model construction, and the size of the training and testing sets are all design issues in Intrusion Detection (eg. [4], [25]). The statistical model is parametrized by a threshold variable, associated with the rates of false alarm and detection, estimated from the available data sets. Let $A1$ denote the alarm connected to the resulting IDS. The operation of $A1$ is depicted in Figure 8.2. It is important to notice that the IDS was designed using data sets collected *before* the malicious activities at $T1$-$T4$ took place. $A1$ flags the malicious activity at $T1$ and $T2$, but misses $T3$ and $T4$. There are also several false alarms, a virtually unavoidable consequence of the statistical nature of the whole design process. There are two interrelated points to keep in mind, that characterize passive IDSs:

- **Static rules** The antecedent and the consequent items of the rule occur at the same instant of time. The design of a Passive IDS is a classical anomaly detection or classification problem.
- **Report immediate danger** If the antecedent is true, a security violation occurs instantaneously.

Consider now that the house is surrounded by a fence - Fence 1, placed at about 100 yards from the house, as shown in Figure 8.1. Authorized Personnel are given a key to a gate in this fence. Alarm $A2$ is attached to the fence. If an incoming subject uses the key to open the gate, nothing happens with $A2$. Alarm $A2$ goes off if someone crosses Fence 1 without going through the gate. Notice that the act of jumping the fence does not imply that the house is in danger. The incoming subject may walk towards the house or not, may attempt to enter the house or not. As Figure 8.2 shows, it is noticeable that $A2$ goes off sometime before each of the four instances when malicious activity was recorded. Now, it is also known that the time needed for someone to walk from Fence 1 to the house is consistent with the average delay between fence-crossing and malicious activity - about 2 minutes. It suggests the following if-then rule: Whenever $A2$ goes off, malicious activity will be observed within 2 minutes. There are three false alarms in this case, but the rule led to a 100% detection rate. We call these Causal Rules, which are special cases of the Temporal Rules formally introduced in the Data Mining literature in [1]. Assume finally that a few of the malicious subjects were apprehended, and revealed that they approached the house from behind, to avoid visual detection. Based on this information, a second, smaller fence - Fence 2 - was raised in the back of the house, placed at about 150 yards from the house. Alarm $A3$ is attached to this fence.

Whenever someone jumps through Fence 2, alarm $A3$ goes off. Figure 8.2 now suggests the following Causal Rule: Whenever $A3$ goes off, malicious activity will be observed within 3 minutes. This rule works for 3 out of 4 instances of malicious activity, without false alarms.

2.3 Proactive Intrusion Detection

A Proactive, or Anticipatory Intrusion Detector is a scheme that issues alarms based on Temporal Rules such as those described above. It is driven by events that are observed to consistently precede a security violation. In contrast with Passive Intrusion Detectors, Proactive Intrusion Detectors are characterized by:

- **Temporal rules** The antecedent and the consequent items of the detection rule occur at distinct instants of time; the antecedent precedes the consequent. The design of a Proactive IDS is a problem in Time Series Analysis ([10], [14]).
- **Report incoming danger** If the antecedent is true, a security violation occurs within a certain amount of time.

The extraction of Temporal Rules is performed off-line. Large data sets recorded from the multiple domains of the Information System are analyzed, in search of rules relating security violations at the Target with variables that *may* contain precursors. These data sets include Attack Runs, in which Attacks were known to be present, and also Normal, or Attack-Free Runs, used to construct profiles of normal operation. Figure 8.3 depicts the proposed scheme. The Temporal Correlation Engine is presented with the evolution of the network states, and the history of security violations. The Engine extracts Temporal Rules, relating network states that precede security violations. The design of a Proactive Intrusion Detector follow four steps, which will be formalized and examined in detail in section 3:

- **Step 1**: Determining the variables at the targets that characterize the security violation. This is performed by using domain knowledge about the type of attack, or by data-based methods. This step essentially corresponds to the design of a Passive IDS. In our Thought Experiment, Step 1 corresponds to determining the variables characterizing the behavior of the incoming subject.
- **Step 2**: Determining key variables containing attack precursors. This is effected through Statistical Causality Tests, that measure the relative causality strengths between a number of candidate variables and the variables determined in Step 1. In our Thought

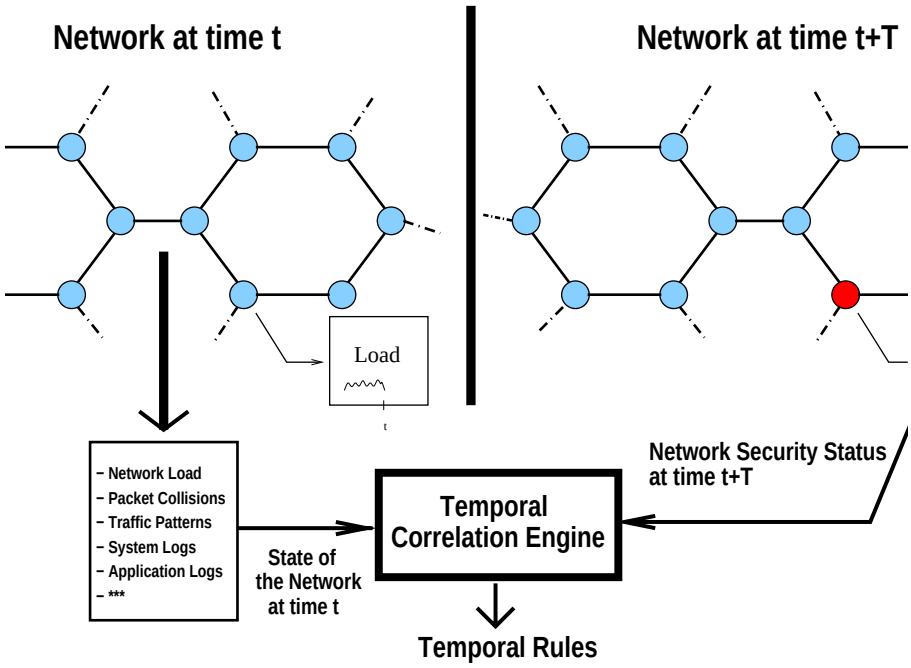


Figure 8.3. Proactive Intrusion Detection - Extracting Temporal Rules.

Experiment, Step 2 corresponds to discovering that the alarm variables $A2$ and $A3$ associated with jumps across Fence 1 and Fence 2 contain useful information to forecast future malicious activity at the house.

- **Step 3**: Determining the events most likely to be the ones preceding the security violation, or the precursor events. This is performed by comparing the evolution of the variables determined in Step 2 with their normal profile. Deviations from the profile characterize the precursors we are seeking. In our Thought Experiment, these precursor events are the jumps in the alarm outputs at $A2$ and $A3$.
- **Step 4**: Verifying if the precursor events extracted in Step 3 are consistently followed by the security violations observed in the variables determined in Step 1.

Remark 1 (Proactive Intrusion Detection and Computer Forensics) Computer Forensics is an emerging discipline lying at the intersection of Computer Security and Law Enforcement (eg. [2], [37]). Computer Forensics starts with the fact that a security violation has occurred, and attempts to gather the evidence needed by the investigators to identify the culprits, and allow their prosecution. The precursors determined in Step 2 above have the potential to be used in Computer Forensics as corroborative evidence that an intrusion took place (eg. [6]).

Remark 2 (Temporal Data Mining and Intrusion Detection - Related Work) The time component of a data record has been used for model building in previous work in Intrusion Detection - eg. [27], where time is seen as a discriminant feature for designing classifiers. In [29] a framework is introduced, allowing for the construction of rules corresponding to different time granularities and possibly non-contiguous time intervals (eg. all Tuesday mornings during a year, the first five minutes after market opening during weekdays in the Summer, etc). However, in both cases the market basket set-up ([1]) is followed, i.e. one seeks for association rules where the antecedent and consequent itemsets belong to the same time interval (fixed and contiguous in [27], possibly varying and non-contiguous in [29]).

3. A methodology for discovering precursors - Assumptions, Objectives, Procedure and Analysis

3.1 Notation and Definitions

3.1.1 Time Series, Multivariate Time Series and Collections. A time series is an ordered finite set of numerical values recorded from a variable of interest. It is assumed that the time elapsed between the recording of two consecutive elements is constant.

The kth element of the time series $\{z(k)\}$ is denoted as $z(k)$, where $k = 0, 1, \cdots, N-1$ and N denotes the number of elements in $\{z(k)\}$. A multivariate time series of dimension m is an ordered finite set of $m \times 1$ numerical vectors collected from m variables of interest. The kth element of the multivariate time series $\{Z(k)\}$ is denoted as $Z(k) = [z_1(k)\ z_2(k) \cdots z_m(k)]^T$, where $z_i(k)$, $i = 1, 2, \cdots, m$ and $k = 0, 1, \cdots, N-1$ are the kth elements of the individual time series z_i that form Z. It is assumed that all $z_i(k)$ are recorded at the same instant of time, which allows the common index k to be used for their specification. The multivariate time series $\{Z(k)\}$ is represented as an $m \times N$ matrix of numerical variables. We call m the dimension of $\{Z(k)\}$, and N its size. A collection of multivariate time series is a finite set of multivariate time series corresponding to the same variables, but not necessarily having the same size. The jth element of the collection $\mathcal{Z}$ is denoted as Z^j, $j = 1, 2, \cdots, C$. Collections of multivariate time series will be associated with the regimes of operation of the system of interest. In the case of Network Security, $\mathcal{N}$ corresponds to normal network activity, while $\mathcal{A}$ corresponds to periods of time during which an attack is detected. $C_{\mathcal{N}}$ denotes the number of elements in the collection $\mathcal{N}$, while $C_{\mathcal{A}}$ denotes the number of elements in collection $\mathcal{A}$. Finally, we call the dataset $\mathbb{D}$ as the union of the two collections. $\mathbb{D}$ represents the Training Set, from where knowledge is to be extracted.

3.1.2 Events, Event Sequences, Causal Rules and Precursor Rules. Events are defined in [33] as an ordered pair (A, κ) where $\kappa = 0, 1, 2, \cdots, K-1$ is a time index representing the occurrence time of the event and $A \in \mathcal{E}$ is an Event Type. $\mathcal{E}$ is a finite set in [33], which we call the Event Alphabet. Event types provide a framework for transforming the raw time series data into more meaningful descriptions. As an example, consider the following procedure for transforming a time series $\{z(k)\}$ having an even size N, into an an event sequence $\{\epsilon(\kappa)\}$ with size $K = \frac{N}{2}$ and Event Alphabet $\mathcal{E} = \{E_1, E_2\}$:

- If $[z(k+1) + z(k)] \leq 200$, Then $\epsilon(\kappa) = E_1$, for $\kappa = \frac{k}{2}$ and $k = 0, 2, 4, \cdots, N-2$.
- Otherwise, $\epsilon(\kappa) = E_2$, for $\kappa = \frac{k}{2}$ and $k = 0, 2, 4, \cdots, N-2$.

$z(k)$ can denote the number of alarms issued by a network monitoring device during a single day. A system administrator may only be interested in monitoring a higher level alarm, defined by $\epsilon(\kappa)$, i.e. every couple of days check if more than 200 alarms were issued. If yes, a message is sent. Otherwise, nothing happens. The transformation from the time series

space into the event space is the process of Time Series Quantization. The selection of the "right" parameters for performing the Quantization depends on the problem at hand. If m time series $\{z_i(k)\}$, $k = 1, 2, \cdots, m$ are quantized according to the same procedure along the time index, producing m event sequences $\{\epsilon_i(\kappa)\}$, $i = 1, 2, \cdots, m$, we can define Multivariate Event Sequences and Collections of Multivariate Event Sequences the same way we defined their Time Series counterparts in section 3.1.1. It is understood that the events $\epsilon_1(\kappa) \in \mathcal{E}_1,\ \epsilon_2(\kappa) \in \mathcal{E}_2,\ \cdots\ \epsilon_m(\kappa) \in \mathcal{E}_m$ are all recorded at the same instant κ, although the individual Event Alphabets $\mathcal{E}_i$, $i = 1, 2, \cdots, m$ are not necessarily the same.

Definition 1 (Causal Rule - [14]) If A and B are two events, define $A \stackrel{\tau}{\Rightarrow} B$ as the rule: If A occurs, then B occurs within time τ. We say that $A \stackrel{\tau}{\Rightarrow} B$ is a Causal Rule. □

Definition 2 (Precursor Rule - [10]) If A and B are two events, define $A \stackrel{\tau}{\Leftarrow} B$ as the rule: If B occurs, then A occurred not earlier than τ time units before B. We say that $A \stackrel{\tau}{\Leftarrow} B$ is a Precursor Rule. □

Causal Rules and Precursor Rules are special cases of Temporal Rules, introduced in [1]. Clearly, the rules $A \stackrel{\tau}{\Leftarrow} B$ and $A \stackrel{\tau}{\Rightarrow} B$ are not the same. Notice that B is the antecedent of the Precursor Rule, while A is the antecedent of the Causal Rule. Hence, the confidence of the Precursor Rule - $c(A \stackrel{\tau}{\Leftarrow} B)$ - is the fraction of occurrences of B that were preceded by A within τ units. In the problem at hand, A and B are events recorded at two different event sequences. If $c(A \stackrel{\tau}{\Leftarrow} B) = 1$, it means that if B occurs, then A always occurred not earlier than τ units before B, and is therefore a precursor of B, in the usual sense of the word. It *does not* mean however that all occurrences of A are followed by B.

The proposed methodology discovers Precursor Rules of the type $A \stackrel{\tau}{\Leftarrow} B$ in $\mathbb{D}$, but utilizes the associated Causal Rule $A \stackrel{\tau}{\Rightarrow} B$ for detection. The reason for this procedure is clear: we first characterize the security violation (item B) and then search for the precursors (item A). In summary, we mine Precursor Rules, but apply Causal Rules.

3.2 Assumptions, Problem Set-Up, Objectives and Procedure

Assumptions

1 The variables are recorded as two collections of multivariable time series of dimension m. Collection $\mathcal{N}$ corresponds to normal operation, while collection $\mathcal{A}$ corresponds to abnormal operation. Typically $C_{\mathcal{A}} \ll C_{\mathcal{N}}$.

2 The m variables can be split into two subsets: Output variables y_i, $i = 1, 2, \cdots, m_1$ and **candidate** input variables u_i, $i = 1, 2, \cdots, m_2$, with $m_1 + m_2 = m$.

3 The output variables are the ones in which the phenomenon of interest manifests itself.

4 The Phenomenon is only observed at collection $\mathcal{A}$.

5 The candidate input variables correspond to variables that may be or may not be related to the occurrence of the phenomenon observed in the output variables.

Given the assumptions above, we identify three interrelated problems related to the extraction of knowledge from the dataset $\mathbb{D}$. To simplify our discussion, it is assumed that $m_1 = 1$, i.e. the Phenomenon is only observed on a sigle output.

Problem 1: Phenomenon Characterization. Given the output time series, Phenomenon Characterization is related to the definition of a suitable Event Space, through Time Series Quantization. Let us return to the example in section 3.1.2. Define the quantity $\zeta(\kappa) := z(k) + z(k+1)$, for $\kappa = \frac{k}{2}$ and $k = 0, 2, 4, \cdots, N-2$, and the time series $\{^{\mathcal{N}}\zeta^j(\kappa)\}$, $j = 1, 2, \cdots, C_{\mathcal{N}}$, and $\{^{\mathcal{A}}\zeta^j(\kappa)\}$, $j = 1, 2, \cdots, C_{\mathcal{A}}$, which are the ζ time series belonging to Collections $\mathcal{N}$ and $\mathcal{A}$. Finally, define $\text{Max}(^{\mathcal{A}}\zeta^j) := \max_\kappa\{^{\mathcal{A}}\zeta^j(\kappa)\}$, $\text{Max}(^{\mathcal{N}}\zeta^j) := \max_\kappa\{^{\mathcal{N}}\zeta^j(\kappa)\}$, and the *overall* maxima $\text{Max}(\mathcal{A}^\zeta) := \max_j(\mathcal{A}^\zeta)$, $\text{Max}(^{\mathcal{N}}\zeta) := \max_j(^{\mathcal{N}}\zeta^j)$. If $\text{Max}(^{\mathcal{N}}\zeta) = 100$, and $\min_j \text{Max}(^{\mathcal{N}}\zeta^j) = 300$, a threshold of say, 200 separates the two collections. Event sequences constructed using the procedure in the example of section 3.1.2 will be such that the event type E_2 never occurs on time series belonging to collection $\mathcal{N}$, and will occur at least once in all time series belonging to collection $\mathcal{A}$. If the occurrences of E_2 are time-localized, i.e. $^{\mathcal{A}}\epsilon^j(\kappa) = E_1$ most of the time, except for few isolated spikes where E_2 is present, then E_2 is a good characterization of the Phenomenon of interest. In many problems in computer security the problem of Phenomenon Characterization is very simple. As shown in section 4, the output variables related to traffic counting in machines that are targets of Denial of Service Attacks records readings of 50,000 units during an Attack, compared with about 100 units during normal operation. Also, these bursts are time localized, i.e. the time series in collection $\mathcal{A}$ remain at readings of about 100 units (similar to collection $\mathcal{N}$), except for the bursts characterizing the presence of the attack.

Problem 2: Identifying the Input Variables. The objective is to select which among the m_2 variables contain precursors for the phenomenon observed in the output. The objective is to obtain time series for which high confidence Precursor Rules of the type $A \overset{\tau}{\Leftarrow} B$ exist, where B is the Phenomenon Characterized in Problem 1, while A is an event extracted from the candidate time series. Notice that τ is not known. Hence, it is not advisable at this stage to do Time Series Quantization at the candidate inputs, as valuable Precursor Information may be destroyed in the process. Clearly, we need a procedure capable of performing the following two tasks: **(1) Detection**: Given an input-output pair $(\{u(k)\}, \{y(k)\})$ measure the likelihood that a rule of the type $A \overset{\tau}{\Leftarrow} B$ exists, where A is a Precursor extracted from $\{u(k)\}$ and B is a Phenomenon extracted from $\{y(k)\}$ *without knowing the true nature of the Precursor, or the delay between Precursor and Phenomenon*; **(2) Gradation**: Given a fixed output and m_2 candidate inputs, rank candidate input variables according to the likelihood that Precursor Rules exist, *without knowing the true nature of the Precursor, neither the delay between Precursor and Phenomenon.* We show in the sequel that the Granger Causality Test is an adequate procedure for addressing both tasks.

Problem 3: Precursor Characterization. Given the input variables that are most likely to contain Precursors, the problem of Precursor Characterization is to extract the Precursors as time-localized occurrences in the time series through a process of Time Series Discretization of the same nature as Problem 1. The key point however, is that following the solution of Problem 2, one has evidence that these time-localized occurrences give rise to Event Types that are related to the Phenomenon at the output through a rule of the type $A \overset{\tau}{\Leftarrow} B$.

Procedure

Based on the above, we suggest the following procedure for extracting Precursor Rules relating Phenomenon in the outputs with Precursors at candidate inputs:

- **Step 1**: Solve Problem 1 through adequate Time Series Quantization at the output time series.
- **Step 2**: Solve Problem 2 by applying the Granger Causality Test (GCT) to all input-output pairs $\{(u_i(k), y(k)\})$, $i = 1, 2, \cdots, m_2$, and compute the GCI g_i corresponding to each candidate input. Select candidate inputs either by setting a threshold on g_i, or by choosing the top v scores, where typically $v \ll m_2$.

- **Step 3**: Solve Problem 3 through adequate Time Series Quantization of the input time series selected on Step 2. The objective is to extract time-localized structures at the selected inputs that precede the Phenomenon in the output. Discard input time series that do not show time-localized structure preceding the Phenomenon. At the end of this step, one has determined the **Phenomenon** and **Precursor** events of interest, as well as a number of candidate Precursor Rules of the form **Precursor** $\stackrel{\tau}{\Leftarrow}$ **Phenomenon**.

- **Step 4**: Compute the confidence of the *associated* Causal Rules **Precursor** $\stackrel{\tau}{\Rightarrow}$ **Phenomenon**, and select the best ones either by thresholding or ranking. At this Step we are verifying if indeed the Precursor events at the inputs are preceding the Phenomenon events at the output.

In the next section we will describe the Granger Causality Test, and discuss its suitability as an Exploring Tool for Knowledge Discovery, targeted on Step 2 above. A more complete discussion of the theoretical developments can be found in [10]. In section 4 we summarize the results obtained when applying this procedure for the extraction of Precursor Rules relating Attacking Nodes with Target Nodes in various types of Distributed Denial of Service Attacks. Our complete results can be found in [8] and [9].

3.3 Analysis - Detection and Gradation of Causality in Time Series

3.3.1 Notation and Definitions. The following notation and nomenclature are used extensively in the following sections. It is commonly used in Statistics and Systems Science. The reader is referred to standard textbooks in these areas for more details - eg. [19] or [22].

Shift Operators, Transfer Functions and Impulse Responses. Given a time series $\{z(k)\}$, $k = 0, 1, \cdots, N-1$, the backward and forward shift operators q and q^{-1} are defined as follows: $qz(k) := z(k+1)$, $k = 0, 1, \cdots, N-2$ and $q^{-1}z(k) := z(k-1)$, $k = 1, 2, \cdots, N-1$. The backward and forward shift operators are used to describe dynamical input-output relations among variables. In particular, the expression $y(k) = \frac{\beta(q^{-1})}{\alpha(q^{-1})}u(k) = T(q^{-1})u(k)$ where $\alpha(q^{-1}) = 1 + \sum_{\ell=1}^{p} \alpha_\ell q^{-\ell}$, $\beta(q^{-1}) = \sum_{\ell=0}^{p} \beta_\ell q^{-\ell}$ denotes $y(k) = -\sum_{\ell=1}^{p} \alpha_\ell y(k-\ell) + \sum_{\ell=0}^{p} \beta_\ell u(k-\ell)$, for $p+1 \leq k \leq N-1$. $T(q^{-1})$ is called the Transfer Function between $\{u(k)\}$ and $\{y(k)\}$. $T(q^{-1})$ is a stable Transfer Function if $\alpha(q^{-1})$ is a Hurwitz polynomial, i.e. all the zeros of $\alpha(q^{-1})$ belong to the open unit

disk. In this case, we can write $\frac{\beta(q^{-1})}{\alpha(q^{-1})} = t(q^{-1}) = \sum_{\ell=0}^{\infty} t_\ell q^{-\ell}$ and $t(q^{-1})$ is called the Impulse Response associated with the Transfer Function $T(q^{-1})$. The relationship between $\{u(k)\}$ and $\{y(k)\}$ is written in terms of the Impulse Response as $y(k) = \sum_{\ell=0}^{\infty} t_\ell u(k-\ell)$.

Probability Distributions. $x \sim X$ indicates that the random variable x has the distribution X. $\mathbb{E}(x)$ denotes the expected value of x. $n(\mu, \sigma^2)$ denotes a Gaussian, or Normal Random Variable with mean μ and variance σ^2. $F(v_1, v_2)$ denotes an F distribution with parameters v_1 and v_2. $\Gamma^{-1}(v_1, v_2)$ denotes an Inverse-Gamma distribution with parameters v_1 and v_2. The definitions and properties of these distributions are given in [17].

3.3.2 The Granger Causality Test as an Exploratory Tool.

Testing for causality in the sense of Granger involves using statistical tools for testing whether *lagged* information on a variable u provides any statistically significant information about the variable y. If not, then u does not Granger-cause y. The Granger Causality Test (GCT - [18]) compares the residuals of an AutoRegressive Model (AR Model) with the residuals of an AutoRegressive Moving Average Model (ARMA Model). Assume a particular lag length p, and estimate the a_i and b_i parameters $(1 \leq i \leq p)$ in the following unrestricted equation:

$$y(k) = \sum_{i=1}^{p} \alpha_i y(k-i) + \sum_{i=1}^{p} \beta_i u(k-i) + e_1(k) \tag{8.1}$$

Parameter estimation is performed using Ordinary Least Squares (OLS) - [19]. If $\{y(k)\}$ and $\{u(k)\}$ are time series of size N, it results on a regression with $T := N - p$ equations, out of which $2p$ parameters are estimated. The computational cost of the procedure is $O(T^2)$. The Null Hypothesis H_0 of the GCT is given by:

$$H_0 : \qquad \beta_i = 0, \quad i = 1, 2, \cdots, p,$$

i.e. u does not affect y up to a delay of p units. The null hypothesis is tested by estimating the parameters of the following restricted equation

$$y(k) = \sum_{i=1}^{p} \delta_i y(k-i) + e_0(k) \tag{8.2}$$

Again, estimation of the δ parameters lead to an OLS problem with T equations. The procedure of the GCT is as follows. Let R_1 and R_2

denote the sum of the squared residuals under the two cases:

$$R_1 = \sum_{k=1}^{T} e_1^2(k), \;\; R_0 = \sum_{k=1}^{T} e_0^2(k)$$

If the Granger Causality Index (GCI) g given by:

$$g = \frac{(R_0 - R_1)/p}{R_1/(T - 2p - 1)} \sim F(p, T - 2p - 1) \tag{8.3}$$

is greater than the specified critical value for the $F-$test, then reject the null hypothesis that u does not Granger-cause y. As g increases, the $p-$value[2] associated with the pair $(\{u(k)\}, \{y(k)\})$ decreases, lending more evidence that the Null Hypothesis is *false*. In other words, high values of g are to be understood as representing strong evidence that u is causally related to y. In this work, we utilize the GCT in a exploratory manner, to compare the causality strength of two candidate input time series with respect to a given output. Following the $p-$value interpretation, we say that $\{u_1(k)\}$ is more likely to $\{u_2(k)\}$ to be causally related with $\{y(k)\}$ if $g_1 > g_2$, where g_i, $i = 1, 2$ denote the GCI for the input-output pair (u_i, y). We may be interested in selecting the top 5 or 10 individual candidate input time series that are more likely to be causally related to $\{y(k)\}$ for more detailed inspection. The GCI is an adequate index to perform this selection.

3.3.3 GCT and the Extraction of Precursor Rules - Modeling and Theoretical Developments. In usual statistical practice, the GCT is utilized to decide if a given u causes y for a specified significance level. No assumption is made about the presence (or absence) of localized structure in the time series. In these cases, one is interested in gauging how the time series u affects the time series y as a whole. However, in the problem at hand, we are ultimately interested in extracting rules relating time-localized segments of the time series. In our context, the GCT is merely an intermediate step in this process. We argue as follows: if time-localized structures at u consistently precedes time-localized structures at y, there is good evidence that events in u are related to events in y. The localized structures will be examined separately at u and y after the existence of a causality relation between the two time series is suggested by GCT. The determination of these structures correspond to Steps 1 and 3 in section 3.2. In this section, we

[2] The $p-$value of a Statistical Test is the smallest significance level that leads to the rejection of the Null Hypothesis - [12], p. 364.

investigate how GCT behaves when time series with localized structure are tested for causality. If GCT is used to identify the presence of a causal relationship between two variables, it is implicitly assumed that these variables are related by a model of the form (8.1), where $e_1(k)$ is a noise term that captures the mismatch between the recordings and the model output. Let y represent the output variable that displays the phenomenon, and u represent the unknown input variable which triggers the phenomenon. Following section 3.2, in [10] we model the relationship between u and y as follows:

$$y(k) = q^{-r}\gamma H(q^{-1})u(k) + w(k) \tag{8.4}$$

r represents the delay between the input and the output, γ is an amplification gain typically large [3] ($\gamma \gg 1$), while $H(q^{-1})$ models the dynamic interaction between u and y. In the following, we consider the simpler case where $H(q^{-1}) = 1$, i.e. u and y are related by the expression:

$$y(k) = \gamma u(k - r) + w(k) \tag{8.5}$$

The general case is treated in [10]. It is assumed that an upper bound ρ for r is available, i.e. we are only interested in Precursor Rules of the form $A \overset{\rho}{\Leftarrow} B$, where $1 \leq r \leq \rho$. Knowledge of ρ is needed for selecting the prediction window p for performing GCT. The obvious choice for p is to take $p = \rho$. The event A is to be mined from u, while the event B is to be mined from y. $w(k)$ is the noise process, which we assume to be zero mean Gaussian white noise with variance σ^2, i.e. $w(k) \sim n(0, \sigma^2)$, for all k. We expect this model to be a good description for collection $\mathcal{A}$. To complete the modeling, we need to characterize the input signal u. Since we will be looking for time-localized events, we assume that u has a time-localized structure, as depicted in Figure 8.4. In particular, $u(k)$ is defined as:

$$\begin{aligned} u(a_i) &= A_i,\ i = 1, 2, \cdots, n, \text{ where } a_1 < a_2 < \cdots < a_n < N - p, \\ &\text{and} \quad a_i - a_{i-1} > \rho,\ i = 1, 2, \cdots, n \\ u(k) &= 0, \text{ for } k \neq a_i \end{aligned} \tag{8.6}$$

Here, N is the size of the collected dataset. In the absence of noise ($w(k) \equiv 0$), the output $y(k)$ essentially follows its impulse response each time inputs are applied at a_i, $i = 1, \cdots, n$, as depicted in Figure 8.4. The blips at u happening at each time sample a_i model the Precursors, while the response at y according to the Impulse Response of $\gamma H(q^{-1})$ models the Phenomenon. If the GCT is applied to time series coinciding

[3] As described in section 3.2, it is expected that the Phenomenon will be much more pronounced than the Precursor.

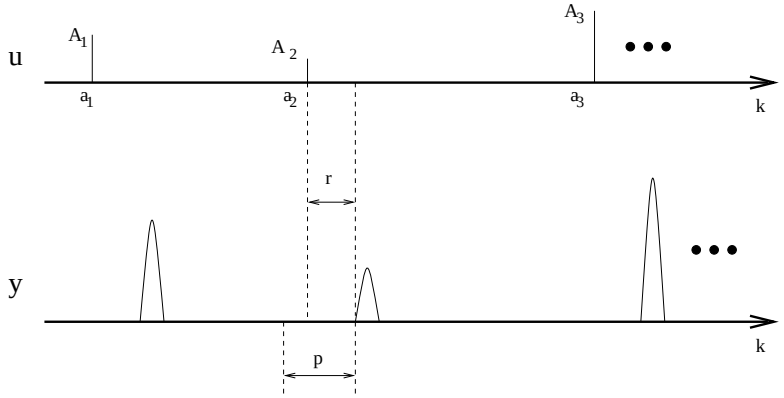


Figure 8.4. The idealized inputs and output signals. p is the length of the window used for parameter estimation when applying GCT. If $p \geq r$, the representation (8.1) captures model (8.4) exactly, and the **Precursor** $\Rightarrow$ **Phenomenon** rule is "visible" through the model. When $H(q^{-1}) = 1$, the response in y collapses into a blip.

with the idealized signals, we will have $g = \infty$, since R_0 is finite, while $R_1 = 0$, assuming that the parameter γ is exactly estimated estimated in equation (8.1)[4]. Notice that the distance between two consecutive blips $(a_i - a_{i-1})$ is assumed to be larger than the delay between the blip and the response. This assumption serves to "isolate" each individual **Precursor** $\Rightarrow$ **Phenomenon** occurrence. When $\sigma > 0$, one is interested in evaluating how GCT performs, i.e. in determining the relationship between the GCI and the other quantities in the problem. g is a random variable in this case, so $\mathbb{E}(g)$ is the quantity of interest. We now relate $\mathbb{E}(g)$ with the key variables in the problem.

Theorem 1 (The GCI for the idealized signals) Assume that time series $\{y^*(k)\}$ and $\{u^*(k)\}$ with size N are generated from equation (8.5) with $u^*(k)$ given by equation (8.6) and $w(k) \sim n(0, \sigma^2)$, for $k = 0, 1, \cdots, N-1$. If $\gamma^2(\sum_{i=1}^{n} A_i^2) \gg \sigma^2$ and $p \geq r$, the expected value of GCI for the pair $(\{u^*(k)\}, \{y^*(k)\})$ computed using equation (8.3) can be approximated by:

$$\mathbb{E}(g^*) \approx \frac{N - 3p - 1}{N - p - 2} \gamma^2 \frac{(\sum_{i=1}^{n} A_i^2)}{p\sigma^2} \quad \square \tag{8.7}$$

Theorem 2 (The GCI for input signals missing a few blips) Assume $\{y^*(k)\}$ and $\{u^*(k)\}$ with size N satisfy the conditions in Theorem 1. Let $\mathcal{I} \subset \{1, 2, \cdots, n\}$ denote a non-empty collection of indices.

[4]This will be true for the idealized signals in this case - [30].

Define $u^{\mathcal{I}}(k)$ as follows:

$$u^{\mathcal{I}}(a_i) = A_i, \text{ if } i \in \mathcal{I}, \quad u^{\mathcal{I}}(k) = 0, \text{ otherwise} \tag{8.8}$$

i.e. $u^{\mathcal{I}}(k)$ has only a fraction of the blips present in $u^*(k)$. If $\gamma^2(\sum_{i\in\mathcal{I}} A_i^2) \gg \sigma^2$ and $p \geq r$, the expected value of GCI for the pair $(\{u^{\mathcal{I}}(k), y^*(k)\})$ computed using equation (8.3) can be approximated by:

$$\mathbb{E}(g^{\mathcal{I}}) \approx \frac{N-3p-1}{p}\gamma^2 \frac{(\sum_{i\in\mathcal{I}} A_i^2)}{(N-p-2)\sigma^2 + \gamma^2 \sum_{i\notin\mathcal{I}} A_i^2} \quad \square \tag{8.9}$$

The reader is referred to [10] for the proofs of Theorems 1 and 2. We have conducted several numerical experiments to evaluate the validity of the approximations in Theorems 1 and 2. The results were very satisfactory, as described in [10]. A number of key observation can be made, on basis of Theorems 1 and 2:

1 The term $S^* := \frac{(\sum_{i=1}^n A_i^2)}{p\sigma^2}$ can be understood as the Signal-to-Noise Ratio (SNR) between the input blips carrying the Precursors to be extracted, and the noise present in the estimation window of length p. $\mathbb{E}(g^*)$ grows with S^*, which intuitively means that for u signals of the type shown in Figure 8.4, if a larger value of g is observed, it indicates that it is more likely that Precursors could be found. This is certainly a desirable property.

2 The term γ^2 can be understood as an amplification gain between the input and the output. GCI is proportional to γ^2.

3 As $N \to \infty$, $\mathbb{E}(g^*)$ converges to $\gamma^2 S^*$, which is a constant. Hence, the corresponding expected $p-$value converge to zero. The interpretation is that for a fixed SNR, the certainty that the pair (u, y) is Granger causal grows with N. This is also a desirable property.

4 Finally, we turn our attention to equation (8.9). This expression allows one to relate $\mathbb{E}(g^{\mathcal{I}})$ with the confidence of Precursor Rules extracted from the pair $(\{u^{\mathcal{I}}(k)\}, \{y^*(k)\})$. If $\{u^{\mathcal{I}}(k)\}$ is selected in Step 2, the resulting Time Series Quantization in Step 3 is trivial: just define $\epsilon(k) = E_1$ if $u^{\mathcal{I}}(k) > 0$, $\epsilon(k) = E_2$, otherwise. Consider now the Precursor Rule $E_1 \overset{p}{\Leftarrow} B$, where B is obtained by performing Step 1 in $\{y^*(k)\}$. The confidence of this rule is given by $\frac{\#(\mathcal{I})}{n}$, where $\#(\mathcal{I})$ denotes the number of elements in $\mathcal{I}$. This follows from the fact that among the n times event B occurs, the E_1 event occurs only $\#(\mathcal{I})$ times. Consider now two time sequences $\{u^{\mathcal{I}_1}(k)\}$ and $\{u^{\mathcal{I}_2}(k)\}$ with $\mathcal{I}_1 \subset \mathcal{I}_2$, i.e. $\{u^{\mathcal{I}_2}(k)\}$

contains all the blips contained in $\{u^{\mathcal{I}_1}(k)\}$ plus a few more. It is clear from equation (8.9) that $\mathbb{E}(g^{\mathcal{I}_2}) > \mathbb{E}(g^{\mathcal{I}_1})$. It essentially means that higher values of g are associated with Precursor Rules with higher confidence, which is another property that indicates the suitability of the GCT as an Exploratory Tool for extracting Precursor Rules.

Remark 3 (Temporal Data Mining and Intrusion Detection - Related Work) The time component of a data record has been used for model building in previous work in Intrusion Detection - [27], where time is seen as yet one more itemset for rule discovery and [29], where a framework is introduced, allowing for the construction of rules corresponding to different time granularities (eg. all Tuesday mornings during a year, the first five minutes after market opening during weekdays in the Summer, etc). The objective in [29] is to construct profiles of normal activity at different levels of granularity. In both cases, the market basket set-up ([1]) is followed, i.e. one seeks association rules where the antecedent and consequent itemsets belong to the same time interval. In contrast, in the present work we are explicitly searching for rules where the antecedent and the consequent belong to different time intervals. □

4. A Case Study - Precursor Rules for Distributed Denial of Service Attacks

4.1 DDoS Attacks and the experiments

Distributed Denial of Service (DDoS) attacks have two phases, and involve three classes of systems: the Master, the Slaves, and the Target (eg. [13]). In the first phase of the attack, the Master infiltrates multiple computer systems, and installs the DDoS tools, which are scripts capable of generating large volumes of traffic under command from the Master. We call these infiltrated systems the Slaves. The second phase is the actual DDoS attack. Under command from the Master, the Slaves generate network traffic to bring down the Target system. We assume that the Master is not under monitoring, but the Target and a few Slaves (not all) are. Figure 8.5 presents a simplified timeline for the DDoS attacks. A Data Set for studying DDoS attacks was produced at North Carolina State University (NCSU). All the nodes (attackers and targets) were linked to a common Ethernet. The Network Management System collected 64 MIB variables corresponding to four SNMP MIB (Management Information Base - [42]) groups: `ip`, `icmp`, `tcp` and `udp`. Variables were collected for intervals of 2 hours, at a sample rate of 5 seconds. The details can be found in [8] and [9]. We used the data corresponding to

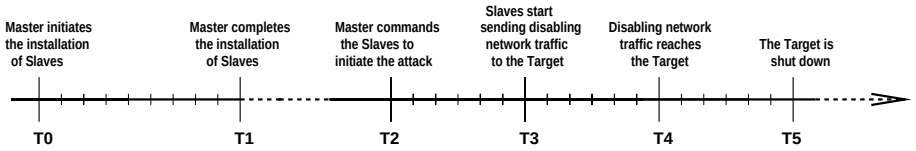


Figure 8.5. DDoS Attacks - A simplified Timeline.

both Attack Runs (Collection $\mathcal{A}$) and Normal Runs (Collection $\mathcal{N}$) as described below:

Attack Runs - Collection $\mathcal{A}$. Three types of DDoS attacks produced by TFN2K (Ping Flood and Targa3) and Trin00 (UDP Flood). During each of the attacks, MIBs were collected for the Attacking Machine and for the Target. TFN2K and Trin00 are the names of the hacker toolkits, while Ping Flood, Targa3 and UDP Flood are types of DoS attacks they induce. The time series for MIB variables corresponding to counter variables were differentiated. Two runs were recorded for each type of attack. According to the terminology introduced earlier, Attacker 1 and Attacker 2 are Slaves; the Master is not under monitoring from the Network Management System, so no time series are available for the Master.

Normal Runs Collection $\mathcal{N}$. MIBs were collected during times when the machines were not being targets of attacks, nor being the attackers. 12 runs are available for the Target Machine, 7 runs are available for Attacker 1, and 14 runs are available for Attacker 2. The data set includes events starting on T2, defined in Figure 8.5; the DDoS tools are assumed to be already installed in the Attacking Machines when the Attack Runs start. Hence, prospective Precursor Rules should relate events in T2 or T3 at the Attacker with events in T4 and T5 at the Target. To illustrate the nature of the MIB variables and their relevance for attack detection during a TFN2K Ping Flood Attack, Figure 8.6 depicts `icmpInEchos` at the Target, aligned with four MIB variables at the Attacker Machine that show remarkable activity before the pings reach the target. These are `ipOutRequests`, `icmpInEchoReps`, `tcpInErrs` and `udpInErrors`. These four variables were obtained from domain knowledge about the TFN2K Ping Flood attack. In practice, we need a procedure to extract these Key Variables for the Attacker automatically, from the entire collection of MIB data at the Attacker Machine. This is exactly the problem addressed in this chapter. In the sequel, we show the results obtained using the methodology in section

3.2 for the case of the TFN2K Ping Flood Attack. Similar results were also verified for the other two types of DDoS attacks ([8], [9]).

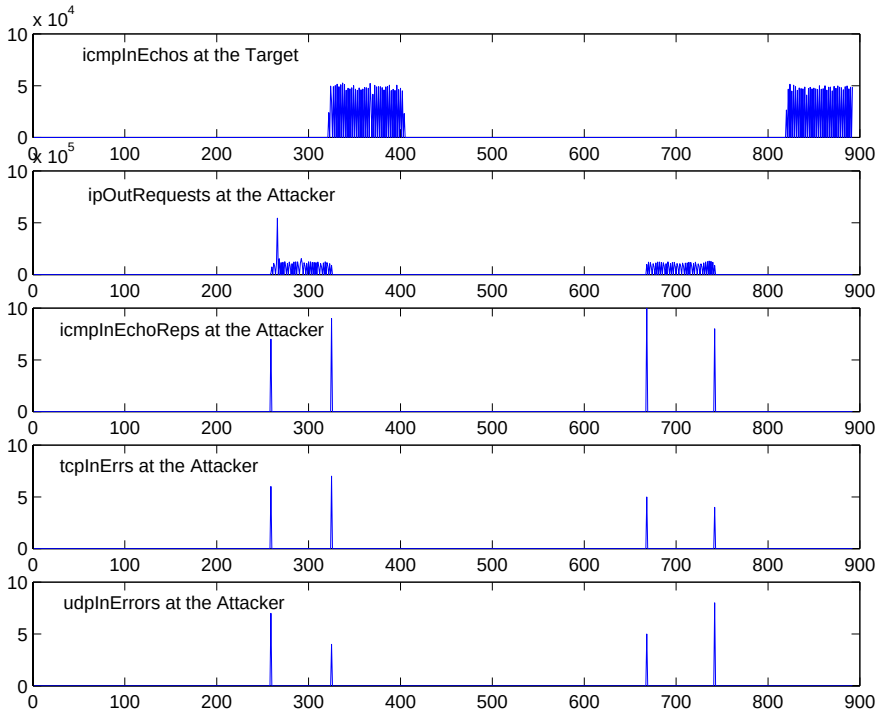


Figure 8.6. TFN2K Ping Flood: Selected MIB variables at the Attacker and at the Target.

4.2 TFN2K Ping Flood - Extracting Precursor Rules

The four steps presented in section 3.2 were followed:

Step 1: Phenomenon Characterization. The Ping Flood attack is effected by sending a large amount of ICMPECHOREQUEST packets to the Target. The variable `icmpInEchos` counts the number of ICMPECHOREQUEST packets received by a node. Hence, `icmpInEchos` is the output in this case. As shown in Figure 8.6, Phenomenon Characterization is very simple, considering that `icmpInEchos` never rises above a few hundred units during Normal Runs. On [11] it is shown that for large classes of Denial-of-Service attacks, traffic counting across various time

scales serve as an adequate discriminator between normal and abnormal behavior, as good as much more complex schemes.

Step 2: Extracting inputs containing Precursors. In this step, we attempt to determine the variables at the Attacker Machine that contain Precursors. Based on [13], we have domain knowledge (Ground Truth - Table 8.1) about the Key Variables at the Attacker for TFN2K. The GCT was applied for two runs of TFN2K Ping Flood. T4 events happen more than once in each run, as shown in Figure 8.6. To test the validity of the GCT for automatically extracting the Key Variables at the Attacker, we consider a scenario in which there are nine potential Attackers against the Target: the true attacker and eight decoys corresponding to the normal runs. We then apply the GCT to measure the causality strength of all MIB variables in the potential attackers, with respect to the Key Variable at the Target in each of the Attacks. MIB variables at potential attackers resulting on a GCI statistic above the threshold for 95% significance level were considered to Granger-cause the Key Variables at the Target, and where kept for analysis in Step 3. The selected variables and scores are shown in Table 8.2 for one of the Runs. Comparing Tables 8.1 and 8.2 it is clear that GCT extracted *all* Ground Truth Variables in this case. We count detections whenever the ground-truth variables described in [13] are correctly picked by the GCT. False alarms correspond to MIB variables being flagged in the decoys. Table 8.3 summarizes the results for both runs. Notice that at least one "true" MIB variable at the Attacker is detected in each run. The FA (False Alarm) Rate for Decoy MIBs is obtained by computing the total number of significant MIB variables found in all normal runs, divided by the total number of MIB variables.

Steps 3 and 4: Precursor Characterization and Determining the Confidence of the Causal Rules. The Key Variables at the Attacker determined in Step 2 are labeled as causally related with the Attack at the Target, but we still need to find the Precursors. As discussed in section 3.2, we have a problem of Time Series Quantization. We looked for jumps in the MIB variables, by monitoring the absolute values of the differentiated time series $z(k) = |y(k) - y(k-1)|$. Using 12 Normal Runs, we constructed a *Normal Profile of Jumps* for each of the 64 MIB variables. Given a Key Attacker Variable determined on Step 2, Key Events at the Attacker are defined as jumps larger than the largest jump encountered the *Normal Profile of Jumps*. Key Attacker Variables with no Key Events are discarded. As shown in Table 8.4, We have found that this procedure led to a substantial reduction of the

Table 8.1. Key Variables at the Attacker for TFN2K - Ground Truth.

MIB	Event
`icmpInEchoReps`	T2
`tcpInErrs`	T2
`tcpInSegs`	T2
`udpInErrors`	T2
`udpInDatagrams`	T2
`ipOutRequests`	T3

False Alarms produced on Step 2, with small reductions in the detection rates. Notice that we are still detecting at least one valid precursor at each Attack Run.

Remark 4 (Explaining the Precursor) According to [13], the communication between Master and Slave in TFN2K happens through ICMP, UDP or TCP. These ICMPECHOREPLY packets are the command from the Master to the Slave to initiate the attack, i.e. these constitute a T2 event, according to Figure 8.5.

Table 8.2. TFN2K Ping Flood Run 1: Top MIBs at the Attacker according to the g statistic.

Rank	MIB	g
1	`ipOutRequests` (T3)	5.26
2	`tcpInErrs` (T2)	3.50
3	`ipInReceives`	2.67
4	`ipInDelivers`	2.65
5	`udpInErrs` (T2)	2.63
6	`udpOutDatagrams`	2.58
7	`udpInDatagrams` (T2)	2.57
8	`icmpInEchoReps` (T2)	2.04
9	`icmpInMsgs`	1.99
10	`tcpInSegs` (T2)	1.31
11	`udpNoPorts`	1.27

Table 8.3. Results of Step 2: Detection Rates and FA Rates for MIB variables that contain precursors to DDoS Attacks.

Run	Detections	FA per Decoy MIBs (%)
1	6/6	4.49
2	1/6	3.13

Table 8.4. Final Results: Detection Rates and FA Rates for Events at MIB variables for TFN2K Ping Flood.

Run	Detections	FA per Decoy MIBs (%)
1	4/6	1.37
2	1/6	0.52

5. Conclusions

This chapter presents a principled approach for discovering precursors to security violations in databases recorded from large scale information systems. Proactive Intrusion Detection consists of the utilization of these precursors as part of an overall defense-in-depth scheme, including Prevention, Detection, Response and Tolerance.

While we consider the results in section 4 very encouraging, we are well aware that these were limited experiments, on a local test bed, under controlled traffic loads. We are currently experimenting with a larger networked testbed, including six hosts and three routers, instrumented by a Network Management System. Our results will appear in the near future.

Besides the obvious applicability of Proactive Intrusion Detection for response, we are also investigating the possibility of correlating the outputs of passive IDSs with the precursors. Current IDSs are plagued by high rates of false alarm ([23], [36]), explainable in part by the base rate fallacy of classical statistics ([3]), a result of the rarity of attacks in comparison with normal activity. The presence (or, rather, the absence) of reliable precursors may be used to prune false alarms from passive IDSs. A more elaborate scheme involving Alarm Correlation (eg. [46]) can also be tried. Clearly, extensive experimentation is needed to validate the concept.

We end this chapter on a cautionary tone. The use of statistical methods for extracting "subtle" precursor events for extraordinary phenomena such as Traffic Floods should give everyone pause. Mostly everybody has an anecdote relating the systematic occurrence of trivial facts preceding a major phenomenon. Some of these rules are so accurate that they find their way into the press. One such example is the *Super Bowl Predictor*, featured in [43], which supposedly forecasts the performance of the US stock market on basis of the result of the Super Bowl. No serious financial analyst would pay attention to this rule, in the same way as no serious network security analyst would pay attention to rules relating security incidents with variables that are known to be removed from the operation of the Information System. Hence, the message here is that common sense should be exercised in the choice of the input variables for temporal correlation, and domain knowledge about the system's operation must be utilized for pruning spurious rules. After a precursor rule is extracted, its root cause should be investigated, as described in Remark 4. An interactive, exploratory approach, coupling human interpretation of the rules with automation, will likely lead to the most useful results.

Acknowledgments

We are grateful to Prof. Sushil Jajodia for the invitation to contribute to this volume, and for his interest in our work. This work was supported by the Air Force Research Laboratory (Rome, NY - USA) under contracts F30602-00-C-126 and F30602-01-C-057 to Scientific Systems Company and by Aprisma's University Fellowship 1999/2000. Scientific Systems Company acknowledges the continuing support from the Defensive Information Warfare Branch at the Air Force Research Laboratory in Rome, NY. We are particularly grateful to Mr. Peter J. Radesi and Dr. Leonard J. Popyack, Jr. from AFRL, for their encouragement.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993.

[2] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the Practice of Intrusion Detection Technologies. Technical Report CMU/SEI-99-TR-028, Carnegie Mellon University - Software Engineering Institute, January 2000.

[3] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Confer-*

ence on Computer and Communications Security, Singapore, November 1999.

[4] S. Axelsson. Intrusion Detection Systems: A Taxonomy and Survey. Technical Report 99-15, Department of Computer Engineering - Chalmers University of Technology, Sweden, March 2000.

[5] C. Bettini, S. Jajodia, and X. S. Wang. *Time Granularities in Databases, Data Mining and Temporal Reasoning.* Springer-Verlag, Berlin, 2000.

[6] J. B. D. Cabrera, L. J. Popyack, Jr., L. Lewis, B. Ravichandran, and R. K. Mehra. The Monitoring, Detection, Interpretation and Response Paradigm for the Security of Battlespace Networks. In *Proceedings of IEEE MILCOM 2001*, Washington, DC, October 2001.

[7] J. B. D. Cabrera, L. Lewis, and R. K. Mehra. Detection and Classification of Intrusions and Faults using Sequences of System Calls. In *ACM SIGMOD Record Special Issue on Data Mining for Intrusion Detection, Security and Threat Analysis*, December 2001.

[8] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, and R. K. Mehra. Proactive Intrusion Detection of Distributed Denial of Service Attacks - A Case Study in Security Management. *Journal of Network and Systems Management*, June 2002. In Press.

[9] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, R. K. Prasanth, B. Ravichandran, and R. K. Mehra. Proactive Detection of Distributed Denial of Service Attacks using MIB Traffic Variables - A Feasibility Study. In *Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management*, pages 609–622, Seattle, WA, May 2001.

[10] J. B. D. Cabrera and R. K. Mehra. Extracting Precursor Rules from Time Series - A Classical Statistical Viewpoint. In *Proceedings of the Second SIAM International Conference on Data Mining*, Arlington, VA, USA, April 2002. In Press.

[11] J. B. D. Cabrera, B. Ravichandran, and R. K. Mehra. Statistical Traffic Modeling for Network Intrusion Detection. In *Proceedings of the Eighth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 466–473, San Francisco, CA, August 2000. IEEE Computer Society.

[12] G. Casella and R. L. Berger. *Statistical Inference.* Duxbury Press, Belmont, CA, 1990.

[13] P. J. Criscuolo. Distributed Denial of Service - Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht. Technical Re-

port CIAC-2319, Department of Energy - CIAC (Computer Incident Advisory Capability), February 2000.

[14] G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[15] H. Debar, M. Dacier, and A. Wespi. Towards a Taxonomy of Intrusion-Detection Systems. *Computer Networks*, 31:805–822, 1999.

[16] D. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[17] M. Evans, N. Hastings, and B. Peacock. *Statistical Distributions.* John Wiley and Sons, Inc., New York, Second edition, 1993.

[18] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 34:424–438, 1969.

[19] J. Hamilton. *Time Series Analysis.* Princeton University Press, 1994.

[20] P. Helman and G. Liepins. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering*, 19(9):886–901, September 1993.

[21] H. S. Javitz and A. Valdes. The NIDES statistical component: Description and justification. Technical report, SRI International, March 1993.

[22] T. Kailath. *Linear Systems.* Prentice-Hall, Inc., 1980.

[23] S. Kent. On the trail of intrusions into information systems. *IEEE Spectrum*, pages 52–56, December 2000.

[24] C. Ko. Logic induction of valid behavior specifications for intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2000.

[25] W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems.* PhD thesis, Columbia University, June 1999.

[26] W. Lee, S. Stolfo, and K. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, 16(6):533–567, December 2000.

[27] W. Lee and S. J. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and Systems*, 3(4), November 2000.

[28] W. Lee, S. J. Stolfo, and P. K. Chan. Learning Patterns from Unix Process Execution Traces for Intrusion Detection. In *Proceedings of*

the AAAI Workshop on AI Methods in Fraud and Risk Management, pages 50–56, July 1997.

[29] Y. Li, N. Wu, X. S. Wang, and S. Jajodia. Enhancing Profiles for Anomaly Detection Using Time Granularities. *Journal of Computer Security*, 2002. In Press.

[30] L. Ljung. *System Identification - Theory for the User*. Prentice Hall, Second edition, 1999.

[31] T. Lunt. Automated audit trail analysis and intrusion detection: A survey. In *Proceedings of the 11th National Computer Security Conference*, pages 65–73, October 1988.

[32] E. Mach. On thought experiments. In *Knowledge and Error*. Dordrecht:Reidel, 1976. English translation of the 1905 German original.

[33] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[34] J. Markoff. U.S. drawing plan that will monitor computer systems - Looking for intruders. *The New York Times*, 1999. July 28, page A1.

[35] R. K. Mukkamala, J. Gagnon, and S. Jajodia. Integrating Data Mining Techniques with Intrusion Detection. In V. Atluri and J. Hale, editors, *Research Advances in Database and Information Systems Security*, pages 33–46. Kluwer Publishers, 2000.

[36] S. Northcutt. *Network Intrusion Detection - An Analyst's Handbook*. New Riders Publishing, 1999.

[37] A. Patel and S. O. Ciardhuain. The Impact of Forensics Computing on Telecommunications. *IEEE Communications Magazine*, pages 64–67, November 2000.

[38] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[39] F. B. Schneider, editor. *Trust in Cyberspace*. National Academy Press, 1998.

[40] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2000.

[41] E. G. Spafford. Reexamining Intrusion Detection, January 1999. Presentation at the University of Virginia, available at CERIAS homepage, Purdue Univ.

[42] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[43] R. Stovall. The Play from Pasadena (Super Bowl and the Stocks). *Financial World*, 156, 1987. Issue of January 28-February 10, 1987.

[44] J. D. Ullman. Data Mining Lecture Notes. Stanford University, Spring 2000.

[45] P. Uppuluri and R. Sekar. Experiences with Specification-based Intrusion Detection. In *Recent Advances in Intrusion Detection (RAID 2001)*. Springer-Verlag, Lecture Notes in Computer Science, Number 2212, 2001.

[46] A. Valdes and K. Skinner. Proabilistic Alert Correlation. In *Recent Advances in Intrusion Detection (RAID 2001)*. Springer-Verlag, Lecture Notes in Computer Science, Number 2212, 2001.

Chapter 9

E-MAIL AUTHORSHIP ATTRIBUTION FOR COMPUTER FORENSICS

Olivier de Vel
Communications Division
Defence Science and Technology Organisation
Department of Defence
P.O. Box 1500, Edinburgh SA5111, Australia
olivier.devel@dsto.defence.gov.au

Alison Anderson
School of Information Systems
Faculty of Information Technology
Queensland University of Technology
Brisbane Q4001, Australia
a.anderson@qut.edu.au

Mal Corney
School of Computer Science and Software Engineering
Faculty of Information Technology
Queensland University of Technology
Brisbane Q4001, Australia
m.corney@qut.edu.au

George Mohay
School of Computer Science and Software Engineering
Faculty of Information Technology
Queensland University of Technology
Brisbane Q4001, Australia
g.mohay@qut.edu.au

Abstract In this chapter, we briefly overview the relatively new discipline of computer forensics and describe an investigation of forensic authorship attribution or identification undertaken on a corpus of multi-author and multi-topic e-mail documents. We use an extended set of e-mail document features such as structural characteristics and linguistic patterns together with a Support Vector Machine as the learning algorithm. Experiments on a number of e-mail documents generated by different authors on a set of topics gave promising results for multi-topic and multi-author categorisation.

Keywords: Computer forensics, e-mail, support vector machines, author attribution, stylometrics.

1. Introduction and Motivation

With the rapid uptake of computer technology, many government organisations and private industries have become highly dependent on extensive computer networks for their operations, such as intra-nets and the Internet, to communicate and exchange information that have varying degrees of sensitivity: e.g., financial data and reports, intellectual property documents, personnel records, commercial-in-confidence reports, intelligence data and operational plans. These network environments offer many opportunities for improving the efficiency of corporate operations. At the same time, however, these environments are increasingly exposed to "attacks" such as unauthorised access from unauthorised users as well as from authorised users (insider misuse). Attacks may compromise both isolated and connected networks, increase the risks of unauthorised access to sensitive information, and potentially affect the continuity of information operations of the organisation.

1.1 Computer Forensics

With the increase in the number of unauthorised activities by authorised users (CSI Report, 2001)), the rise in the use of computers and computer networks for illegal activities (e.g., fraud, money laundering etc.) and, more recently, the growing impact of asymetric warfare on the information operations of critical infrastructure, the area of computer forensics has become increasingly important. Computer forensics has rapidly evolved over the past few years. The "oldest" form of computer forensics is digital evidence recovery which deals with data recovery methods from media such as hard disks. More recently, computer forensics has developed and branched out into several overlapping areas, generating a plethora of terms such as, *digital forensics*, *data forensics*, *network forensics*, *intrusion forensics*, *cyber-forensics*, *forensic*

data analysis etc. Furthermore, there are several different end-user areas where the term "computer forensics" is employed, with each area having different requirements. For example, in the traditional law enforcement area, the primary focus is the prosecution of the perpetrator. Once the crime has been perpetrated, the *post-mortem* collection and preservation of the chain of evidence custody, data analysis, interpretation etc. are undertaken subject to strict established prosecutorial guidelines. However, in some areas such as e-commerce where the continual availability of the on-line business service is of prime concern, the focus is on ensuring the continuity and survivability of computer networks. Therefore, the timeliness of the cycle of detection, forensic analysis and reaction is of critical importance in these application areas.

Computer Forensics can be (narrowly) defined as

> *an investigative activity employing a set of scientifically derived methods that attempt to anticipate, discover and reconstruct the sequence of events arising from some computer-based criminal activity.*

In a closer analogy with the field of forensic science, a broader definition of computer forensics would also include prosecutorial issues such as the collection and extraction of digital evidence, the preservation of the chain of evidence custody, admissibility and presentation of evidence and so on. Evidence may be obtained from many different digital and non-digital sources including storage devices (e.g., disks, discs), volatile data (e.g., RAM), network activity data (e.g., packet data, router tables), databases, logs, communications call data, behavioural profile data, social network data etc.

An increasingly important source of digital evidence is e-mail which has now become the dominant form of inter- and intra-organisational written communication for many companies and government organisations. E-mail is used in many legitimate activities such as message and document exchange. Unfortunately, it can also be misused for the distribution of unsolicited and/or inappropriate messages, images and documents. Examples of misuse include the distribution of unsolicited junk mail, unauthorised conveyancing of sensitive information, mailing of offensive or threatening material. E-mail evidence can be central in cases of sexual harassment or racial vilification, threats, bullying and so on. In some e-mail misuse cases the sender will attempt to hide his/her true identity in order to avoid detection. For example, the sender's address can be spoofed or anonymised by routing the e-mail through an anony-

mous mail server, or the e-mail's contents and header information may have been modified in an attempt to hide the true identity of the sender. In other cases the sender may wish to masquerade as another user.

1.2 E-mail Forensics

E-mail forensics deals with one or more e-mails as evidence in a forensic investigation. The sources of e-mail evidence are wide and varied and can include both data and meta-data. Data include the e-mail body, e-mail headers (sender, receiver, date, time etc.), e-mail attachments, etc. whereas e-mail meta-data include other related evidence such as e-mail trace routes, e-mail file time stamps, e-mailer application used etc.. Both data and meta-data can, and should, be used in a computer forensic analysis as they all could be contributing factors in a forensic investigation and the consequent successful prosecution of an offending user. A forensic investigation will attempt to determine one or more issues. For example, the investigation may wish to determine the author(s) of an e-mail, or the timeline of multiple e-mails, or the type of author(s) and so on. In this paper we focus uniquely on the authorship attribution of an e-mail forensic investigation and uniquely on the analysis of the contents of the e-mail body.

The process of attributing e-mail authorship has a few important characteristics. Firstly, the identification of an author is usually attempted from a small set of known candidates, rather than from a large set of potentially unknown authors. That is, e-mail authorship attribution in computer forensics is essentially a classification problem using a known population distribution. Secondly, as mentioned above, the text body of the e-mail is not the only source of authorship attribution in a forensic investigation. Finally, the number of false negatives should be minimised (at the expense of a higher number of false positives), to avoid the problem of the offending author "slipping through the net".

As a result of this growing e-mail misuse problem, efficient automated methods for analysing the content of e-mail messages and identifying or categorising the authors of these messages are becoming imperative. The principal objectives are to classify an ensemble of e-mails as belonging to a particular author and, if possible, obtain a set of characteristics or features that remain relatively constant for a large number of e-mails written by the author. The question then arises; can characteristics such as language, structure, layout etc. of an e-mail be used, with a high degree of confidence, as a kind of author phrenology and thus link the e-mail document with its author? Also, can we expect the writing characteristics or style of an author to evolve in time and change in different

contexts? For example, the composition of formal e-mails will differ from informal ones (changes in vocabulary etc.). Even in the context of informal e-mails there could be several composition styles (e.g., one style for personal relations and one for work relations). However, humans are creatures of habit and have certain personal traits which tend to persist. All humans have unique (or near-unique) patterns of behaviour, biometric attributes, and so on. We therefore conjecture that certain characteristics pertaining to language, composition and writing, such as particular syntactic and structural layout traits, patterns of vocabulary usage, unusual language usage (e.g., converting the letter "f" to "ph", or the excessive use of digits and/or upper-case letters), stylistic and sub-stylistic features will remain relatively constant. The identification and learning of these characteristics with a sufficiently high accuracy are the principal challenges in authorship categorisation.

Authorship categorisation or attribution can be effected using various approaches. Firstly, the simplest method is to use domain experts to identify new e-mail documents and allocate them to well-defined author categories. This can be time-consuming and expensive and, perhaps most limiting, provides no continuous measure of the degree of confidence with which the allocation was made. Secondly, the domain expert can establish a set of fixed rules which can be used to classify new e-mail documents. Unfortunately, in many cases, the rule-set can be large and unwieldy, typically difficult to update, and unable to adapt to changes in document content or author characteristics. Finally, categorisation can be undertaken automatically by inductively learning the classifiers from training example documents. This approach should, hopefully, generalise well to new, unseen e-mail documents and has the advantage that it should be able to adapt to a measure of drift in the characteristics of authors and create a more accurate profile of each author.

A closely related, but clearly separate, area of authorship categorisation is text categorisation, which attempts to categorise a set of text documents based on its contents or topic. Text categorisation provides support for a wide variety of activities in information mining and information management. It has found applications in areas such as document filtering, and can be used to support document retrieval by generating the categories required in document retrieval. Many methods that automatically learn rules have been proposed for text categorisation. Most of these techniques employ the "bag–of–words" or word vector space feature representation (Salton, 1983) where each word in the text document corresponds to a single feature. A learning algorithm such as decision trees (Apte, 1998), neural networks (Ng, 1997), Bayesian probabilistic approaches (Mitchell, 1997)(Yang, 1999), or support vec-

tor machines (Joachims, 1998) is then used to classify the text document. de Vel (de Vel, 1999) studied the comparative performance of text document categorisation algorithms using the Naive Bayes, Support Vector Machines, multi-layer Perceptron and k–NN classifiers. Work in e-mail text classification has also been undertaken by some researchers in the context of automated e-mail document filtering and filing. Cohen (Cohen, 1996) learned rule sets based on a small number of keywords in the e-mail. Sahami *et al* (Sahami, 1998) focused on the more specific problem of filtering junk e-mail using a Naive Bayesian classifier and incorporating domain knowledge using manually constructed domain-specific attributes such as phrasal features and various non-textual features.

In this paper we investigate methods for the multi-topic machine learning of an authorship attribution classifier using e-mail documents as the data set. We focus on the problem of authorship attribution of e-mails and not e-mail document categorisation, i.e. not the classification of e-mail messages for topic categorisation etc. We incorporate various document features such as structural characteristics and linguistic evidence in the learning algorithm. We study the effect of multiple e-mail topics on the discrimination performance of authorship attribution. For example, can an author be identified in the context of different e-mail topics? That is, we wish to investigate the degree of orthogonality existing between e-mail authorship and e-mail topic content. We first introduce the field of authorship categorisation in Section 2 and, more specifically, e-mail authorship categorisation in Section 3. We then briefly outline the Support Vector Machines learning algorithm in Section 4 and present the database of e-mail documents used in the experiments together with the experimental methodology in Section 5, respectively. Validation of the method is then undertaken by presenting results of categorisation performance in Section 6. Finally, we conclude with some general observations and present future directions for the work in Section 7.

2. Authorship Categorisation

Formally, authorship attribution or categorisation is the task of determining the author of a piece of work. In particular, we are interested in categorising textual work given other text samples produced by the same author. We assume that only one author is responsible for producing the text – contributions by, or text modified by, multiple authors are not considered here (though, as we describe later on, e-mails with labeled text from other authors are included in our analysis).

Authorship categorisation is a subset of the more general problem called "authorship analysis" (Gray, 1997). Authorship analysis includes other distinct fields such as *author characterisation* and *similarity detection*. Authorship characterisation determines the author profile or characteristics of the author that produced a piece of work. Example characteristics include gender, educational and cultural backgrounds, language familiarity etc. (Thomson, 2001). Similarity detection calculates the degree of similarity between two or more pieces of work without necessarily identifying the authors. Similarity is used extensively in the context of plagiarism detection which involves the complete or partial replication of a piece of work with or without permission of the original author. We note, however, that authorship categorisation and author characterisation are different from plagiarism detection. Plagiarism detection attempts to detect the similarity between two substantially different pieces of work but is unable to determine if they were produced by the same author.

Authorship analysis has been used in a small but diverse number of application areas. Examples include identifying authors in literature, in program code, and in forensic analysis for criminal cases. We briefly outline the work undertaken in each one of these areas.

Perhaps the most extensive and comprehensive application of authorship analysis is in literature and in published articles. Well-known authorship analysis studies include the disputed Federalist papers (For example, (Mosteller, 1964) and (Bosch, 1998)) and Shakespeare's works, the latter dating back over many years (see, for example, where attempts were made to show that Shakespeare was a hoax and that the real author was Edward de Vere, the Earl of Oxford (Elliot, 1991)). In these studies, specific author features such as unusual diction, frequency of certain words, choice of rhymes, and habits of hyphenation have been used as tests for author attribution. These authorial features are examples of *stylistic evidence* which is thought to be useful in establishing the authorship of a text document. It is conjectured that a given author's style is comprised of a number of distinctive features or attributes sufficient to uniquely identify the author. Stylometric features ("style markers") used in early authorship attribution studies were character or word based, such as vocabulary richness metrics (e.g., Zipf's word frequency distribution and its variants), word length etc.. However, some of these stylometric features could be generated under the conscious control of the author and, consequently, may be content-dependent and are a function of the document topic, genre, epoch etc.. Rather than using content-dependent features, we employ features derived from words and/or syntactic patterns since such features are more

likely to be content-independent and thus potentially more useful in discriminating authors in different contexts. It is thought that syntactic structure is generated dynamically and sub-consciously when language is created, similar to the case of the generation of utterances during speech composition and production (Crain, 1998). That is, language patterns or syntactic features are generated beyond an author's conscious control. An example of such features is short, all-purpose words (referred to as *function words*) such as "the", "if", "to" etc. whose frequency or relative frequency of usage is unaffected by the subject matter. Another example syntactic feature is punctuation which is thought to be the graphical correlate of intonation which is the phonetic correlate of syntactic structure (Chaski, 1998). As punctuation is not guided by any strict placement rules (e.g., comment placement), punctuation will vary from author to author. Chaski (Chaski, 2001) has shown that punctuation can be useful in discriminating authors. Therefore, a combination of syntactic features may be sufficient to uniquely identify an author. According to Rudman, over 1,000 stylometric features have been proposed (Rudman, 1997). Tweedie *et al* also list a variety of different stylometric features (Tweedie, 1998). However, no set of significant style markers have been identified as uniquely discriminatory. Furthermore, some proposed features may not be valid discriminators as, for example, prescriptive grammar errors, profanities etc. which are not generally considered to be idiosyncratic. Just as there is a range of available stylometric features, there are many different techniques using these features for author identification. These include statistical approaches (e.g., cusum (Farringdon, 1996), Thisted and Efron test (Thisted, 1987)), neural networks (e.g., radial basis functions (Lowe, 1995), feedforward neural networks (Tweedie, 1996), cascade correlation (Waugh, 2000)), genetic algorithms (e.g., (Holmes, 1995)), Markov chains (e.g., (Khmelev, 2000)). However, there does not seem to exist a consensus on a correct methodology, with many of these techniques suffering from problems such as questionable analysis, inconsistencies for the same set of authors, failed replication etc.

Program code authorship has been researched by some workers in the context of software theft and plagiarism, software author tracking and intrusion detection. For example, software author tracking enables the identification of the author of a particular code fragment from a large set of programmers working on a software project. This can be useful for identifying authors for the purpose of effecting upgrades to software and software maintenance. The authorship of a computer virus or trojan horse can be identified in a similar manner (Spafford, 1993). By examining peculiar characteristics or metrics of programming style it

is possible to identify the author of a section of program code (Oman, 1989), in a similar way that linguistic evidence can be used for categorising the authors of free text. Program metrics such as typographical characteristics (e.g., use of lower and upper case characters, multiplicity of program statements per line, etc.), stylistic metrics (e.g., length of variable names, preference for `while` or `for` loops, etc.), programming structure metrics (e.g., placement of comments, use of debugging symbols, etc.) have been employed (Krsul, 1997)(Krsul, 1994)(Sallis, 1997).

The forensic analysis of text attempts to match text to authors for the purpose of a criminal investigation. The forensic analysis of text generally includes techniques derived from linguistics or behavioural profiling. Linguistic techniques usually employ common knowledge features such as grammatical errors, spelling, and stylistic deviations. These techniques, contrary to popular belief, do not quantify linguistic patterns and fail to discriminate between authors with a high degree of precision. However, the use of language-based author attribution testimony as admissible evidence in legal proceedings has been identified in many cases (Chaski, 1998). The textual analysis of the Unabomber manifesto is a well-known example of the use of forensic linguistics. In this case, the manifesto and the suspect bomber used a set of similar characteristics, such as a distinctive vocabulary, irregular hyphenations etc. (Crain, 1998)(Foster, 2000). Techniques based on scientific evidence of language have not, to the authors' knowledge, been used in court proceedings. Profiling is based on the behavioural characteristics contained within an author's text. For example, educated guesses on the type of personality of an author based on particular sequences of words are employed in profiling studies.

E-mail documents have several characteristics which make authorship categorisation challenging compared with longer, formal text documents such as literary works or published articles (such as the Federalist Papers). Firstly, e-mails are generally short in length indicating that certain language-based metrics may not be appropriate (e.g., vocabulary richness). Secondly, the composition style used in formulating an e-mail document is often different from normal text documents written by the same author. That is, an author profile derived from normal text documents (e.g., publications) may not necessarily be the same as that obtained from an e-mail document. For example, e-mail documents are generally brief and to the point, can involve a dialogue between two or more authors, can be punctuated with a larger number of grammatical errors etc. Also, e-mail interaction between authors can be frequent and rapid, similar to speech interactivity and rather dissimilar to normal text document interchange patterns. Indeed, the authoring composi-

tion style and interactivity characteristics attributed to e-mails shares some elements of both formal writing and speech. Thirdly, the author's composition style used in e-mails can vary depending upon the intended recipient and can evolve quite rapidly over time. Fourthly, the vocabulary used by authors in e-mails is not stable, facilitating imitation. Thus the possibility of being able to disguise authorship of an e-mail through imitation is potentially high. Furthermore, similar vocabulary subsets (e.g., technology-based words) may be used within author communities. Finally, e-mail documents have generally few sentences/paragraphs, thus making contents profiling based on traditional text document analysis techniques, such as the "bag–of–words" representation (e.g., when using the Naive Bayes approach), more difficult. However, as stated previously, certain characteristics such as particular syntactic and structural layout traits, patterns of vocabulary usage, unusual language usage, stylistic and sub-stylistic features will remain relatively constant for a given e-mail author. This provides the major motivation for the particular choice of attributes/features for the authorship categorisation of e-mails, as we shall discuss in Section 5.

3. E-mail Authorship Attribution

To date few studies in e-mail authorship attribution have been undertaken. An initial study was undertaken by de Vel (de Vel, 2000) where e-mail authorship categorisation was investigated using a basic subset of structural and stylometric features on a set of authors without consideration of the author characteristics (gender, language, etc.) nor of the e-mail topic and size. Anderson *et al* (Anderson, 2001) extended this study and used a larger set of stylometric features and also studied the effect of a number of parameters such as, the type of feature sets, text size, and the number of documents per author, on the author categorisation performance for both e-mails and text documents. Some feature types such as N-graphs (where $N = 2$ was used) gave good categorisation results for different text chunk sizes but these results were thought to be due to an inherent bias of some types of N-graphs towards content rather than style alone (N-graphs are contiguous sequences of characters, including whitespaces, punctuation etc. . .). They observed almost no effect of the text chunk size on the categorisation performance, for text chunks larger than approximately 100 words. Also, they observed that as few as 20 documents may be sufficient for satisfactory categorisation performance. These results are significant in the context of e-mail authorship categorisation and computer forensics as they indicate that satisfactory results can still be achieved with a small text size and a small

number of available e-mails. Although Anderson *et al* concluded that it is possible to categorise e-mail authors based on a small number of e-mails and small text sizes, they did not consider other author attribution characteristics such as multi-topic categorisation performance, nor author characteristics. More recently, in the context of e-mail authorship characterisation, Thomson *et al* have investigated the existence of gender-preferential language styles in e-mail communication (Thomson, 2001). The types of styles investigated included references to emotion, provision of personal information, use of intensive adverbs, the frequency of insults and opinions (it was hypothesised that the first three of these features are characteristic of female authors whereas the last set of features are male-preferential). Using manual feature extraction and discriminant analysis, Thomson *et al* claimed that they were able to predict the gender of e-mail authors.

In this paper we extend the results of these investigations and study the author attribution performance in the context of multiple e-mail topic categories. We investigate the e-mail document feature types that enable us to discriminate between e-mail authors independent of e-mail topic. We use a publicly-derived e-mail corpus for our evaluation experiments.

4. Support Vector Machine Classifier

The fundamental concepts of Support Vector Machines (SVM) were developed by Vapnik (Vapnik, 1995). The SVMs' concept is based on the idea of structural risk minimisation which minimises the generalisation error (i.e. true error on unseen examples) which is bounded by the sum of the training set error and a term which depends on the Vapnik-Chervonenkis (VC) dimension of the classifier and on the number of training examples. The use of a structural risk minimisation performance measure is in contrast with the empirical risk minimisation approach used by conventional classifiers. Conventional classifiers attempt to minimise the training set error which does not necessarily achieve a minimum generalisation error. Therefore, SVMs have theoretically a greater ability to generalise. For further reading, see (Vapnik, 1995).

SVMs effectively enlarge the input feature space, using basis functions such as polynomials, where better class separation can be achieved. In some cases, the transformed feature space can be very large. SVMs belong effect to the class of the more general basis expansion and regularisation problem to which methods such as smoothing splines, multidimensional splines (eg, MARS (Friedman, 1991)) and wavelet smoothing belong. Unlike many other learning algorithms, such as linear discrimi-

nant analysis, the number of free parameters used in the SVM depends on the margin that separates the data and does not depend on the number of input features. Thus the SVM does not require a reduction in the number of features in order to avoid the problem of over-fitting (see, however, Section 6). This property is clearly an advantage in the context of high-dimensional applications, such as text document and authorship categorisation, as long as the data vectors are separable with a wide margin. Unfortunately, SVMs require the implementation of optimisation algorithms for the minimisation procedure which can be computationally expensive. A few researchers have applied SVMs to the problem of text document categorisation using approximately 10,000 features in some cases, concluding that, in most cases, SVMs outperform conventional classifiers (Yang, 1999)(Joachims, 1998). Drucker *et al* used SVMs for classifying e-mail text as spam or non-spam and compared it to boosting decision trees, Ripper and Rocchio classification algorithms (Druker, 1999). Bosch *et al* used a separating hyperplane based on a similar idea to that of a linearly separable SVM for determining the authorship of two authors of the formal articles published within the set of the Federalist Papers (Bosch, 1998). Teytaud *et al* investigated different SVM kernels for author identification (principally well-known French authors) and language discrimination using N-graphs as the relevant features (Teytaud, 2001). Diederich *et al* evaluated the performance of SVMs with various features such as term frequencies, as well as structural features such as tagword bi-grams using the German Berliner Zeitung newspaper corpus (Diederich, 2000). Multi-topic author attribution experiments were also undertaken by Diederich *et al*. They obtained poor recall performance results when using function word bi-grams, in apparent disagreement with the assumption that function words minimise content information.

5. E-mail Corpus and Methodology

The availability of the e-mail corpus was severely constrained by privacy issues and ethical considerations. Publicly available e-mail corpuses include newsgroups, mailing lists etc. However, in such public e-mail databases, it is generally quite difficult to obtain a sufficiently large and "clean" (i.e., void of cross-postings, off-the-topic spam, empty bodied e-mails with attachements etc.) corpus of e-mails. This is particularly true when attempting to obtain an e-mail corpus matrix sufficiently populated with multiple authors and multiple topics with no (or, at worst, minimally) intersecting classes. For example, there should be minimal content overlap between different topics.

Table 9.1. Summary statistics of the e-mail newsgroup and author corpus used in the experiment.

Newsgroup Category	*Author Category* AC_i $(i = 1, \ldots, 4)$				*Newsgroup Total*
	damnfine AC_1	*galbraith* AC_2	*gensch* AC_3	*zaphy* AC_4	
`aus.tv`	175	87	89	64	415
`aus.tv.buffy`	233	123	0	13	369
`aus.film`	80	14	45	0	139
`aus.dvd`	113	0	144	35	292
Author Total	601	224	278	112	1215

The corpus of e-mail documents used in the experimental evaluation of authorship categorisation contained a total of 1215 documents sourced from four public newsgroups and four native language (English) male authors. Any cross-postings and off-the-topic e-mails were purged from the corpus. The body of each e-mail document was parsed, based on an e-mail grammar, and the relevant e-mail body features were extracted. The body was pre-processed to remove (if present) any salutations, reply text and signatures. However, the existence, position within the e-mail body and type of some of these are retained as inputs to the categoriser (see below). Attachments are excluded, though the e-mail body itself is used. A summary of the global e-mail document corpus statistics is shown in Table 9.1.

A number of attributes/features identified in baseline authorship attribution experiments undertaken on constrained topics (see (Anderson, 2001) and (de Vel, 2000)) as most useful for e-mail authorship discrimination were extracted from each e-mail body document. These attributes included both style markers as well as structural features. A total of 191 attributes, comprising 170 style marker attributes and 21 structural attributes, were employed in the experiment. These are listed in Tables 9.2 and 9.3, respectively. Note that M = total number of *tokens* (i.e., words), V = total number of *types* (i.e., distinct words), C = total number of characters, and H = total number of HTML tags in the e-mail body. Also, the hapax legomena count is defined as the number of types that occur only once in the e-mail text.

We briefly clarify how we derive some of the attributes shown in Table 9.2. Firstly, the set of short words in each e-mail document consists of all words of length less or equal to 3 characters (e.g., "all", "at", "his"

Table 9.2. E-mail document body style marker attributes. Total of 170 features are used in the experiment. See text for clarification.

Attribute Type, A_i $(i = 0, \ldots, 169)$	
A_0:	Number of blank lines/total number of lines
A_1:	Average sentence length
A_2:	Average word length (number of characters)
A_3:	Vocabulary richness i.e., V/M
A_4:	Number of function words/M
A_5 to A_{126}:	Function word frequency distribution (122 features)
A_{127}:	Number of short words/M
A_{128}:	Count of hapax legomena/M
A_{129}:	Count of hapax legomena/V
A_{130}:	Number of characters in words/C
A_{131}:	Number of alphabetic characters in words/C
A_{132}:	Number of upper-case characters in words/C
A_{133}:	Number of digit characters in words/C
A_{134}:	Number of white-space characters/C
A_{135}:	Number of spaces/C
A_{136}:	Number of spaces/Number white-space characters
A_{137}:	Number of tab spaces/C
A_{138}:	Number of tab spaces/Number white-space characters
A_{139}:	Number of punctuation characters/C
A_{140} to A_{169}:	Word length frequency distribution/M (30 features)

etc.). Only the count of short words is used as a feature. The short word frequency distribution may be biased towards e-mail content and was therefore not used in our experiments. Secondly, the set of all-purpose function words ("a", "all", "also", ..., "to", "with") and its frequency distribution is obtained and also used as a sub-vector attribute. The number of function words used is 122. Finally, a word length frequency distribution consisting of 30 features (up to a maximum word length of 30 characters) is employed.

Though our choice of attributes is specifically biased towards features that have been shown to be able to effectively discriminate between authors, rather than discriminating between topics, some of the style marker attributes may have a combination of author and content bias as, for example, hapax legomena (Chaski, 1998).

The requoted text position refers to the reply status of e-mail. A reply text can generally be placed in any position in the e-mail document and each line is usually prefixed with a special character (e.g., ">"). In our experiment, the position of requoted text allowed for 6 different possibilities (e-mail body text interspersed with the requoted text, e-

Table 9.3. E-mail document body structural attributes. Total of 21 attributes/features are used in the experiment. See text for clarification.

Attribute Type, A_i *(*$i = 170, \ldots, 190$*)*	
A_{170}:	`Has a greeting acknowledgement`
A_{171}:	`Uses a farewell acknowledgement`
A_{172}:	`Contains signature text`
A_{173}:	`Number of attachments`
A_{174}:	`Position of requoted text within e-mail body`
A_{175} to A_{190}:	`HTML tag frequency distribution/`H (16 features)

mail body text preceeded by requoted text etc.). Due to some e-mailers using HTML formatting, we include the set of HTML tags as a structural metric. The frequency distribution of HTML tags was included as one of the 21 structural attributes.

To ensure all attributes are treated equally in the classification process, each attribute A_i is scaled as follows:

$$A_i^{(\text{scaled})} = (A_i - A_{i,\min})SF_{A_i} + LB_{A_i}$$

where the scaling factor is calculated as:

$$SF_{A_i} = \frac{UB_{A_i} - LB_{A_i}}{A_{i,\max} - A_{i,\min}}$$

with $A_{i,\min}$ and $A_{i,\max}$ being the minimum and maximum values of the attribute A_i, respectively. Also, LB_{A_i} and UB_{A_i} are the defined lower and upper bounds of the scaled attribute, respectively (we have used $LB_{A_i} = 0.0$ and $UB_{A_i} = 1.0$).

This produces a scaling factor and threshold value for each feature and is applied to the training data and then saved to a file so that they can be used to scale any training data. In the Analyser package this is all done by the PrepInput program. Experiments can be run as k-fold cross validation tests where the scaling factors and thresholds are calculated and used internally in the production of the scaled data or the experiments can be run where one data set is used for training and another data set can be used for testing. In the latter case, the scale factors and thresholds have to be saved to a file and that file used in production of the properly scaled test data.

The classifier used in the experiments was the Support Vector Machines classifier, SVM^{light}, developed by T. Joachims from the University of Dortmund (SVMLight, 2001). SVM^{light} is an implementation of Vapnik's Support Vector Machines. It scales well to a large number of

sparse instance vectors as well as efficiently handling a large number of support vectors. In our experiments we explored a number of different kernel functions for the SVM classifier namely, the linear, polynomial, radial basis and sigmoid *tanh* functions. We obtained maximal F_1 classification results (see below for the definition of F_1) on our data set with a polynomial kernel of degree 3. The "LOQO" optimiser was used for maximising the margin.

As Support Vector Machines only compute two-way categorisation, Q two-way classification models were generated, where Q is the number of author categories ($Q = 4$ for our e-mail document corpus), and each SVM categorisation was applied Q times. This produced Q two-way confusion matrices. The SVM classifier was trained on the `aus.tv` e-mail document set (415 documents) and tested on the remaining (unseen) 3 newsgroup topic sets.

To evaluate the categorisation performance on the e-mail document corpus, we calculate the accuracy, recall (R), precision (P) and combined F_1 performance measures commonly employed in the information retrieval and text categorisation literature (for a discussion of these measures see, for example, (Witten, 2000)), where:

$$F_1 = \frac{2RP}{(R+P)}$$

We note in passing that the recall value is a more indicative performance metric than precision for authorship attribution in computer forensics, as it effectively measures the impact of false negatives.

To obtain an overall performance figure over all binary categorisation tasks, a macro-averaged F_1 statistic is calculated (Yang, 1999). Here, N_{AC} per-author-category confusion matrices (where N_{AC} is the total number of author categories, $N_{AC} = 4$ in our experiment) are computed and then averaged over all categories to produce the macro-averaged statistic, $F_1^{(M)}$:

$$F_1^{(M)} = \frac{\sum_{i=1}^{N_{AC}} F_{1,AC_i}}{N_{AC}}$$

where F_{1,AC_i} is the per-author-category F_1 statistic for author category AC_i ($i = 1, 2, , \ldots N_{AC}$):

$$F_{1,AC_i} = \frac{2R_{AC_i}P_{AC_i}}{(R_{AC_i} + P_{AC_i})}$$

Table 9.4. Per-author-category P_{AC_i}, R_{AC_i} and F_{1,AC_i} categorisation performance results (in %) for the four different author categories ($i = 1, \ldots, 4$). The newsgroup `aus.tv` is used as the training set (see text).[a]

Newsgroup Category	*Author Category, AC_i ($i = 1, \ldots, 4$)*					
	damnfine, AC_1			*galbraith, AC_2*		
	P_{AC_1}	R_{AC_1}	F_{1,AC_1}	P_{AC_2}	R_{AC_2}	F_{1,AC_2}
`aus.tv.buffy`	98.6	93.6	96.0	86.2	91.1	88.6
`aus.film`	96.4	100.0	98.2	65.0	92.9	76.5
`aus.dvd`	95.5	94.7	95.1	-	-	-

[a] The symbol "-" in a matrix cell indicates a small number or non-existent e-mail data.

Table 9.4 (continued)

Newsgroup Category	*Author Category, AC_i ($i = 1, \ldots, 4$)*					
	gensch, AC_3			*zaphy, AC_4*		
	P_{AC_3}	R_{AC_3}	F_{1,AC_3}	P_{AC_4}	R_{AC_4}	F_{1,AC_4}
`aus.tv.buffy`	-	-	-	40.9	69.2	51.4
`aus.film`	95.7	97.8	96.7	-	-	-
`aus.dvd`	100.0	93.1	96.4	100.0	71.4	83.3

6. Results and Discussion

We report our results presenting the per-author-category F_1 statistic for the Support Vector Machines (SVM) classifier. The results are displayed as a newsgroup-author performance matrix as shown in Table 9.4.

As observed in Table 9.4, results indicate that, in general, the SVM classifier combined with the style markers and structural attributes is able to discriminate between the authors for the different newsgroup categories. Discrimination of the authors for the individual `aus.film` and `aus.dvd` newsgroups is consistently high. One exception in these newsgroup category results is the low performance obtained with author "*zaphy*" in the `aus.tv.buffy` newsgroup. In particular, a poor precision but satisfactory recall performance is observed though, as noted previously, this trade-off is perhaps not such a negative result in computer forensics since a higher recall value at the expense of a lower precision value is preferred to a low recall-high precision value scenario. This re-

duced F_{1,AC_4} performance value may be due to the similarity between the training newsgroup set (`aus.tv`) and the test newsgroup, with some of the attributes having a content-based bias, such as hapax legomena, possibly biasing the categorisation towards the e-mail document topic content rather than on its author. Classification performance for authors "*damnfine*" and "*gensch*" is consistent and satisfactory across all of the newsgroups. The performances for "*galbraith*" and "*zaphy*" are not as convincing and further experimentation to understand the discrepancy is required.

We also investigated the categorisation performance as a function of word collocation and the type and dimensionality of the function word vector attributes. The number of function words was increased to 320 (from 122) and the set of these were split into two categories, namely parts-of-speech (POS) words and others. It was observed that word collocation, increased function word distribution dimensionality, and using POS function words did not improve the author categorisation performance across the different newsgroups. In particular, the categorisation performance worsened with increasing function word distribution size, which seems to be at odds with the belief that SVMs are robust in high dimensional settings (see also Hastie *et al* who show that the test error of an SVM can significantly increase in the presence of additional noisy independent features compared with additive and adaptive spline models (Hastie, 2001)).

7. Conclusions

We have investigated the learning of authorship categories from multi-topic e-mail documents. We used an extended set of predominantly content-free e-mail document features such as structural characteristics and linguistic patterns together with a Support Vector Machine learning algorithm. Experiments on a number of e-mail documents generated by different authors on a set of topics gave promising results for multi-topic and multi-author categorisation, though some author categories produced better categorisation performance results than other categories. We also observed a reduction in classification performance with increasing function word dimensionality and no improvement with word collocation.

There are several limitations with the current approach. Firstly, the fact that some authors have a better categorisation performance than other authors indicates that more identifiable author traits need to be obtained. For example, we are investigating function word vector subset selection in an attempt to identify the most useful function words for

a given set of authors. Secondly, we wish to investigate a denser and more orthogonal e-mail author-topic matrix for the e-mail corpus with minimally intersecting classes to minimise any bias (eg, content bias) in the authorship categorisation results. Some progress in this direction has been achieved (de Vel, 2001). Thirdly, more studies on the usefulness of specific N-graphs for author identification should be investigated as it is conjectured that, for example, certain bi-graphs incorporating punctuation are effective author discriminators (Chaski, 2001). Finally, the number of author categories considered in our experiments at the moment is quite small. Though it is not easy to obtain a sufficiently large set of e-mails from multiple authors and multiple topics, we are currently attempting to build up a suitable forensic database and test our approach.

References

Computer Security Institute (2001). "2001 CSI/FBI Computer Crime and Security Survey", Computer Security Issues & Trends.

Salton G., and McGill M. (1983). *Introduction to Modern Information Filtering,* McGraw-Hill, New York.

Apte C., Damerau F., and Weiss S. (1998). "Text mining with decision rules and decision trees", Workshop on Learning from text and the Web, Conference on Automated Learning and Discovery.

Ng H., Goh W., and Low K. (1997). "Feature selection, perceptron learning, and a usability case study for text categorization", Proc. 20th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR97), pp.67–73.

Yang Y., and Liu X. (1999). "A re-examination of text categorisation methods", Proc. 22nd Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR99), pp.67–73.

Joachims T. (1998). "Text categorization with support vector machines: Learning with many relevant features", Proc. European Conf. Machine Learning (ECML'98), pp.137–142.

de Vel O. (1999). "Evaluation of Text Document Categorisation Techniques for Computer Forensics", Journal of Computer Security, (submitted).

Cohen W. (1996). "Learning rules that classify e-mail", Proc. Machine Learning in Information Access: AAAI Spring Symposium (SS-96-05), pp.18–25.

Sahami M., Dumais S., Heckerman D., and Horvitz E., "A Bayesian approach to filtering junk e-mail", Learning for Text Categorization Workshop: 15th National Conf. on AI. AAAI Technical Report WS-98-05, pp.55–62.

Mitchell T. (1997). *Machine Learning,* McGraw-Hill, New York.

Gray A., Sallis P., and MacDonell S. (1997). "Software forensics: Extending authorship analysis techniques to computer programs", Proc. 3rd Biannual Conf. Int. Assoc. of Forensic Linguists (IAFL'97), pp.1–8.

Thomson R., and Murachver T. (2001). "Predicting gender from electronic discourse", British Journal of Social Psychology, pp.193–208.

Mosteller F., and Wallace D. (1964). *Inference and Disputed Authorship: The Federalist,* Addison-Wesley, Reading, Mass.

Bosch R., and Smith J. (1998). "Separating hyperplanes and the authorship of the disputed federalist papers", American Mathematical Monthly, **105**, pp.601–608.

Elliot W., and Valenza R. (1991). "Was the Earl of Oxford the true Shakespeare?", Notes and Queries, **38**, pp.501–506.

Crain C. (1998). "The Bard's fingerprints", Lingua Franca, pp.29–39.

Chaski C. (1998). "A Daubert-inspired assessment of current techniques for language-based author identification", US National Institute of Justice, available through *www.ncjrs.org*.

Chaski C. (2001). "Empirical evaluations of language-based author identification techniques", Forensic Linguistics, to appear.

Rudman J. (1997). "The state of authorship attribution studies: Some problems and solutions", Computers and the Humanities, **31**, pp.351–365.

Tweedie F., and Baayen R. (1998). "How variable may a constant be? Measure of lexical richness in perspective", Computers and the Humanities, **32**, pp.323–352.

Farringdon J. (1996). *Analysing for Authorship: A Guide to the Cusum Technique,* University of Wales Press, Cardiff.

Thisted B., and Efron R. (1987). "Did Shakespeare write a newly discovered poem?", Biometrika, pp.445–455.

Lowe D., and Matthews R. (1995). "Shakespeare vs Fletcher: A stylometric analysis by radial basis functions", Computers and the Humanities, pp.449–461.

Tweedie F., Singh S., and Holmes D. (1996). "Neural network applications in stylometry: The Federalist papers", Computers and the Humanities, **30**, pp.1–10.

Waugh S., Adams A., and Tweedie F. (2000). "Computational stylistics using artificial neural networks", Literary and Linguistic Computing, **15**, pp.187–198.

Holmes D., Forsyth R. (1995). "The Federalist revisited: New directions in authorship attribution", Literary and Linguistic Computing, pp.111–127.

Khmelev D. (2000). "Disputed authorship resolution using relative entropy for Markov chain of letters in a text", Proc. 4th Conference Int. Quantitative Linguistics Association, R. Baayen (Ed.), Prague.

Spafford E., and Weeber S. (1993). "Software forensics: tracking code to its authors", Computers and Security, **12**, pp.585–595.

Oman P., and Cook C. (1989). "Programming style authorship analysis", Proc. 17th Annual ACM Computer Science Conference, pp.320–326.

Krsul I., and Spafford E. (1997). "Authorship analysis: Identifying the author of a program", Computers and Security, **16**, p..248–259.

Krsul I. (1994). "Authorship analysis: Identifying the author of a program", Technical Report CSD-TR-94-030, Department of Computer Science, Purdue University.

Sallis P., MacDonell S., MacLennan G., Gray A., and Kilgour R. (1997). "Identified: Software authorship analysis with case-based reasoning", Proc. Addendum Session Int. Conf. Neural Info. Processing and Intelligent Info. Systems, pp.53–56.

Foster D. (2000). *Author Unknown: On the Trail of Anonymous,* Henry Holt, New York.

de Vel O. (2000). "Mining e-mail authorship", Proc. Workshop on Text Mining, ACM International Conference on Knowledge Discovery and Data Mining (KDD'2000), Boston.

Anderson A., Corney M., de Vel O., and Mohay G. (2001). "Identifying the Authors of Suspect E-mail", Computers and Security, (submitted).

Vapnik V. (1995). *The Nature of Statistical Learning Theory,* Springer-Verlag, New York.

Druker H., Wu D., and Vapnik V. (1999). "Support vector machines for spam categorisation", IEEE Trans. on Neural Networks, **10**, pp.1048–1054.

Teytaud O., and Jalam R. (2001). "Kernel-based text categorization", International Joint Conference on Neural Networks (IJCNN'2001), Washington DC.

Diederich J., Kindermann J., Leopold E., and Paass G. (2000). "Authorship attribution with Support Vector Machines", Applied Intelligence, (submitted).

de Vel O., Anderson A., Corney M., and Mohay G. (2001). "Mining E-mail Content for Author Identification Forensics", SIGMOD Record, **30**(4).

SVMLight (2001). Support Vector Machine software, University of Dortmund, Germany.

Witten I., and Frank E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations,* Morgan Kaufmann, San Francisco.

Yang Y. (1999). "An evaluation of statistical approaches to text categorization", Journal of Information Retrieval, **1**, pp.67–88.

Friedman J. (1991). "Multivariate adaptive regression splines", Annals of Statistics, **19**, pp.1–141.

Hastie T., Tibshirani R., and Friedman J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,* Springer Series in Statistics, Springer-Verlag, New York, NY.

Index

Alert stream, 55, 109, 119
Anomaly detection, 1–3, 9–10, 12, 18–22, 24–25, 42–43, 45–48, 53, 56, 64, 78–82, 85, 91, 95–97, 124–131, 134, 141, 147–150, 154–156, 163, 177, 179–180, 185, 190–191, 203
Association rules, 7, 11, 36–39, 43, 45–46, 48–49, 56, 68–71, 75, 206, 217
Author attribution, 230, 235, 237, 239–240
Buffer overflow, 94, 107
Classification, 1–3, 7, 11, 13–14, 17, 20–21, 23–25, 39–40, 47, 52, 55, 67–68, 80, 89, 94, 125–126, 128–129, 131, 156, 162, 171, 181–185, 187–189, 203, 232, 234, 240, 243–244, 246
Clustering, 40, 43, 47, 52, 86–88, 96, 134, 150
Computer forensics, 230–232, 244–245
Correlation, 38, 50, 54–55, 65, 103–105, 128, 150, 154, 157, 175, 177–178, 191, 223, 236
Data mining, 1–2, 4, 7–10, 14, 17, 20–21, 25, 33–36, 38, 41–43, 48–56, 67, 74–75, 78, 104, 108, 114, 116–118, 120, 124, 133, 150, 153–156, 162–163, 171, 176, 188
Degrees of attack guilt, 16–19
Denial of service, 10
E-mail, 230–234, 237–244, 246–249
Entropy, 125–127, 131, 249
Euclidean distance, 40, 47, 52, 84, 134–135, 142
False alarms, 3, 10, 21, 25, 64–65, 72–73, 75, 203–204, 222
Fragmentation, 147
Frequent episode rules, 38–39, 43, 45, 49, 53
Fusion, 34, 36, 55, 75, 103–106, 108–110, 114, 120, 244
Granger Causality Test, 196, 210–212
Intrusion detection, 1–6, 8, 10, 13–18, 20, 25, 33–34, 36–38, 41–43, 45, 48–56, 74–75, 78–79, 81, 83, 88, 91, 94, 103–105, 124–125, 127–128, 148–150, 153–154, 156, 158–159, 162, 171, 174–175, 177, 181–182, 187–188, 190–191, 223, 225–226, 236
Kernel functions, 86, 90, 182–183, 244
Machine learning, 10, 25, 35–36, 51, 75, 78, 82, 94, 114, 120, 133, 154, 178, 181, 191, 198, 234
MIB II, 124–131, 133–136, 138, 141, 147–150
Misuse detection, 1–4, 6–9, 15–18, 25, 42–43, 78, 124, 127–128, 135, 138, 147, 154–155, 177, 179–181, 183–184
Naive Bayes classifier, 12, 24
Network management systems, 125
Outlier detection, 80–81, 86
Performance, 3, 9, 11–12, 22, 91, 93–94, 97–98, 108, 116–118, 124–126, 131, 134, 141, 147, 149, 175, 178, 188, 190–191, 223, 234, 238–240, 244–246
Port scan, 94, 107, 141
Prediction, 51, 79, 133–134, 214
Privilege, 5, 8, 42, 107, 113, 149, 201
Probability distribution, 82–83, 86
Probe, 7, 10–11
Profile, 2–3, 5, 9–10, 12, 14, 18–23, 25, 42, 45–46, 48–49, 53–54, 64–65, 67–72, 74–75, 79, 109, 124–125, 127–129, 131, 142, 149, 204, 206, 217, 231, 233, 235, 237
Pseudo-Bayes estimator, 22, 24, 74
Reliability, 5, 75
Responses, 197
ROC curves, 94, 97–98
Scalability, 9, 41
Scenarios, 4, 23, 103–106, 108, 110–112, 119–120
Signature analysis, 4, 6, 16
Similarity, 40–41, 47, 52, 105, 119, 235, 246
Sniffer, 160
Spoof, 112–113, 115, 135, 231
Statistical methods, 9, 18–20, 75
Stylometrics, 230
Support vector machines, 157, 162, 183, 230, 233, 247
Validation, 35, 49, 52, 111, 118, 243

Visualization, 154, 156–157, 163, 167, 176–177
Vulnerabilities, 4, 13, 42, 94, 177, 201