THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

A CASE-BASED FRAMEWORK FOR META INTRUSION DETECTION

By

JIDONG LONG

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Degree Awarded:
Summer Semester, 2006

The members of the Committee approve the Dissertation of Jidong Long defended on May 18, 2006.

Daniel G. Schwartz
Professor Directing Dissertation

Jerry Magnan
Outside Committee Member

Mike Burmester
Committee Member

Lois Hawkes
Committee Member

Xiuwen Liu
Committee Member

Approved:

David Whalley, Chair
Department of Computer Science

Joseph Travis, Dean, College of Arts and Sciences

The Office of Graduate Studies has verified and approved the above named committee members.

To my parents.

ACKNOWLEDGEMENTS

My Ph.D. Program has turned out to be a long and challenging, but rewarding, journey. I would not have made it this far without the help of many people. My advisor Dr. Daniel G. Schwartz has taught me to be a hardworking and effective researcher. He was always supportive of every idea I came up with for my research and patient in helping me to refine them. I want to express many thanks to Dr. Jerry Magnan, Dr. Xiuwen Liu, Dr. Mike Burmester, and Dr. Lois Hawkes for serving on my dissertation committee, providing helpful suggestions, and enriching my knowledge. I would also like to thank Dr. Sara Stoecklin, an expert in software engineering, who has shaped me into a good programmer.

I am also grateful to have been part of a collaborative research project wherein we studied problems of critical infrastructure protection for the US army. The intercommunication with researchers from other universities, namely the University of Wisconsin, George Washington University, and the University of Central Florida, has greatly opened my mind and extended my scope of understanding. I offer my special thanks to the DARPA GCP Project for allowing me use their software to evaluate my work.

I have benefited greatly from people who have kindly answered my email questions. I am thankful to Joshua Haines, Richard Lippmann, and David Fried of MIT Lincoln Lab, Dr. Stephanie Forrest of the University of New Mexico, Wenke Lee of Georgia Institute of Technology, and Dingbang Xu of North Carolina State University for helping me better understand their works.

Two good friends of mine, also the team members of our project, Patel Mahesh and Wayne Liu, were great sources for ideas, and fun. Their participation was invaluable. I thank them for making my graduate study enjoyable and memorable.

Finally, I would like to show my true appreciation for the love and support of my family. I dedicate this dissertation to my parents, who continually encourage me to make my best effort to realize my dreams. I also thank my brother for being so supportive and caring for

me.

— Jidong

TABLE OF CONTENTS

List of Tables . . . viii

List of Figures . . . ix

Abstract . . . xi

1. INTRODUCTION . . . 1
- 1.1 Problem Statement and Our Approach . . . 4
- 1.2 Dissertation Contributions . . . 6
- 1.3 Dissertation Outline . . . 8

2. Meta Intrusion Detection . . . 10
- 2.1 Rationale for Meta Intrusion Detection . . . 11
- 2.2 Elements of Meta Intrusion Detection . . . 12
- 2.3 Intrusion Detection Message Exchange Format . . . 14
- 2.4 Related Work in Meta Intrusion Detection . . . 18
- 2.5 Problems with Current Meta IDS and Our Approach . . . 19

3. Case-Based Reasoning . . . 22
- 3.1 Basics . . . 22
- 3.2 The Reasoning Process . . . 23
- 3.3 Case-Based Reasoning Applications . . . 24
- 3.4 The Adaptive Similarity Measure for CBR . . . 25
- 3.5 Our Case-Based Framework for Meta Intrusion Detection . . . 30

4. Data Mining . . . 36
- 4.1 Approaches . . . 37
- 4.2 Related Work on Data Mining in Intrusion Detection . . . 38
- 4.3 A Novel XML Distance Measure . . . 39
- 4.4 A Supervised Clustering Algorithm . . . 51
- 4.5 Evaluation of Our Data Mining Techniques with Snort . . . 53

5. Alert Correlation . . . 62
- 5.1 Related Work . . . 63
- 5.2 Explicit Alert Correlation . . . 64
- 5.3 Case-Oriented Alert Correlation . . . 65

5.4 Enhanced Case-Oriented Alert Correlation 68

6. Experimental Evaluation 80
6.1 The Experiment with DARPA 1998 data sets 80
6.2 Experiment with DARPA 2000 datasets 90
6.3 Experiments with DARPA Cyber Panel Program 92

7. CONCLUSION 99
7.1 Summary 99
7.2 Dissertation Contributions 101
7.3 Drawbacks 102
7.4 Future Work 103
7.5 Closing Remarks 104

APPENDIX 105

A. Hungarian Algorithm 105

REFERENCES 107

BIOGRAPHICAL SKETCH 114

LIST OF TABLES

4.1 Overall results of the first experiment on DARPA 1998 data 53
4.2 Clustering results on DARPA 1998 data 60
4.3 Distribution of 'ftp-write' sessions over assigned pure clusters 61
4.4 Distribution of 'satan' sessions over assigned pure clusters 61
6.1 Clustering results of the initial cases generated for the DARPA 1998 training data.The last column is singletons. 88
6.2 Overall results of the first experiment on DARPA 1998 data 89
6.3 Overall results of the first experiment on DARPA 2000 data 93
6.4 Experimental results with (1,1,1) 95
6.5 Experimental results with (5,5,5) 95
6.6 Experimental results with (10,10,10) 96
6.7 Experimental results with (1,5,10) 96
6.8 Experimental results with (10,5,1) 96

LIST OF FIGURES

2.1 The relationships of the components of meta intrusion detection 14

2.2 Overview of IDMEF Data Model 16

2.3 An Instance of Analyzer Class in an XML Document 18

3.1 A Simple Case-Based Reasoning System 24

3.2 The Reflective Architecture 26

3.3 The Reflective Framework Of the Similarity Measure 28

3.4 UML diagram of the core similarity modules. 29

3.5 The XML representations of a case, a problem and an alert 32

3.6 The Framework of Case-Based Meta Intrusion Detection System 33

4.1 Sample documents XML-1 (left) and XML-2 (right) 43

4.2 Algorithm 1: distance between two attribute sets. 45

4.3 Algorithm 2: distance between two elements. 46

4.4 A distance matrix and its maximal matching with a virtual element 47

4.5 Algorithm 3: distance between two element sets 48

4.6 XML-3 49

4.7 A Sample Distance Matrix 53

5.1 Reduction of alert correlation to matching problems 67

5.2 An example of case-oriented alert correlation 69

5.3 arbitrary matching (a) and (b) and Ordered matching (c) and (d). 70

5.4 Possible matchings for the x_i. 73

5.5 Decomposition of optimization problem into subproblems. 73
5.6 An example of reduction of order matching to shortest path on a graph . . . 76
5.7 Algorithm of ordered maximal matching 77
5.8 Initialize the cache table 78
5.9 An example of changes in a cache table 78
6.1 the fragment of a list file 81
6.2 The data flow in Preprocess 83
6.3 The IDMEF representation of STIDE outputs 83
6.4 the Overview of Training 85
6.5 the Overview of Testing 86
6.6 The three steps of case library construction. 86
6.7 An example of selecting the representative 87
6.8 Service Plot for DDoS1.0 92

ABSTRACT

Intrusion Detection has become an essential component of security mechanisms for information systems. Traditional Intrusion Detection Systems generally apply a single detection model and data source. Thus, they tend to suffer from large numbers of errors. To address this issue, the concept of meta intrusion detection was recently introduced. It suggests combining the results from multiple sensors with the aim of providing global decisions and avoiding errors.

This dissertation describes a novel case-based reasoning framework for meta intrusion detection, including its rationale, design, implementation, and evaluation. Briefly, a case consists of a problem-solution pair, where a problem is an attack and its solution is the type of the attack. Attacks are represented as the collection of alerts arising from sensors. The alerts are encoded in an XML language.

Three experiments were conducted. The first used the 1998 DARPA data sets. Two sensors were employed. For each session, all alerts generated formed a pattern. These patterns were then clustered, and representatives from the clusters were chosen to build a case library. For this purpose an XML distance measure was created, to measure the distance between patterns in XML representation. The clustering very effectively distinguished normal sessions from attack sessions.

A key issue in meta intrusion detection is *alert correlation*, that is, determining which alerts are results of the same attack. The above employed what we have called *explicit* alert correlation. This makes use of session information contained in the alerts.

The second experiment used the 2000 DARPA data sets containing denial of service attacks. Here the original contribution has been a new case-oriented approach to alert correlation which does not require the presence of session information. The experiment showed that this approach can be very effective in detecting new attacks.

The third experiment made use of the DARPA Grand Challenge Problem program. This experiment explored case-oriented alert correlation with two underlying methods, one based on the Hungarian algorithm and one employing dynamic programming. It was found that both methods are effective for attack detection, and produce almost identical results. However, the dynamic programming is significantly more efficient.

CHAPTER 1

INTRODUCTION

Computer and network systems, or information systems in general, are extensively used in modern society. They have become both an irreplaceable part of society's infrastructure and the targets of attacks as well. Therefore, security mechanisms are needed to protect them.

The security of an information system is compromised when an intrusion takes places. An intrusion can thus be defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a resource"[1]. Nowadays, intrusions are generally associated with attacks launched over the Internet against remote systems. It is difficult to prevent such attacks by the use of classical information security technologies, such as authentication and cryptography, because system and application software typically contain exploitable weaknesses due to design and programming bugs. It is both technically difficult and economically expensive to build and maintain an information system that is free of security flaws and not susceptible to attacks. In addition, attackers take the initiative in exploiting unforeseen interactions between software components and protocols. Successful attacks inevitably take place despite the best security precautions. Since it is impossible to prevent security flaws and attacks, intrusion detection is needed as a last line of defense.

This field has been studied for more than twenty years since Anderson's report [2]. A variety of approaches have been explored. Generally speaking, the whole intrusion detection technology is based on two primary assumptions: (1) system activities are auditable or observable via some information sources, such as system logs, sequences of system calls, and network traffic; and (2) normal and intrusive activities have distinct behaviors, for example, an intrusive activity makes the system fail to provide a certain service. Moreover, such distinction can be manifested somehow in some information sources. The task of intrusion detection is therefore to identify normal or/and intrusive activities through the information sources. In order to make it do so, a detection model is necessary that characterizes normal and/or abnormal behaviors in the form of patterns. In addition, a technique should be applied to compare the unknown behaviors with the constructed detection models and indicate those deemed intrusive. In short, targets under protection, audit data (information sources), detection models, and detection techniques are the essential elements of intrusion detection.

According to the types of patterns of interest, there are two general categories of intrusion detection techniques: misuse detection and anomaly detection. Misuse detection (e.g., NetSTAT [3]) looks for patterns of known attacks and any behavior that conforms to the pattern of a known attack is considered intrusive. By contrast, anomaly detection (e.g., NIDES/STAT [4]) looks for patterns of normal behaviors. The deviation of current behavior from normal behavior is considered as potentially intrusive. Alternatively, intrusion detection can be also classified as either host-based or network-based. Host-based intrusion detection (e.g., STIDE [5][6]) uses information sources for a host computer and aims to detect attacks against that particular host. Typical host-based information sources include Sys logs and SUN's BSM (Basic Security Module) audit. Network-based intrusion detection (e.g. Snort and Bro) uses network packets as the information source to detect attacks. It can be used to detect attacks against hosts or against the network itself.

Accuracy is the key issue for any IDS. Although intrusion detection has been explored for a long time, current IDSs still suffer from a large number of mistakes (false positive and false negatives). A false positive happens when an intrusion detection system fires an alert while there is actually no attack. A false negative happens when an intrusion detection system is not able to detect an attack. Two important causes contribute to mistakes by IDSs. First, both misuse detection and anomaly detection have weaknesses. Misuse detection is mainly criticized for being unable to detect new and unforeseen attacks because it only works on patterns of known attacks. Although anomaly detection can detect new attacks, it tends to generate a lot of false positives due to the difficulty of defining, collecting, and updating normal behavior when establishing the detection model. The second cause is the limited information sources used by IDSs. Most IDSs only use a single information source to detect attacks. That is certainly insufficient to provide a good coverage for some attacks. Especially with the widespread use of the Internet, intrusions are becoming more and more complicated. Sophisticated attackers usually incorporate actions spanning over multiple steps, operating systems, and applications to make intrusions successful. For example, some distributed denial of service (DDoS) attacks need to perform IP scans from a remote host to find live machines in a network, and then apply a well-known vulnerability (e.g., buffer overflow) to gain the root access of a victim machine, and finally use that machine to send a flood of network packets to other machines. These intrusive activities are normally recorded in different audit data sources. It is impossible to get a global picture of this attack if only host data source or network traffic is checked. Besides accuracy, extensibility and adaptability are also critical to IDSs. However, almost all IDS projects in the past have largely been pursued in isolation. Much of the effort within a project is specific to a certain data source and a certain detection model. Components, such as collection methods and analysis results, were rarely intended to be shared across IDSs. Furthermore, both new detection approaches and new intrusions are continuously appearing. It is imperative that IDSs be extensible to incorporate additional abilities and, meanwhile, be adaptable to new attacks in a timely manner.

Meta intrusion detection is an emerging concept introduced in the intrusion detection community to help develop better IDSs, meeting the desirable goals, namely, that they be accurate, extensible, and adaptable. The basic idea behind the concept is combining analysis results from multiple IDSs, also called 'sensors' in the context of meta intrusion detection. Each sensor is an independent IDS and interfaces with other intrusion detection components via messages containing analysis results. Incorporating new modules with extra detecting ability are as simple as adding a new sensor. Thus, meta intrusion detection tends to have good extensibility. As these sensors may apply complementary intrusion detection models and work on complementary audit data sources, the meta IDS is potentially able to make more comprehensive decisions than each individual sensor.

The intrusion detection community has made some valuable progress in meta intrusion detection research. For example, the IETF Intrusion Detection Working Group (IDWG) has proposed an alert description language, called Intrusion Detection Message Exchange Format (IDMEF). It is increasingly being accepted as the standard alert format. Current research in meta intrusion detection has been largely concentrated on the problem of alert correlation, namely, determining whether alerts from different sensors are caused by the same attack event. A variety of approaches (e.g. [7][8][9][10]) have been proposed. However, building a meta IDS still heavily relies on human involvement, e.g., hard-coding prior knowledge. This manual and ad hoc nature of the development process is time-consuming and error-prone and limits the benefits and advantages of meta IDSs over traditional IDSs.

1.1 Problem Statement and Our Approach

This thesis research studies the problem of: how to design an "automated" meta intrusion detection system effective in detecting attacks and adaptable to detecting new attacks. To be more specific, there are three main tasks:

- How exactly to combine the results from multiple sensors in order to provide global decisions. This task is often referred to as alert correlation by intrusion detection community and is an essential requirement of meta intrusion detection.
- How to make the process of developing the system automated. Although complete automation is unrealistic, we hope the process will rely as little on human involvement as possible.
- How to provide our meta intrusion detection system with adaptability. This is a desirable goal for every intrusion detection system. It is concerned with discovering new types of attacks.

We expect our approach to be able to work with different IDSs and assume that sensors send alerts in IDMEF, the standard XML alert language. Thus, the information from sensors is streams of IDMEF Messages. We observed that the application of meta intrusion detection is analogues to 'diagnosis' applications. The success of Case-Based Reasoning (CBR) approach in diagnosis applications has inspired us to develop a CBR-based framework for meta intrusion detection. More specifically, we take a different point of view on our problem and consider intrusion detection as a 'diagnosis' process on the protected system according to the information from sensors deployed at different locations. The historical attack records are saved as cases in a case library. Information from sensors is formulated as problems that are compared with the cases in the case library. If there is a case sufficiently similar to a problem, the problem is then considered as representing an attack situation.

We understand that the effectiveness of this approach highly depends on the design of cases and problems and how to measure the similarity between a case and a problem. In order to avoid information loss, the collection of original alerts generated from sensors during an attack or a potential attack is used to describe the situation. As original alerts are IDMEF messages and take the form of XML, cases and problems are also formulated as objects in terms of XML representations, and IDMEF alerts are embedded in cases and problems as XML elements. We seek a thorough analysis approach and make our data analysis algorithms directly work with XML objects. A novel XML distance measure is designed for our needs. It serves as the similarity measure applied by the CBR engine to find cases most similar to problems.

Our CBR approach is analogous to misuse detection, where attacks are identified by their similarity to previously known attacks. However, unlike other misuse detection models where the intrusion patterns are manually analyzed and encoded, our approach eliminates the need for complicated human analysis. An attack is identified as a pattern of alerts from the various sensors. This pattern forms the problem part of a case, and the identity (name or type) of the attack forms the solution part of the case. A clustering approach is applied that groups cases by similarities among their alert patterns (problem parts). Then the knowledge of the attack identities (the solution parts) enables one, for each cluster, to determine whether all the cases in that cluster are cases of the same attack. For each such cluster, one representative case can be chosen as a case for the case library. This has the effect of greatly reducing the size of the library by eliminating redundancy. This in turn greatly enhances the run-time performance of the CBR system.

Our approach additionally has features of anomaly detection, wherein it is adaptable to new attacks. These features are related to the use of a threshold and our alert correlation approach. Firstly, the threshold allows some deviation from known cases. As long as the distance between a problem and a case falls in a certain threshold, it will be considered as representing a certain attack situation. Although the threshold is adjustable, we try to avoid the guesswork in determining it and develop a learning approach to discover the threshold that in some sense best agrees with or supports the training data. Secondly, the proposed case-oriented alert correlation approaches always formulate problems most similar to given cases. It also provides the CBR-based meta IDS, to some degree, with the ability to detect new attacks or variants of known attacks.

1.2 Dissertation Contributions

This dissertation research makes the following contributions to the fields of data mining, case-based reasoning, and intrusion detection:

- **The CBR Approach for Meta Intrusion Detection.** We study the nature of meta intrusion detection and find it analogous to diagnosis applications. Considering that the CBR approach has been successful in diagnosis applications, we explored the application of CBR to meta intrusion detection. This approach saves the previously known attacks as cases. A problem is defined as a pattern of alerts generated during an attack. A case consists of a description of the problem plus a solution. The intrusion detection is thus equivalent to CBR's reasoning process to find a case similar to a problem. We designed cases and problems in a straightforward way and let them take the form of XML objects. Hence, an alert in IDMEF can be easily accommodated in a case or a problem. The design also eliminates the reliance on expert analysis to find the complicated relationship between alerts and encode intrusion patterns. The relationship is already implicit in the description of the problem.

- **Adaptive Similarity Measure.** A similarity measure is a critical component in any CBR system. We study various similarity measures popularly used in CBR and other fields, and find that while these similarity measures will employ different algorithms, they have basically the same functionality. In a typical CBR system, a case is described by a set of predefined features. When comparing two cases, their similarity with respect to each feature is given by an appropriate feature comparator. Then the results of these comparators for all features are combined according to some rule, giving an overall measure of similarity of the cases. Thus, a similarity measure can be generally defined by a set of feature comparators together with a combinational rule. We use an "adaptive" or "reflective" software architecture wherein case features are associated with their comparators and combinational rule dynamically via run-time references to metadata. A new similarity measure can be easily obtained by changing the metadata file without reprogramming other software components.

- **A Novel XML Distance Measure.** We study our need for a similarity measure to compare cases in the form of XML. We develop a distance-based similarity measure for two arbitrary XML documents. The smaller the distance, the more similar. We view an XML document as a collection of XML elements organized in a tree structure where there is only one root element. We first introduce the concept of content-free XML and the way to convert a common XML document into a content-free XML document. Any XML element in a content-free XML document can be represented as a triple with component element name, attribute set, and sub-element set. Thus, an XML document can be represented by a set of elements given in such triple representations. We develop a series of algorithms for computing the distance between two XML attribute sets, two XML element sets, and two XML elements. The distance between two XML documents is determined as the distance between their two root elements.

- **Extension to the Hungarian Algorithm for Distance Computation.** The problem of computing the distance between two sets (XML attribute sets or XML element sets) has been cast into a maximal matching problem (or job assignment problem). The cost for an element in one set matching with an element in the other set is defined as the their distance. The Hungarian algorithm solves maximal matching problems. But it is not perfect for our needs when two sets are not of equal size. Surplus elements in the larger set can never find matches and they do not account in the overall distance if the Hungarian algorithm is straightforwardly applied. In order to make all elements have matches, we create a 'virtual' element in the smaller set for each surplus element in the larger set. A virtual element is defined as having the maximal distance to any other element.

- **Extend Data Analysis Algorithms to Work with Objects in Terms of XML Representations.** Many well-known data analysis algorithms in knowledge and learning systems work only with objects in attribute-value representations. But objects in our approach employ XML representations. If we transform XML representations to traditional attribute-value representations for the purpose of using those algorithms, it will cause information loss and redundant data. We believe it is better to perform analysis directly over it natural forms. We replace distance functions (or distance measures) used in those algorithms with our own distance measure to make them work with XML objects. For example, a hierarchical clustering algorithm is developed based on our XML distance measure and used in the construction of case library; the reasoning process for finding the most similar case is actually a 1-nearest neighbor algorithm.

- **Case-Oriented Alert Correlation.** We study alert correlation, one of the critical problems in field of meta intrusion detection. It is also referred to as problem formulation in the context of CBR. As there is no knowledge to decide which alerts from different sensors are caused by the same attack, we choose to use the known cases in the case library to direct alert correlation. We are aware of the connections between alerts in a case is still implicit so we consider them as a whole in alert correlation. Given a known case, we assume there is an on-going attack the same as the one in the case. We develop our algorithm to look for a group of currently generated alerts that can best match the alerts in the case. The group of alerts is formulated as a problem. If it turns out the problem is close to the given case, it is reasonable to believe the attack represented by the case is likely to have occurred.

- **Evaluation.** We choose the well-known DARPA data to evaluate our approach. For our evaluation purpose, Snort and STIDE are picked as sensors to process network data and host audit data respectively. STIDE was modified to be able to generate IDMEF alerts. We apply a data mining approach to automatically construct a case library from DARPA 1998 training data and learn the threshold for intrusion detection. Our data mining algorithms based on the XML distance measure are shown to be able to distinguish different types of attacks. The constructed case library (detection model) was tested with 1998 testing data. The results showed that CBR for meta IDS made remarkable improvement in detection performance over individual sensors. Our evaluation on DARPA 2000 dataset has demonstrated that our approach has the ability to detect the variant of old attacks.

1.3 Dissertation Outline

The rest of thesis is organized as follows. Chapter 2 states in more detail the rationale of meta intrusion detection, reviews representative research efforts, and examines the challenges of building meta intrusion detection systems. Chapter 3 briefly describes the technologies of case-based reasoning and outlines the case-based framework for the application domain of meta intrusion detection. Chapter 4 describes the data mining approaches, namely, an XML distance measure and a supervised hierarchical clustering algorithm, we have developed. They serve as the basis for building a case-based meta intrusion detection system. Chapter 5 describes our proposed solutions for the problem of alert correlation, a common challenge for all meta intrusion detection approaches. Chapter 6 describes the experiments to evaluate our approaches using three different DARPA datasets. Chapter 7 summarizes the thesis, discusses main drawbacks and open issues for future work.

CHAPTER 2

Meta Intrusion Detection

Meta Intrusion Detection is a quite new concept in the intrusion detection community. There is still not a generally accepted definition for it so far. Ho [11] describes meta intrusion detection as "a technology that allows a single security console to accept from and communicate with all deployed devices that are from different vendors". Loshin [12], from Information Security Magazine adds that meta IDS is "a system that can accept alerts from all deployed security devices, exact useful information and present that information in a manageable format". Despite the differences, there are at least two aspects that every meta IDS should have in common:

- It should work with multiple sensors (or other types of security devices, e.g. firewalls). For this reason, this technology is also called multi-sensor IDS in some works [7][13].
- It performs data analysis based on meta data, namely, the information from sensors, mainly alerts, instead of the raw data sources commonly used by traditional IDSs. With the growing acceptance of IDMEF as the standard alert format for IDSs, meta data for meta intrusion detection usually takes the form of IDMEF messages.

Meta IDSs may be viewed as the extension of traditional IDSs. Sensors in a meta IDS are independent IDSs that can apply different detection models and work on different audit data sources for intrusion detection. They send their results to the meta IDS for further analysis. As these sensors may apply complementary intrusion detection techniques, e.g. misuse detection or anomaly detection, and work on complementary data sources, e.g. host-based or network-based, the meta IDS potentially is able to make more comprehensive decisions than each individual sensor acting alone. Currently, meta intrusion detection is more generally referred to as a technology providing cooperation among different IDSs. Based on this point of view, any approach that involves cooperative behaviors among IDSs will fall within the scope of meta intrusion detection.

2.1 Rationale for Meta Intrusion Detection

Since the first detection model was presented by Dorothy Denning [14], extensive efforts have led to a variety of intrusion detection techniques, which have been applied to numerous data sources and platforms. But most intrusion detection related projects have been largely pursued in isolation. They have concentrated primarily on very specific event streams and analysis objectives; and components in IDSs were rarely intended to be shared across IDSs. This is mainly due to incompatible audit and alert formats [15]. Hence, traditional IDSs normally work alone and are limited by using a single audit data source and a single detection model. On the one hand, every detection model can apply misuse detection or anomaly detection. Both detection techniques have weaknesses. Misuse detection is mainly criticized for being unable to detect new and unforeseen attacks because it only works on patterns of known attacks. Although anomaly detection can detect new attacks, it tends to generate a lot of false positives due to the difficulty of defining, collecting, and updating normal behavior when establishing the detection model. On the other hand, applying a single data source for intrusion detection cannot provide a good coverage for attacks, especially with attacks becoming more sophisticated and more widespread. For example, some distributed denial of service (DDoS) attacks need to perform an IP scan from a remote host to find live machines in a network, and then apply a well-known vulnerability (e.g., buffer overflow of some application) to obtain the root access of a victim machine, and finally use that machine to send a flood of network packets to other machines. These intrusive activities are normally recorded in different audit data sources. It is impossible to get a global picture of this attack if only host data source or network traffic is checked.

There is no doubt that IDSs should pursue a high detection rate. However, too many false positives, for example 100's per day, can also make IDSs almost unusable, even with a high detection rate. Experience has taught the intrusion detection community that the most powerful monitoring solution is one that allows the collection of information from multiple levels of abstraction in the system, from multiple hardware and software platforms, and in different data formats [16][17]. The need for meta IDSs arises in the first place from their better detection performance than transitional IDSs. According to [18], meta intrusion detection can be applied to the following detection scenarios that involve cooperation between IDSs for the improvement of the overall detection performance:

- Analyzing
- Complementing
- Reinforcing
- Verifying
- Adjusting Monitoring
- Responding

A number of commercial and free IDSs are available and more are becoming available all the time. Some are aimed at detecting intrusions on the network; others are aimed at host operating systems; while still others are aimed at applications. Even within a given category, the products have very different strengths and weaknesses. Hence it is likely that users will deploy more than a single product, and users will want to observe the output of these products. The use of meta intrusion detection makes it easy to integrate new IDSs with new detection approaches and new audit data sources. Thus, Intrusion Detection research should migrate into commercial products more easily. Such extensibility also contributes to the need for meta intrusion detection.

In addition, meta intrusion detection is necessary as the solution for enterprise-wide security deployments. A traditional IDS is no longer suitable for current network topologies due to their lack of scope. Intrusions nowadays frequently involve multiple organizations as victims, or multiple sites within the same organization. Typically, those sites will use different IDSs (Sensors) to monitor different portions of the network. It would be very helpful to correlate such distributed intrusions across multiple sites and administrative domains.

2.2 Elements of Meta Intrusion Detection

Meta intrusion detection has much in common with traditional intrusion detection simply because they are all aimed to detect attacks. Meta intrusion detection can be viewed as an extension of traditional IDSs (as shown in Figure 2.1). Hence, it has what all traditional IDSs should have, although some elements use different terms in the context of meta intrusion detection.

- Data Sources: the raw information that an IDS uses to detect attacks. Common data sources include, but are not limited to, network traffic, operating system audit logs, and application audit logs. Multiple data sources can be involved simultaneously under the framework of meta intrusion detection.

- Sensors:the components that can collect data from data sources. The sensors are space to record activity and forward events to the analyzer. Typical sensors include tcpdump and Basic Security Monitor (BSM). The former is a Unix tool that collects network traffic and records it as tcpdump data. The latter is a Solaris tool that collects information about system calls and saves it in the BSM audit files.

- Analyzers: the component that analyzes the data collected by the sensor for signs of undesired activity or for events that might be of interest to the security administrator or other intrusion detection components. In most circumstances, analyzers report critical findings in the form of alerts. In many existing IDSs, the sensor and analyzer are part of the same component. Thus, we call those IDSs 'sensors' in general if they are deployed for meta intrusion detection. For example, the common rule-based IDS Snort is actually a combination of a sensor and a rule inference engine. It can collect network traffic as tcpdump does, and also fire alerts if the attack signatures are found. We use Snort as a sensor for the evaluation of our case-based meta intrusion detection approach(Chapter 6). Although the terms, 'analyzer' and 'sensor', are both used in the thesis, their difference should be clear from the context of our discussion.

- Monitor: the component where the alerts from various analyzers are collected and further decisions will be made. A monitor refines the analysis results from analyzers in order to provide more comprehensive decisions. It may generate hyper-alerts or report decisions in some other forms to the response system or the users of the IDSs.

- Detection Models that characterize the "normal" or "abnormal" behavior of the activities. Every analyzer and the monitor have their own detection models. For

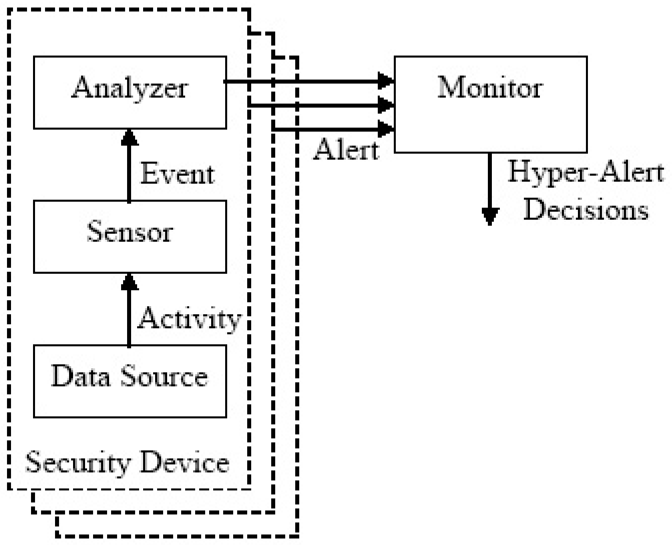


Figure 2.1: The relationships of the components of meta intrusion detection

example, we use a case-based approach for meta intrusion detection; Snort is one of the sensors selected for the evaluation of the approach. The detection model of the monitor is the case library and the detection model of Snort is the set of applied rules.

2.3 Intrusion Detection Message Exchange Format

The IDMEF, proposed by the Intrusion Detection Working Group (IDWG) of the Internet Engineering Task Force(IETF), is intended to be a standard data format that automated IDSs can use to report alerts about events that they deem suspicious. Existing IDSs using different alert formats curbs communication between them. The use of IDMEF enables interoperability among various IDSs. The most obvious place to implement IDMEF is in the data channel between a sensor and the monitor to which it sends alerts. But there are other places where the IDMEF can be useful:

- a single database system that could store the results from a variety of IDSs would make it possible for data analysis to be performed on a global basis instead of a part of it. We will see later on in this thesis that cases and problems are constructed from a database that stores IDMEF alerts from two different sensors, Snort and STIDE;

- an event correlation system that could accept alerts from a variety of IDSs would be capable of performing more sophisticated cross-correlation than one that is limited to a single IDS. Our case-oriented alert correlation approach discussed in Chapter 7 is designed to work directly on IDMEF alerts.

- a graphical user interface that could display alerts from a variety of IDSs. A common data exchange format would make it easier to not only exchange data, but also communicate about it.

In our approach, all alerts are compliant with the IDMEF.

2.3.1 The Flexibility of IDMEF

Alert information is inherently heterogeneous. Some alerts are defined with very little information, such as origin, destination, name, and time of the event. Other alerts provide richer information, such as ports or services, processes, user information, and so on. IDMEF is flexible to accommodate different needs because it chooses an object-oriented model that is naturally extensible via aggregation and sub-classing. Sub-classing and aggregation provide extensibility while preserving the consistency of the model. If an implementation of the data model extends it with new classes, either by aggregation or sub-classing, an implementation that does not understand these extensions will still be able to understand the subset of information that is defined by the data model.

There are different types of Intrusion detection environments. Some analyzers detect attacks by analyzing network traffic; others use system logs or application audit data. Alerts for the same attack, sent by analyzers with different data sources, will not contain the same information. The IDMEF data model defines support classes that accommodate the differences in data sources among analyzers. In particular, the notion of source and target for the alert are represented by the combination of Node, Process, Service, and User classes.

Different analyzers have different capabilities. Depending on the environment, one may install a lightweight analyzer that provides little information in its alerts, or a more complex analyzer that will have a greater impact on the running system but provide more information in alerts. The IDMEF data model allows for conversion to formats used by tools other than intrusion detection analyzers, for the purpose of further processing the alert information. It

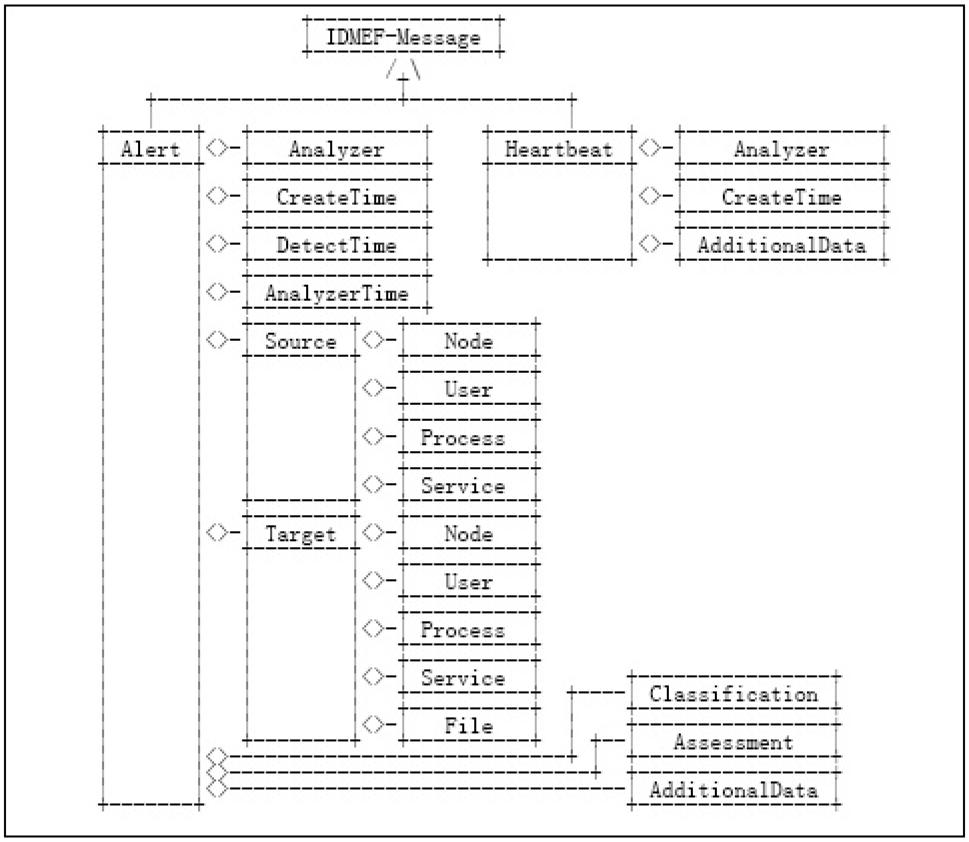


Figure 2.2: Overview of IDMEF Data Model

defines extensions to the basic schema that allow carrying both simple and complex alerts. Extensions are accomplished through sub-classing or association of new classes.

There are different kind of operating environments. Depending on the kind of network or operating system used, attacks will be observed and reported with different characteristics. The IDMEF data model accommodates these differences. Significant flexibility in reporting is provided by the Node and Service support classes. If additional information must be reported, subclasses may be defined that extend the data model with additional attributes.

2.3.2 The XML Implementation of IDMEF

The Extensible Markup Language (XML) is gaining widespread attention as a language for unambiguously representing, interpreting, and exchanging data across platforms. It is a meta-language, a language to describe other languages, providing both a syntax for declaring document markup and structure as well as a syntax for using that markup in documents.

XML was chosen by the IDWG for implementing the IDMEF. Its flexibility and popularity makes it a good choice. To be more specific:

- XML allows the definition of both a custom language to be developed specifically for the purpose of describing intrusion detection alerts and a standard way to extend this language for later revisions.

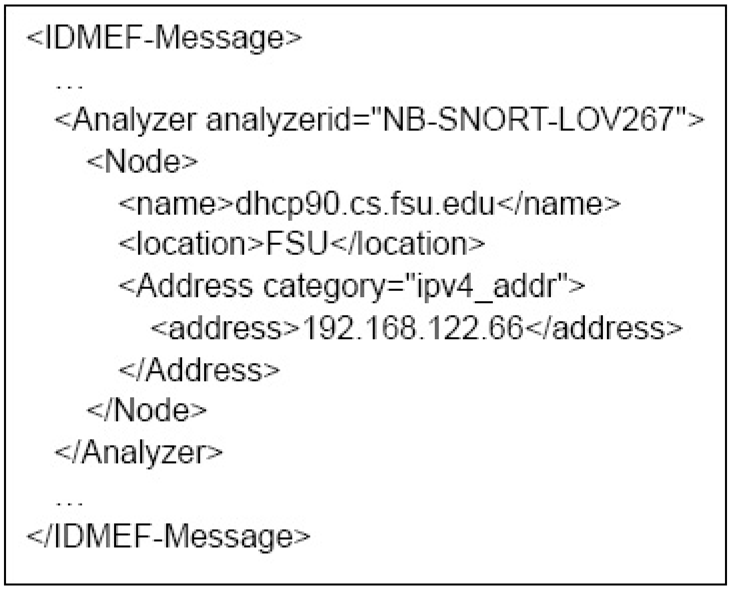
- XML has substantially available supports, including software tools for processing XML documents and APIs for parsing and validating XML in a variety of languages, such as Java, C, and C++. Widespread access to tools will make adoption of the IDMEF easier, faster, and more convenient.

- XML is intended to be an internationally accepted standard and supports common character encodings (e.g. Unicode and UTP-16).

- XML can support filtering and aggregation if integrated with XSL, a style language. More features, such as object-oriented extensions and database support, that are being standardized into XML will be immediately gained, if IDMEP adopts XML.

- XML is free.

The XML data model IDMEF has implemented (in the form of XML DTD or XML Schema) is graphically presented in Figure 2.2. Each box in the figure stands for a class that, if it is present, is further refined by its sub-classes or aggregate classes. For example, all IDMEF messages are instances of the IDMEF-Message class; it is the top-level class of the IDMEF data model. There are currently two subclasses of IDMEF-Message: Alert and Heartbeat. Heartbeat messages are used by analyzers to indicate their current status to monitors. Alert messages are for reporting suspicious events. An alert message is composed of several aggregate classes such as Analyzer, Source, Target, DetectTime, and so on. These aggregate classes provide more detailed information about the alert. They are further described by their subclasses or aggregate classes. Figure 2.3 gives an instance of the Analyzer class (an XML element with a tag of 'Analyzer'), illustrating how detailed information is represented in a real XML file.



```
<IDMEF-Message>
  ...
  <Analyzer analyzerid="NB-SNORT-LOV267">
    <Node>
      <name>dhcp90.cs.fsu.edu</name>
      <location>FSU</location>
      <Address category="ipv4_addr">
        <address>192.168.122.66</address>
      </Address>
    </Node>
  </Analyzer>
  ...
</IDMEF-Message>
```

Figure 2.3: An Instance of Analyzer Class in an XML Document

2.4 Related Work in Meta Intrusion Detection

A first attempt to design an IDS with cooperative functionality between different approaches was suggested in IDES [19] and then refined in EMERALD [20]. More recently, in the JAM project, Stolfo et al. have applied a meta-learning approach to combine the results from different detectors [21][22], which has proven to be effective for fraud and intrusion detection in financial information systems. Those early efforts view intrusion detection as a classification problem and use classifiers as detectors. Classifiers do not generate alerts. The results from them simply indicate some categories. However, such cooperation functionality is restricted within their own IDSs.

The Common Intrusion Detection Framework (CIDF) is an effort addressing the interoperability between different IDSs. It has specified a language that would enable interactions between intrusion detection components and has outlined the ways in which they would interact. The CIDF encouraged the creation of IDWG. Much of the work (e.g. IDMEF) of the IDWG was initially inspired by the ideas in the CIDF.

Current research in meta intrusion detection has largely focused on the problem of alert correlation which is concerned with determining whether alerts from different security devices are caused by the same attack. There have been several proposals. Approaches, such as probabilistic alert correlation [10] and alert clustering methods [7][23], are based on the similarity between alert attributes. Although they are effective in finding similar alerts, approaches of this type are criticized for not being able to discover deep logical connections between alerts. Some approaches [24][25] correlate alerts under the guidance of specified attack scenarios. They are restricted to known attacks or variants of known attacks. Approaches based on prerequisites and consequences of attacks [13][26][9] may discover novel attack scenarios. These correlate alerts if the prerequisites of some later alerts are satisfied by the consequences of some earlier ones. However, in practice, it is impossible to predefine the complete set of prerequisites and consequences. In fact, some relationships cannot be expressed naturally with the given set of terms. Some approaches (e.g., [27][8] apply additional information sources, such as firewalls and system states to assist in alert correlation. In particular, Morin et al. [27] have proposed a formal model, M2M2, for alert correlation using multiple information sources.

Some currently well-known research projects towards developing meta-IDS include MIRADOR [7][13], M-Correlator [8], and GrIDS [28].

2.5 Problems with Current Meta IDS and Our Approach

Generally speaking, a meta IDS relies on the outputs of multiple IDSs to improve the overall detection performance that a single work-alone IDS can not achieve. The outputs of IDSs are alerts for events that are deemed suspicious. Thus, how to find valuable information from alert messages for systematic analysis is a critical issue to every meta intrusion detection approach. A common strategy used by current approaches is abstracting every alert message with a predefined set of attributes. An important reason for doing that is that popular data analysis algorithms normally work with data described by a set of attributes (attribute-value representation). However, alert information is inherently heterogeneous. Some alerts are defined with very little information, while others provide more detailed information. It is difficult to define a set of attributes that cover the information for all alerts from different sensors. Either redundant information or missing information will occur. Moreover, alerts come as IDMEF messages that take the form of XML. XML represents structured data. If an XML document is transformed into a set of attributes, its structure information will be lost. Hence, information loss is inevitable if alerts are described by a predefined set of attributes.

What is worse is that this may impact the adaptability and extensibility of meta IDS. As new types of attacks, as well as new detection approaches, continue to appear, sensors may generate alerts with important information that was not anticipated when the meta IDS was developed. New attributes need to be introduced to represent the new important information. Without new attributes in the set, the signs of new attacks can become invisible and the power of new detection approaches can also become useless due to the loss of important information. Our approach solves the information-loss problem by performing data analysis directly over objects in terms of XML representation instead of attribute-value representation. A general distance measure between two arbitrary XML documents was developed as the basis for our analysis algorithms. A reflective metadata implementation makes the distance measure adaptive to changes. Users can simply modify the metadata to specify important information in alerts without reprogramming. Some common effective data analysis algorithms can apply the XML distance measure with little modification.

The problem of alert correlation is regarded as the most important and challenging issue in meta intrusion detection research. It is concerned with determining whether alerts from different security devices are caused by the same attack. The general strategy is to define the connections between alerts in certain forms, such as rules and terms, (e.g. [26][25][13]) and use them as the knowledge to guide alert correlation. One problem with this strategy is that attacks are diverse in nature and it is infeasible to predefine all possible rules even for known attacks. In fact, some relationships cannot be expressed naturally in rigid terms. Moreover, discovering the connections between alerts and constructing the knowledge base for alert correlation need careful investigation by human experts and thus are time-consuming and error-prone. This also makes the setup of a meta IDS rely heavily on human involvement. Our approach pursues a way of design that tries to involve the human as little as possible. We observed that for the purpose of intrusion detection, the detail of relationships among alerts is unnecessary. Hence, unlike other approaches, our approach avoids revealing the details of the connections between alerts and simply collects alerts related to the same attack together as a case. Although the connections between alerts are not specified, they are implicit in the case. The cases of known attacks are accumulated in the case library that is used as the knowledge for alert correlation. We developed a case-oriented alert correlation approach. For each case in the library and from currently generated alerts, it looks for an alert pattern that is most similar to the one in the case. If they are similar enough, it is reasonable to believe that the alerts in the new pattern are related to an attack of the same type as that which the case represents.

CHAPTER 3

Case-Based Reasoning

Case-based reasoning (CBR) is a method of solving problems by comparing an unknown situation (termed a ‘problem’) to previously experience ones (termed ‘cases’). The comparison is accomplished by applying a similarity measure. The most similar case is usually used to suggest a solution for the unknown situation. Although the entire technology is based on a simple idea, it has been proven effective in solving quite complicated problems, especially in diagnosis applications. Inasmuch as intrusion detection bears a considerable amount of similarity to diagnosis problems, this has inspired us to devise an approach that applies CBR to meta intrusion detection. This chapter first introduces the basic concepts and methods of CBR. Earlier work on an adaptive similarity measure framework for CBR [29] is then described inasmuch as we have used it again here to implement the case-based reasoner used in our experiments. In addition, this chapter shows our design of the CBR system for the application domain of meta intrusion detection. A few issues arising particularly from the design are also mentioned in this chapter. These will be addressed and more clearly stated in the following sections of this dissertation.

3.1 Basics

Since the 1980’s CBR has grown into a field of widespread interest, both from an academic and commercial standpoint [30]. Basically, CBR is motivated by observing how humans reason. When faced with a problem situation, humans often remember how they were able to solve a similar situation, rather than think through the entire problem from scratch. CBR is a decision support methodology that uses earlier experiences, or cases, as the basis to make decisions. It can be defined as “to solve a problem, remember a similar problem you have solved in the past and adopt the old solution to solve the new problem.”[31] A more formal definition is given in [32]:

> "Case-Based Reasoning is a problem solving paradigm which utilizes the specific knowledge of previous experiences, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing its solution in the new problem situation. CBR is a cyclic and integrated process of solving a problem and learning from this experience."

In a CBR system, expertise is embodied in a library of past cases, where a case is comprised of a problem-solution pair. A problem is typically described in terms of a set of predefined features. A solution may simply be an identifier, such as the name or type of an attack, or a prescribed response. The knowledge and reasoning process used by an expert to solve the problem is not recorded, but is implicit in the solution.

Different CBR systems may have different application domains. In any such domain the problem component of the cases will have features appropriate for that domain. In general, a feature is composed of a feature name and a feature value. A set of predefined features plus their corresponding values constitute the description of a situation (case). From the view of describing objects, features are the same as 'attributes' commonly seen in other knowledge and learning systems. We do not make the distinction between features and attributes and sometimes use these terms interchangeably.

3.2 The Reasoning Process

All case-based reasoning methods have a few operations in common, namely, retrieving, reusing, revising, and retaining cases. They deal with different issues during the reasoning process, such as how to find the most related cases and how to enrich the knowledge for the CBR system. The reasoning process can be generally described as follows (Figure 3.1). A problem is first formulated or abstracted from the working environment and is then compared with cases in the case library. The similar cases are retrieved and then used to suggest a solution for the present problem. The solution is applied and is tested for success back in the environment. If it turns out that the solution is not good enough to solve the problem, the solution is revised (possibly found by human experts or other methods afterwards) and, if the revised solution is effective, a case containing the description of the problem plus the final solution will be created and retained in the case library for future reference.

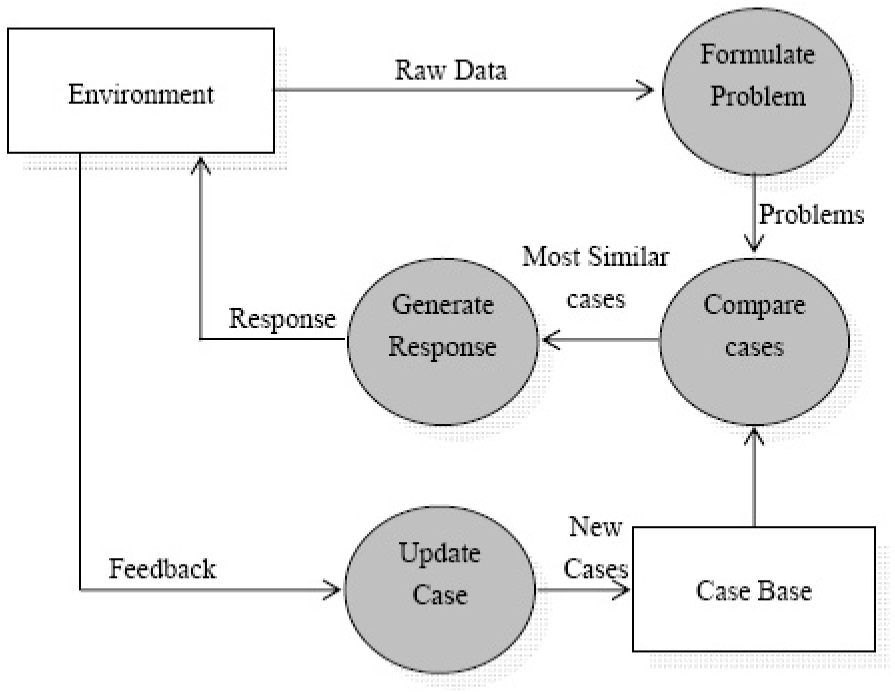


Figure 3.1: A Simple Case-Based Reasoning System

3.3 Case-Based Reasoning Applications

CBR has been pursued to create numerous applications in a wide range of domains since it first appeared in commercial tools in early 1990s:

- Diagnosis: case-based diagnosis systems try to retrieve past cases whose symptom lists are similar in nature to that of the new case and suggest diagnoses based on the best matching retrieved cases. The majority of installed systems are of this type and there are many medical CBR diagnostic systems [33].
- Help desk: case-based diagnostic systems are used in the customer service area dealing with handling problems with a product or service.
- Assessment: case-based systems are used to determine values for a variable by comparing it to the known value of something similar. Assessment tasks are quite common in finance and marketing domains.
- Information retrieval and decision support: in decision making, when faced with a

complex problem, people often look for analogous problems for possible solution. CBR systems have been developed to support this problem retrieval process to find relevant similar problems. CBR is particularly good at querying structured documents.

Case-based reasoning is preferred over more traditional rule-based and model-based approaches when experts find it hard to articulate their thought processes when solving problems. Knowledge acquisition for a classical knowledge-based system turns out to be extremely difficult in a certain class of domains, and is likely to produce incomplete or inaccurate results. In the domain of intrusion, artificial intelligence techniques have gained a lot of attention. Attempts have been made to develop rule-based and model-based expert systems for intrusion detection. Although these systems have been useful for detecting intrusions, they face difficulties in acquiring and representing the knowledge. The issue becomes more critical when coping with meta intrusion detection, where intrusion behaviors are much more complicated. It is not easy to gain a good insight regarding intrusions from a broader perspective. Moreover, in certain cases, the knowledge about attacks cannot be naturally described in rigid terms. However, when using case-based reasoning, the need for knowledge acquisition can be limited to establishing how to characterize cases. In order to avoid the difficulties in acquiring and representing the knowledge, we present and describe a case-based reasoning approach to meta intrusion detection which alleviates some of the difficulties of current approaches.

3.4 The Adaptive Similarity Measure for CBR

A similarity measure is a critical component in any CBR system. Although different systems may apply different similarity measures for their domains, a similarity measure can be generally defined by a set of feature comparators together with a combination rule. More clearly, a case is represented by a set of predefined features. A similarity measure compares two cases with respect to their "features", with each feature using a separate "comparator". The results of the comparators are combined according to some rule to give an overall measure of the similarity between the given cases. The combination rule may normalize and/or weight the results from the comparators to reflect the relative importance of individual features. Similarity measures differ mainly in the selection of comparators for features and the combination rule. The existence of many similarity measures does not always give a clear

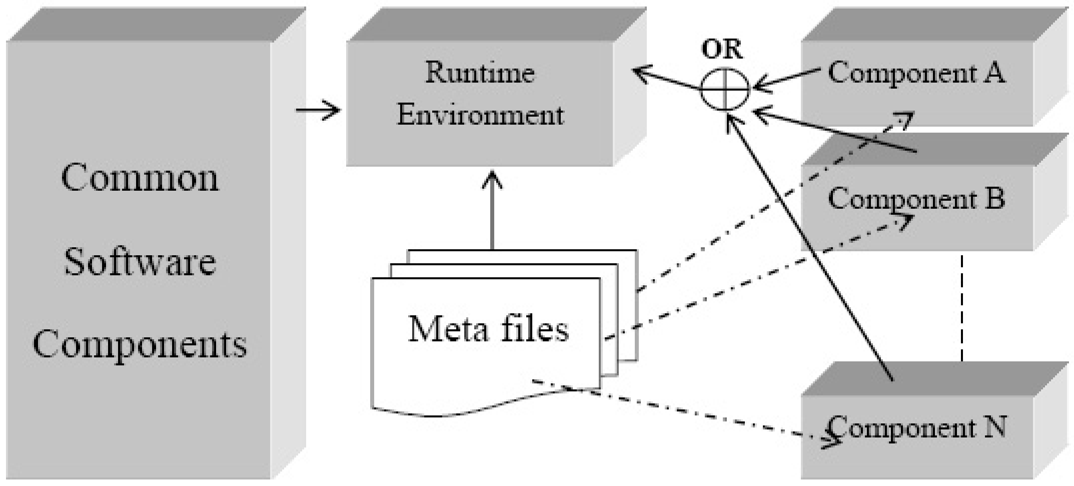


Figure 3.2: The Reflective Architecture

choice of an appropriate measure for a particular domain. Hence, in order to find a good similarity measure for a particular domain, people usually have to experiment with different measures and select one that works best. This process often needs repeating programming work. Moreover, having a fixed similarity measure makes the CBR system only applicable to limited problem domains.

Previous works [34] have described a CBR framework that can easily be instantiated to provide a case-based reasoner for virtually any problem domain. This uses an "adaptive", or "reflective" software architecture wherein case features are associated with their comparators dynamically via runtime references to metadata. New instances of the framework are created simply by changing the metadata. No reprogramming is required. We extend this concept to allow for dynamic selection also of feature-comparator combination rules. This makes the framework more adaptive by eliminating the need to reprogram it for each such new rule. The overall effect is that the entire similarity measure is described by metadata.

3.4.1 Adaptive (Reflective) Architectures

Architectural frameworks capable of dynamically adapting at runtime to new user requirements are called "adaptive architectures", "reflective architectures" or "meta-architectures" [35]. Such architectures emphasize the flexibility of having business rules and algorithmic information dynamically configurable. They represent a system of classes, attributes, relationships, and behavior as metadata, rather than as code embedded in an executable component. The metadata contains the predictable changes to the domain and shifts that information about the domain from actual source code into metadata. Changes to this metadata dynamically change the generic code associated with the adaptive system. Thus by merely changing the metadata, users can change the behavior of the program without changing the code.

Figure 3.2 illustrates an example of software with adaptive architecture design. Under the adaptive architecture, components in any software can be divided into two types in terms of their adaptability. Components of the first type are necessary for the software to function properly. They are common for all application domains. Components of the second type are selectable. They have similar functions but they are implemented for different application domains. Each application domain is associated with a metadata file that specifies which components are employed for this particular domain. Those components are dynamically loaded for execution if necessary. When switching to a new application domain, the user of the software simply writes a new metadata file for the new domain. The software will automatically instantiate itself according to metadata.

From the description, the key idea in reflective architecture is to use metadata, instead of behavioral programming, to control the matching of individual features of the particular CBR domain. In this work, we extend the reasoning capability to allow the selection of not only the comparator for the individual feature but also of the rule used to combine the results of the comparators. This is accomplished by adding a reference to the combination rule as an additional item of metadata. The overall structure of the resulting system is shown in Figure 3.3.

Adaptive programming offers a solution by allowing trials with various similarity measures within various domains. It provides a framework to build each comparator and each combination rule as a separate module, which are then selected dynamically using metadata, rather than coding them directly within the CBR system itself. Thus, through metadata, sophisticated users will have the ability to specify all aspects of a similarity measure and to have those specifications be reflected in the software without the intervention of a programmer. It is only required that certain limits and constraints be observed so as to not compromise the existing software.

As indicated in the figure, there will be a library of similarity measures containing the

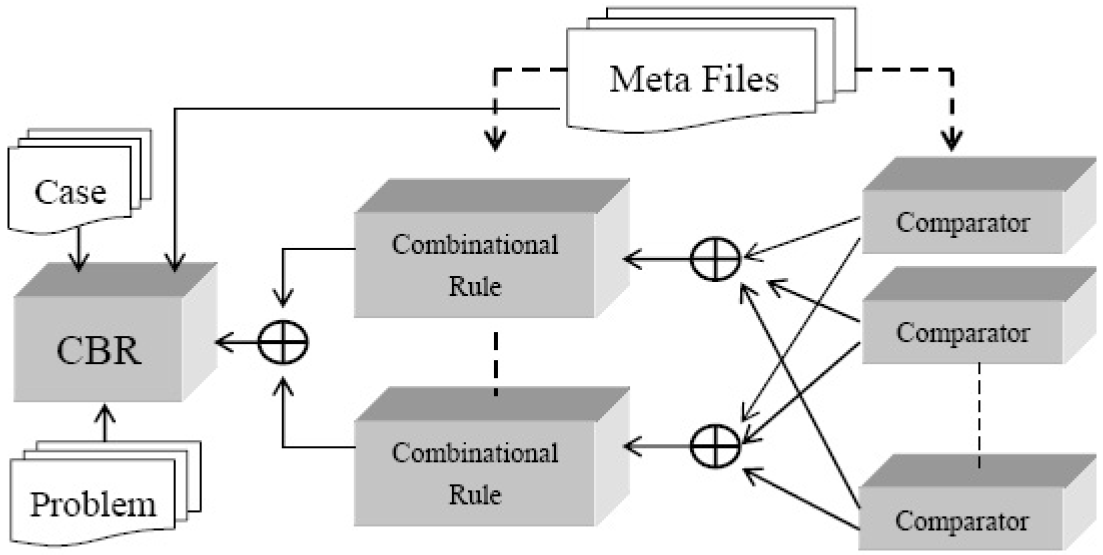


Figure 3.3: The Reflective Framework Of the Similarity Measure

implementations of various similarity measures available for use by the adaptive CBR. This library will contain both comparators and combination rules, each written as a separate class, but with combination-rule classes making reference to comparator classes. It should be noted that combination rules can be quite complex and may need to accommodate comparators that return values of different types, e.g., Boolean, numeric, or even linguistic (strings). New similarity measures can be introduced into the library at will. The metadata files specify both the comparators for features and the combination rule. The CBR system instantiates the similarity measure according to the selected metadata file for a particular domain.

A common interface is defined for the various implementations. This interface allows the case-based system to invoke any of the similarity measures using the same code with the appropriate metadata. Thus the interface acts as a high-level abstraction that lets the implementation details of the similarity metrics be hidden.

3.4.2 A Common Similarity Measure Interface

The main reason one can use an adaptive architecture for the technology of the similarity measures is that all such measures have basically the same functionality. This may be summarized as: (1) accept two cases as input, (2) apply the appropriate comparators to their various features, (3) combine the results of the comparators according to some rule,

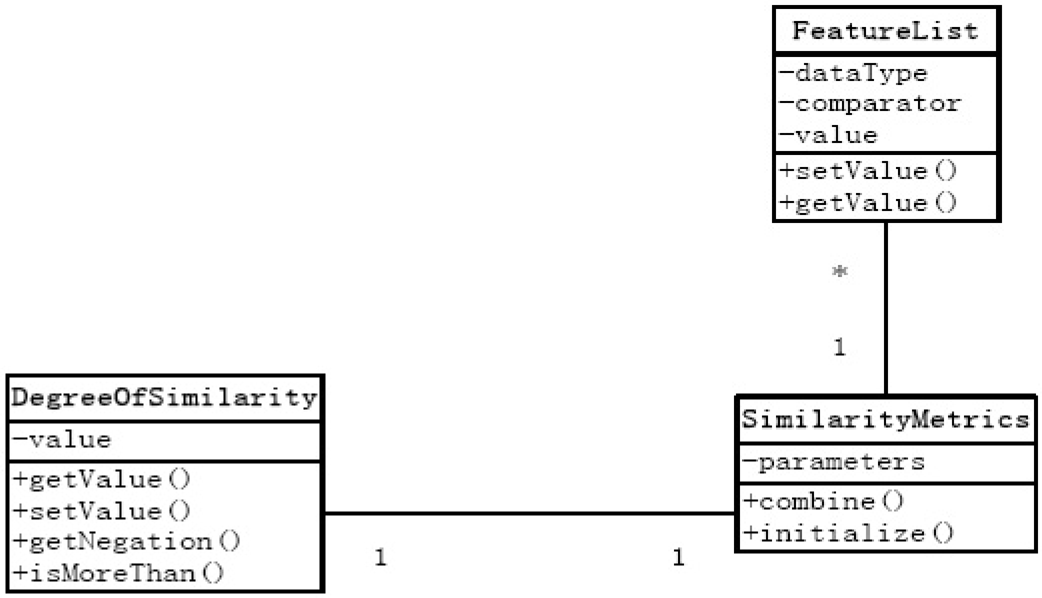


Figure 3.4: UML diagram of the core similarity modules.

and then (4) return the result of the combination.

This drives us to design a common interface at a high-level of abstraction usable for all similarity measures. This higher-level interface is not concerned with the details of the implementation of each individual similarity measure. A UML diagram of the design is shown in Figure 3.4.

The class *SimilarityMetrics* takes the inputs of problem features and case features embodied in the class *FeatureList*. The combination rule is implemented by the method *combine()* in the class *SimilarityMetrics*. The outcome of the similarity metric is encapsulated in the class *DegreeOfSimilarity*. Each instantiation of *SimilarityMetrics* will either use a new *DegreeOfSimilarity* class, or reuse one that has been created previously. The type of value returned by a similarity measure may be boolean(true or false), double(0-1 possibility), string (good, fair, excellent, etc), a fuzzy set, or any other type of object. The method *isMoreThan* is used to compare two values from the same similarity measure.

Typically, one similarity measure will suffice for any given CBR application domain, in which case the *isMoreThan* method is always applied to two values of the same type.

3.5 Our Case-Based Framework for Meta Intrusion Detection

As we have mentioned earlier, the majority of installed CBR systems are diagnostic systems. If an information system is viewed as the subject to be diagnosed, intrusion detection is like a diagnosis problem since they have nearly the same objective, discovering abnormal signs. Considering the success of CBR in diagnosis, CBR can be an effective approach for intrusion detection. In fact, CBR has already been explored in this field [36]. Our approach differs from the former effort with respect to where case-based reasoning is applied. The former applied CBR to a single IDS, whereas we have concentrated on a meta IDS working with multiple strategically deployed sensors (IDSs).

There are two main problems in current meta intrusion detection research. One is information loss. The information from sensors are IDMEF messages in the form of XML, which is structured data. As most available analysis algorithms only work on data formatted in tables, information loss occurs when XML data are flattened into such tables in order for those algorithms to process. CBR is known to be particularly good at querying structured documents. It inherently has advantages over other approaches in dealing with XML data like IDMEF messages. The other problem in meta intrusion detection is alert correlation. A common strategy applied in current approaches uses predefined knowledge in some forms, such as rules or probabilistic models, to guide alert correlation (e.g. [24][26][27]). It is time-consuming and error-prone to obtain the necessary knowledge because the process relies heavily on human involvement and is usually conducted in an ad-hoc manner. This issue, in fact, reflects the difficulties of acquiring and representing the knowledge for building meta intrusion detection systems. If case-based reasoning is applied, the entire process can be largely simplified; the need for knowledge acquisition can be limited to establishing how to characterize cases.

3.5.1 Problems and Cases for Meta Intrusion Detection

Cases and problems have to be clearly defined for a specific domain where CBR is applied. Our domain is meta intrusion detection. For this we define a problem as a potential attack and a case as a previously known attack. The conventional approach to characterize cases and problems is through a set of predefined features. In other words, a case or a problem is described by a set of features together with their values. This is basically the same as the attribute-value representation employed by other knowledge or learning systems.

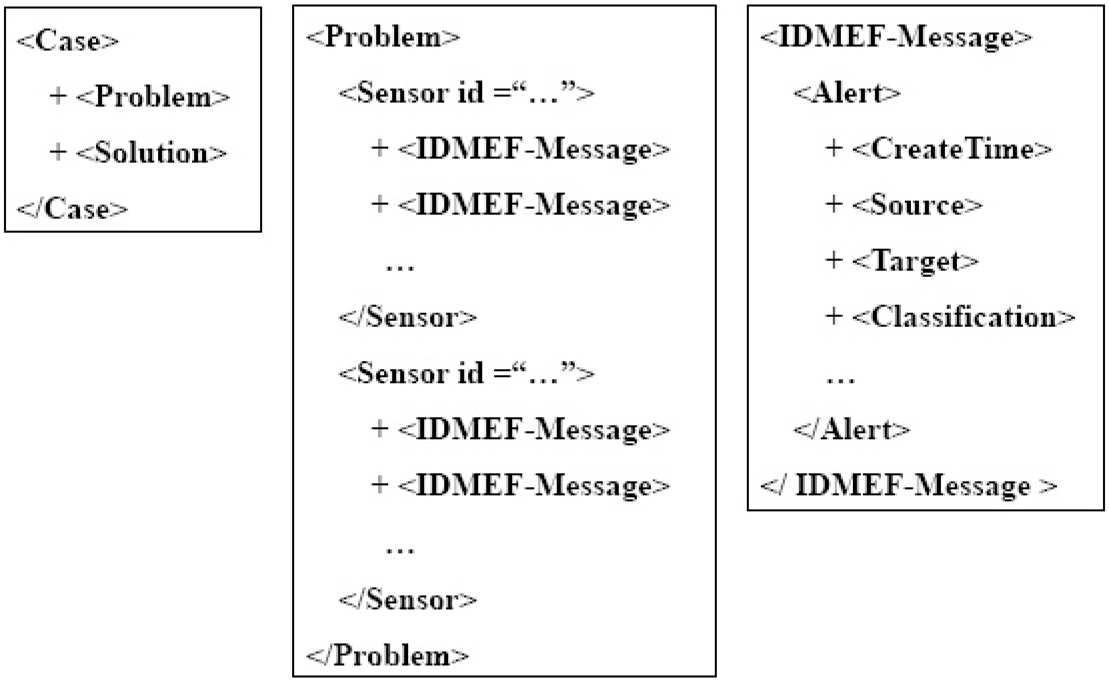
The information the meta IDS can obtain during an attack consists of IDMEF alerts from deployed sensors. Alert information is inherently heterogeneous. Some alerts are defined with very little information, such as origin, destination, name, and time of the event. Other alerts provide much more, information, such as ports or services, processes, user information, and so on. In addition, alert information may contain extended information due to the adoption of new detection approaches. Thus, predefining a fixed set of features that can cover all important information in alerts is almost impossible. Instead of doing so, we choose a more flexible approach to describe cases in our design. The collection of alerts generated from different sensors during the attack constitutes the description of an attack (or a problem). Since an alert is an XML object, in order to facilitate the aggregation of alerts, a problem is also represented in XML. The alerts comprising a problem are organized according to the sensors that produced them, and alerts from the same senor are sorted in chronological order. A case consists of the description of an attack and its solution. Figure 3.5 shows the representations and structures of a case, a problem, and an alert.

The XML representation of objects is inherently different from the attribute-value representation applied by conventional CBR systems. In an attribute-value representation, an object is described by a fixed number of attributes (attribute names and attribute values; attributes are called 'features' in CBR). Although the attribute-value representation is popular and used in a number of knowledge systems, it has a limitation when describing complex objects, such as trees. An XML document is a tree structure. If an XML object is transformed into a set of attributes (the process is called 'flattening a tree'), some information in it may be lost. Thus, in our approach, data analysis is performed directly over the alerts, given as XML objects, rather than working on a set of attributes (features) extracted from the original alerts.

3.5.2 The Framework Overview

Figure 3.6 presents the framework of a case-based meta intrusion detection system. It illustrates the relationships among different components and how they interoperate as a system.

A meta intrusion detection system has to work with a group of sensors strategically



```
<Case>
   + <Problem>
   + <Solution>
</Case>
```

```
<Problem>
   <Sensor id ="…">
      + <IDMEF-Message>
      + <IDMEF-Message>
       …
   </Sensor>
   <Sensor id ="…">
      + <IDMEF-Message>
      + <IDMEF-Message>
       …
   </Sensor>
</Problem>
```

```
<IDMEF-Message>
   <Alert>
      + <CreateTime>
      + <Source>
      + <Target>
      + <Classification>
      …
   </Alert>
</ IDMEF-Message >
```

Figure 3.5: The XML representations of a case, a problem and an alert

deployed at different locations within a certain information system. Each sensor is an independent IDS and performs data analysis over a certain data source and generates alerts when signs of intrusions are found. Those sensors do not have to be physically separated. They can be installed on the same device but watch over different data sources. For example, provided two IDSs, STIDE and Snort, are installed on the same machine, STIDE applies the sequence of system calls for intrusion detection, while the Snort uses network traffic to detect attacks.

Meta intrusion detection differs from traditional intrusion detection approaches in that it works on middle-level data, namely, the outputs from other IDSs working on raw audit data sources. Thus, the raw data sources, including training data, testing data, and runtime data sources, must be fed into the sensors first. Alerts from those sensors serve as the middle-level data and are the input for the meta IDS. Alerts are collected in a database of alerts. If the alerts in the databases are generated for training data, given the labeled information (attack information) in the training data, data mining (DM) or machine learning (ML) approaches can be applied to construct the knowledge base, including historical cases and background

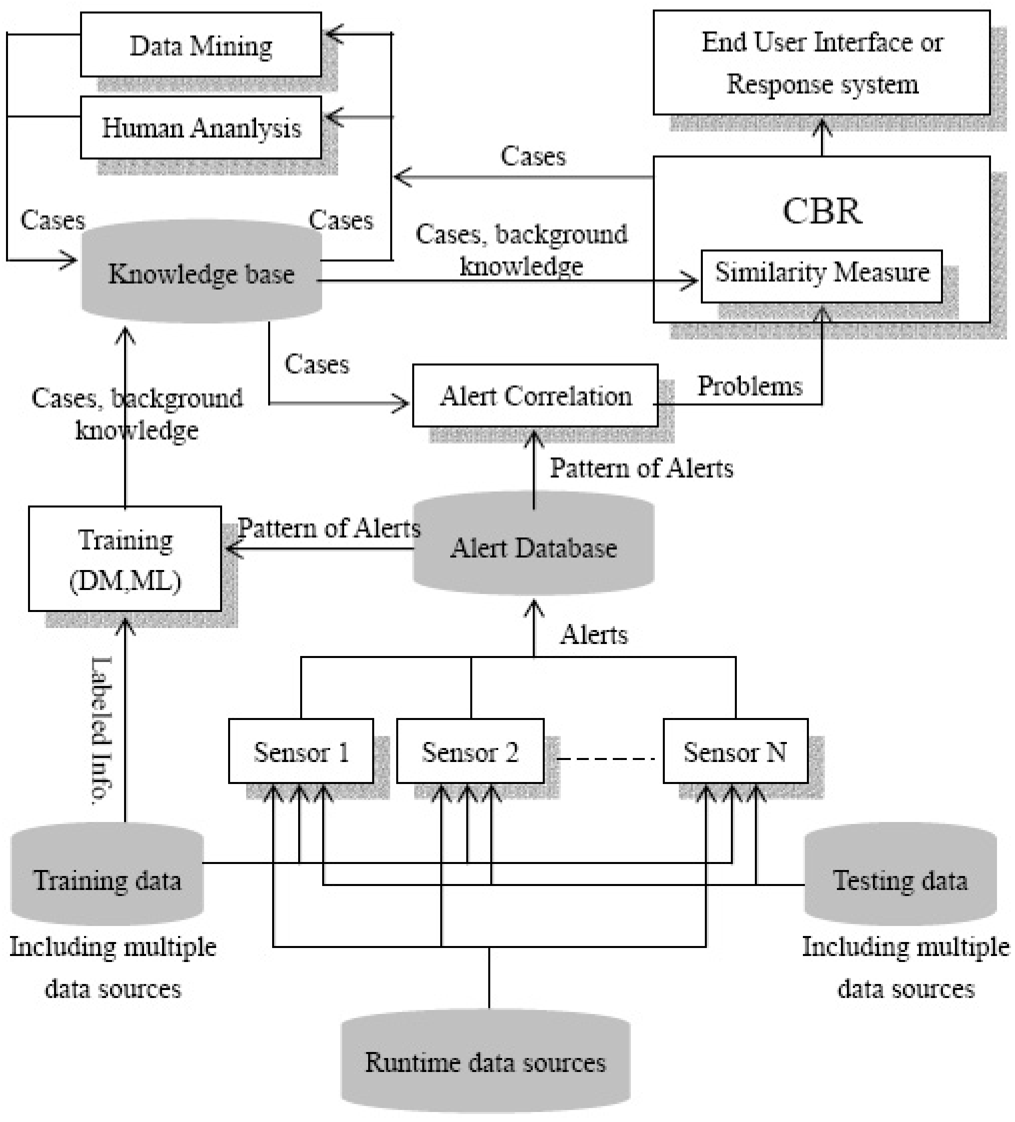


Figure 3.6: The Framework of Case-Based Meta Intrusion Detection System

knowledge, for intrusion detention. If the alerts in the alerts database are from testing data or runtime data sources, problems will be formulated out of them through an alert correlation approach that is concerned with determining which alerts are caused by the same attack. ‘Problem’ is a term used in case-based reasoning. In our application domain, it is a pattern of alerts representing a possible attack.

The knowledge base is critical for intrusion detection. It contains the known cases that the similarity measure will compare with problems and such background knowledge as the threshold value that separates normal and abnormal behaviors. In addition, the knowledge base may also assist alert correlation approaches in finding the most relevant alerts. Since the knowledge base may contain redundancy or missing information that would impact the detection performances, it normally needs to be refined through data mining or human analysis. It is always necessary and valid for sophisticated users to be able to manually update the knowledge base since some knowledge can be more easily obtained through human analysis rather than algorithms.

The final results from the CBR system go to the end users or the response system where the actions or countermeasures should be taken in response to detected attacks.

3.5.3 Main Issues of the Design

Although the design of the case-based meta intrusion detection system seems straightforward, it bears a few problems for implementation mainly due to the choice to use XML representations for cases and problems. Because of this, traditional similarity measures developed for data in attribute-value representations (data formatted in tables) will not work in our system. Moreover, most currently available data mining and machine learning algorithms only take data in attribute-value representations as input. If we want to take advantage of existing algorithms and similarity measures, our data in XML representations has to be transformed into data in tables as attribute-value representations. Our previous discussion, however, has revealed that doing this would cause loss of useful information in the alerts. Accordingly, a novel XML similarity was developed in order for our CBR approach to work with data in XML representations. New data mining and learning algorithms that directly perform analysis over objects in terms of XML representations also needed to be developed for the same reason.

Another issue is alert correlation, which is a common problem in meta intrusion detection research. Most present alert correlation approaches require expertise in certain forms, such as rules and probabilistic models, obtained in some manner or another. One of the distinguishing features of case-based reasoning is it simplifies the acquisition and representation of knowledge. The expertise of a CBR system is just a library of past cases rather than being encoded in traditional rules. But then, as such, cases do not provide rules or probabilistic models. So how to correlate alerts without this kind of knowledge is apparently not an easy task. As the effectiveness of meta intrusion detection, to some degree, relies on how well the problem of alert correlation is dealt with, an effective solution is necessary to make our approach be applicable.

CHAPTER 4

Data Mining

Data mining refers to extracting knowledge from large amounts of data. It is normally regarded as the evolution of another technology called KDD (knowledge discovery from databases). From [37],

> "KDD is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data".

According to this definition, data mining is a step in the KDD process concerned with applying algorithms to find patterns in the data. In a sense, data mining is the central step in the KDD process. The other steps in KDD are concerned with preparing data for data mining and evaluating the discovered patterns. Typical patterns take the form of trees, rules, or equations (formulas). Data mining is also looked upon as an application of machine learning because most algorithms in data mining actually come from the field of machine learning. Although data mining was first applied to traditional databases, the notion of database in data mining has a broader meaning and may be taken to include data repositories of almost any kind. In this paper we are concerned with data sources, such as tcpdump data and host audit data, which can be used for intrusion detection.

As discussed earlier, IDSs aim at detecting attacks against information systems. The objective is to raise alerts whenever an attack is in progress, has recently occurred, or is imminent. There are a number of concepts that can be used to classify IDSs. One distinction is between host-based and network-based IDSs, determined by the audit source location. Another is between anomaly detection and misuse detection, determined by the detection method. But, no mater what classification an IDS belongs to, one basic design principle is that it should search for attack evidence from various data sources. This process of searching for evidence can be based on the results of a learning process that strives to identify the defining characteristics of attacks. This learning process can be viewed as a form of data mining.

More exactly, data mining is good at removing redundant and noisy data from huge data sets, and these same techniques can be adapted to a core problem of IDS research, namely, that of reducing the numbers of false positives (i.e., false alerts) and false negatives (undetected attacks). This is accomplished by providing help for the development of precise detection models, learning rules from data, performing association analysis, and so on. Thus data mining can play a significant role in IDS.

Here it should be noted that all applications of data mining to intrusion detection to date have used data mining in the creation of different kinds of sensors. None so far have applied data mining at the higher level of a monitoring system such as what we have called meta intrusion detection.

4.1 Approaches

There are different approaches for data mining, which can be classified according to output and characterization of algorithms. Most approaches are conducted under a supervised learning framework. This means the input data is already classified or labeled already. Typical data mining approaches include:

- Association analysis: the discovery of association rules. Association rules specify correlations between frequently occurring item sets (items often found together in transactions).

- Classification (with prediction): the process of finding a model that describes and distinguishes data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. Common data mining methods of classification include back-propagation, neural networks, Bayes classification, and decision trees.

- Clustering: unlike in classification, where some classes are given and the goal is to place objects into those classes based on the objects' attributes, in clustering the goal is to identify some classes into which the objects may be grouped by clustering them according to their attributes so that objects within a cluster have high similarity in comparison to one other, but are very dissimilar to objects in other clusters.

- Outlier analysis: a database may contain data objects that don't comply with the general behavior or model of the data. These data objects are outliers. Most data mining methods discord outliers as noise or exceptions. However, in some applications such as intrusion detection or fraud detection, the rare events can be more interesting than the more regularly occurring ones. The analysis of outlier data is referred to as outlier mining.

4.2 Related Work on Data Mining in Intrusion Detection

There is a large amount of work concerned with applying data mining or machine learning to intrusion detection. Since most data mining algorithms come from machine learning, we only refer to data mining in general. Data mining has many approaches, such as association analysis, classification and clustering. They all have been explored for intrusion detection. In W. Lee's dissertation [38], he pointed out that association analysis can be applied to intrusion detection in particularly two ways: link analysis and sequence analysis. Link analysis helps find the correlations between system features that could be used as the basis for constructing usage profiles. Sequence analysis helps find sequential patterns that can be used to detect time-based attacks.

Viewing intrusion detection as a classification problem, various classification approaches have been applied to build an intrusion detection model. Lee and Xiang [39] proposed one information-theoretic measure, information gain, to measure the performance of using some features for classification. Valdes et al [40] have described eBayes, based on Bayesian classification. This has become a new component for the statistical anomaly detector of EMERALD[20]. Fox et al. [41] attempted modeling system and user behaviors using a kind of neural network known as a self-organizing map (SOM). This uses a type of unsupervised learning that can discover underlying structures of the data without prior examples of intrusive and non-intrusive activities. Ghosh et al. [42] used a back-propagation neural network to monitor a program running on a system. They improved the performance of detection by using randomly generated data as anomalous input. Lee et al. [43] proposed using a machine learning approach called RIPPER [44] to learn rules from training data, Esmaili et al. [36] used cased-based reasoning for intrusion detection, and Bridges et al. [45] involved fuzzy data mining and genetic algorithms in intrusion detection.

In [46], the authors presented a data-mining algorithm called the Clustering and Classification Algorithm-Supervised (CCA-S), which was developed specifically for detecting intrusions. The results showed it obtained a better detection performance than popular decision-tree algorithms. In the recognition that labeled data is not readily available in practice, the authors in [47] provided a method based on intersecting segments of unlabeled data and used the intersection as the base data for clustering. They used the results of clustering and performed outlier analysis for intrusion detection.

A recently emerging research field, multi-relational data mining (MRDM), is also explored for intrusion detection. MRDM is based on inductive logic programming (ILP) [48]. Ko [49] proposed the use of ILP to synthesize first-order logic formulas that describes the operations of a program that form its normal runs. Unlike the traditional data-mining algorithms that only take a flat table as input, multi-relational data mining find patterns in more than one table at the same time. The patterns are normally expressed in first-order logic. The process of finding first-order logic patterns is referred to as inductive logic programming. Multi-relational data mining has two advantages over traditional data mining. One is it avoids information loss and redundant data that occur in traditional data mining approaches. The other is it allows the use of complex domain-specific background knowledge (described as first-order rules) in the learning process to produce sound and consistent knowledge.

4.3 A Novel XML Distance Measure

Distance measures are used extensively in data mining and other types of data analysis. Such measures assume it is possible to compute for each pair of domain objects their mutual distance. Much of the research in distance measures concentrates on objects either in attribute-value representation or in first-order representation. With the increasing use of XML technology as a means for unambiguous exchange of information, more and more data come in the form of XML documents. We presented a distance measure between two objects in terms of their XML representation [50]. This measure views an XML object as a tree in which an XML element is a node. It recursively computes the overall distance between two XML trees from root nodes to leaf nodes. Accordingly, this measure can be applied in any domain as long as the object of that domain can be provided with a uniform XML representation.

Distance measures are used extensively in data mining and other types of data analysis. Dzeroski and Lavrac [51] surveys how distance measures can be applied in predictive learning and clustering; Wettschereck and Aha [52] discuss applying distance measures in cased-based reasoning. The central assumption of a distance measure is that it is possible, for a particular domain under consideration, to specify for each pair of objects their mutual distance (or similarity). Normally, measurements begin with objects (or instances) described in a certain representation language and then an appropriate algorithm is applied to obtain distances among those objects in terms of their representations.

The representation languages are typically classified as being either attribute-value or relational/first-order. In an attribute-value representation, the objects in a data set can be summarized in a table, where the columns represent attributes, and each row represents an object, with the cell entries in that row being the specific values for the corresponding attributes. In a first-order representation, an object is represented by a ground atom of a distinguished predicate symbol, where the position on the arguments represents the attributes, the arguments themselves represent the corresponding attribute's values, and these arguments are further defined by their occurrence in some additional set of ground atoms. Analysis of data in terms of a first-order representation is also referred to as multi-relational data mining, since the first-order representation provides a more powerful and reasonable way to describe objects than an attribute value representation.

For attribute-value representations where the attributes have only continuous numerical values, a Euclidean distance measure is normally applied. For objects having attributes of different types, methods for combining the attributes into a single similarity matrix were introduced by [53].

First-order representations, however, need more complicated distance measures. A typical first-order distance measure can be found in [54]. This was used in many well-known multi-relational algorithms, such as RDBC [55] and FORC [56]. Other first-order distance measures may be found in [57] and [58].

With the increased use of XML (Extensible Markup Language) as a means for unambiguous representation of data across platforms, more and more data is delivered in XML. When performing data analysis over such data, this is normally transformed into attribute-value or first-order representation, if distance measures are involved. Such transformation may result in information loss. The new representation may not contain all the contents or attributes of the original. The XML structure may not be preserved either.

In order to provide a more effective method for data analysis over data in XML representations, this paper presents a distance measure between two objects in terms of these representations. It looks upon a XML document as a collection of elements organized in a tree structure. For notational simplification, we transform XML documents into equivalent ones in which every element has no content and is denoted by its name, attribute set, and sub-element set; this is done by reconstructing the content of an element as an attribute of the element. The distance between any pair of elements is determined by their attribute sets and sub-element sets. This measure recursively computes the overall distance between two XML objects from root elements to leaf elements, looking for matchings for attribute sets and sub-element sets at each level. Once a set has more than one matching with the other, the well-known Hungarian algorithm [59] is applied to find the matching that yields the minimal overall distance. The distance between two XML objects is then given as this distance between their root elements. The details of this measure are laid out in the sections below.

This XML distance measure was developed initially for use with a hierarchical clustering algorithm to cluster XML documents representing patterns of alerts in the domain of intrusion detection. The results of that data mining experiment show that the measure is very effective for this purpose. The results are presented in chapter 6.

4.3.1 Related Work on Similarity of XML documents

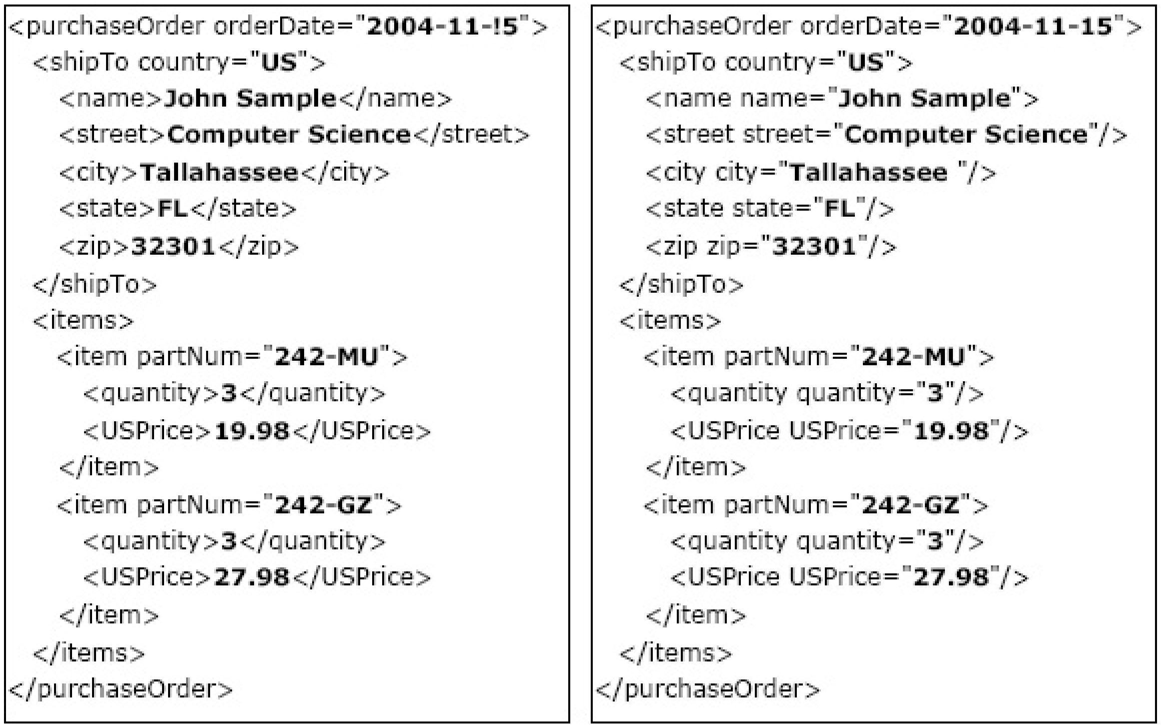
Most of the previous works regarding the similarity between two XML documents have concentrated on structural similarity in XML. Viewing XML documents as trees, Nierman and Jagadish [60] use the graph edit distance measure to compute the structural similarity between two XML documents. The algorithm for this distance measure was derived from one for the edit distance between strings [61]. Given a set of graph edit operations, such as deletion, insertion, and substitution, the edit distance is defined as the shortest sequence of edit operations that transform one tree into the other. In practice, a cost may be assigned to each individual operation to reflect its importance. Typical tree distance algorithms include [62] and [63]. Zhang et al. [64] review the edit distance between XML trees suitable for various applications. Flesca et al. [65] represent XMl documents as time series and compute the structural similarity between two documents by exploiting the Discrete Fourier Transform of the corresponding signals. Bertino et al. [66] worked on the structural similarity between an XML document and a DTD. Microsoft XML Diff (http://apps.gotdotnet.com/-xmltools/xmldiff/) is a tool that detects and shows the differences between two XML documents. Canfora et al. [67] have introduced an XML document similarity measure, based on a common sub-graph algorithm [68], for evaluating of the effectiveness of information extractive systems. Dopichaj [69] has suggested applying case-based reasoning (CBR) technology to integrate background knowledge for better similarity calculation in XML retrieval.

4.3.2 Representation of XML Documents

XML documents are composed of markup (tags) and content. The most basic component in an XML document is the XML element, consisting of some content surrounded by matching starting and ending tags. Elements may be nested within other elements to any depth. Because other components in an XML document, such as the prolog and any comments, are not used for representation of content, we assume they don't contribute to the overall distance and simply ignore them in our discussion. An example XML document is given below as XML-1. This document stores purchase information. It has one root element *purchaseOrder* that represents the contents as a whole. Other elements, such as *shipTo* and *items*, are nested within *purchaseOrder*. The element *shipTo* in turn has five directly nested elements, *name*, *street*, *city*, *state* and *zip*. These represent more detailed data than the containing element *shipTo*. Such nesting is common in XML documents and allows for hierarchical data structure representations. The graphical representation of an XML document is referred to as an XML tree.

An element in XML may have attributes. For example the element *shipTo* in XML-1 has an attribute *country* and it takes the value US. An attribute and its value in XML is a 2-tuple, $< a, v >$, where a is the attribute's name, and v is its value. Thus the attribute of element *shipTo* can be represented as $< country, US >$

In addition to attributes, elements can have nested elements or contents, but not both. More exactly, elements other than leaves in the XML tree representation do not have content,



```
<purchaseOrder orderDate="2004-11-!5">
  <shipTo country="US">
    <name>John Sample</name>
    <street>Computer Science</street>
    <city>Tallahassee</city>
    <state>FL</state>
    <zip>32301</zip>
  </shipTo>
  <items>
    <item partNum="242-MU">
      <quantity>3</quantity>
      <USPrice>19.98</USPrice>
    </item>
    <item partNum="242-GZ">
      <quantity>3</quantity>
      <USPrice>27.98</USPrice>
    </item>
  </items>
</purchaseOrder>
```

```
<purchaseOrder orderDate="2004-11-15">
  <shipTo country="US">
    <name name="John Sample">
    <street street="Computer Science"/>
    <city city="Tallahassee "/>
    <state state="FL"/>
    <zip zip="32301"/>
  </shipTo>
  <items>
    <item partNum="242-MU">
      <quantity quantity="3"/>
      <USPrice USPrice="19.98"/>
    </item>
    <item partNum="242-GZ">
      <quantity quantity="3"/>
      <USPrice USPrice="27.98"/>
    </item>
  </items>
</purchaseOrder>
```

Figure 4.1: Sample documents XML-1 (left) and XML-2 (right)

only other nested elements, whereas leaves have content only (which content may be empty).

For purposes of our distance measure, we wish to represent every element in the same form. To this end, we create a new attribute for each element that has content. The created attribute has the same name as the element and it takes the element content as the value of the new attribute. This results in an XML document that equivalent in terms of its data representation to the original, but which is entirely content-free. XML-2 is the new content-free document that results in this manner from XML-1.

In a content-free XML document, an element may be represented as a 3-tuple $< n, A, E >$, where n is the name of the element, $A = \{< a_1, v_1 >, < a_2, v_2 >, ..., < a_n, v_n >\}$, is the set of attributes of the element, and $E = \{e_1, e_2, ..., e_m\}$ is the set of elements nested within this element. Since a content-free XML document is just a collection of elements, it can be completely represented as a collection of 3-tuples of this form. For example, the following represents XML-2.

$e_{purchaseOrder} =< purchaseOrder, \{< orderDate, 2004-11-15 >\}, \{e_{shipTo}, e_{items}\} >$

$e_{shipTo} =< shipTo, \{< country, US >\}, \{e_{name}, e_{street}, e_{city}, e_{state}, e_{city}\} >$

$e_{name} =< name, \{< name, JohnSample >\}, \emptyset >$

$e_{street} =< street, \{< street, ComputerScience >\}, \emptyset\} >$

$e_{city} =< city, \{< city, Tallahassee >\}, \emptyset\} >$

$e_{state} =< state, \{< state, FL >\}, \emptyset\} >$

$e_{zip} =< zip, \{< zip, 32301 >\}, \emptyset\} >$

$e_{items} =< items, \emptyset, \{e_{item_1}, e_{item_2}\} >$

$e_{item_1} =< item, \{< partNum, 242 - MU >\}, \{e_{quantity_1}, e_{USPrice_1}\} >$

$e_{item_2} =< item, \{< partNum, 242 - GZ >\}, \{e_{quantity_2}, e_{USPrice_2}\} >$

$e_{quantity_1} =< quantity, \{< quantity, 3 >\}, \emptyset\} >$

$e_{quantity_2} =< quantity, \{< quantity, 3 >\}, \emptyset\} >$

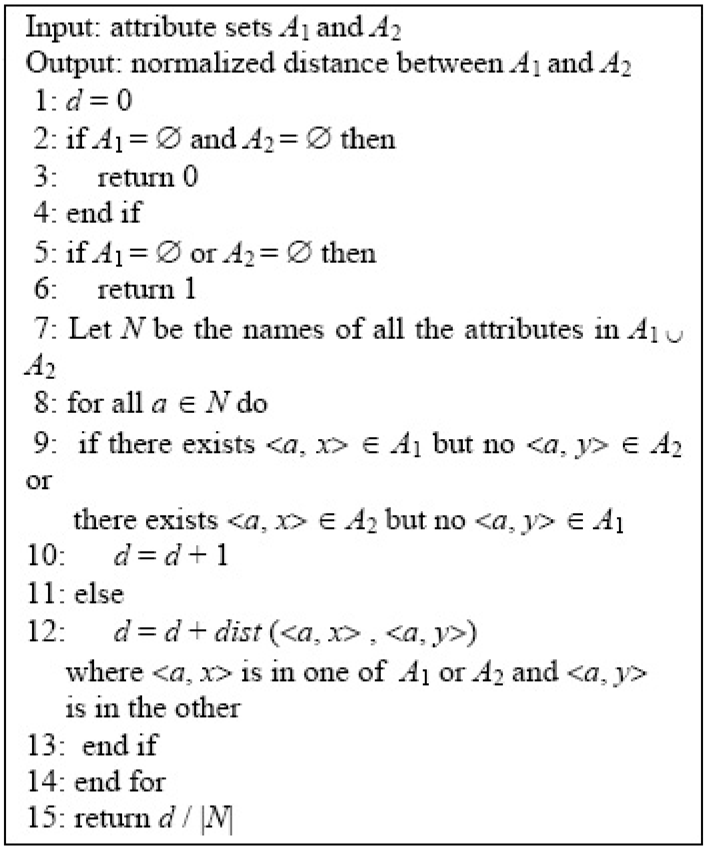
$e_{USPrice_1} =< USPrice, \{< USPrice, 19.98 >\}, \emptyset\} >$

$e_{USPrice_2} =< USPrice, \{< USPrice, 27.98 >\}, \emptyset\} >$

4.3.3 Distance Between Two Attribute Sets

For purposes of computing a distance between XML documents, a metadata file is created which states, for each attribute in the content-free representation of the XML document, whether, for the purpose of the distance calculation, the value is to be interpreted as numeric or should be retained as a string. For example, the values of attributes *quantity* and *USPrice* might be interpreted as numeric for purposes of determining the closeness of two prices, whereas value of attributes *orderDate* and *zip* might be retained as a string, since for purposes of the distance calculation it only matters whether two dates or zip codes are identical or not. Thus the range of values for an attribute can be of two general types, numeric and non-numeric. If numeric (whether continuous or discrete), it is given as an interval $[r_1, r_2]$.

Consider two attributes $\alpha_1 =< a_1, v_1 >$ and $\alpha_2 =< a_2, v_2 >$. We assume that two attributes having the same name will also have the same type of values. We define the distance $dist(\alpha_1, \alpha_2)$ as follows. If $a_1 \neq a_2$, i.e., the attributes have different names, then $dist(\alpha_1, \alpha_2) = 1$. If $a_1 = a_2$ and the values are non-numeric, then $dist(\alpha_1, \alpha_2) = 0$ if $v_1 = v_2$, and $dist(\alpha_1, \alpha_2) = 1$ if not. If $a_1 = a_2$ and the values are numeric with range $[r_1, r_2]$, then $dist(\alpha_1, \alpha_2) = |v_1 - v_2|/(r_1 - r_2)$. Based on this, given two attribute sets A_1 and A_2, we compute their distance according to the algorithm given in Figure 4.2. Briefly, for each attribute α in A_1 we determine its distance to the entire collection A_2 according to (i) if there is no attribute having the same name as in A_2, the distance is 1, and (ii) if there is



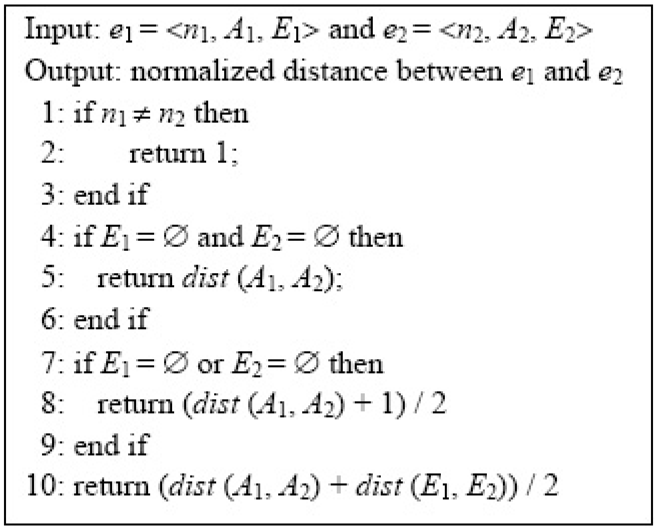
```
Input: attribute sets $A_1$ and $A_2$
Output: normalized distance between $A_1$ and $A_2$
 1: $d = 0$
 2: if $A_1 = \varnothing$ and $A_2 = \varnothing$ then
 3:     return 0
 4: end if
 5: if $A_1 = \varnothing$ or $A_2 = \varnothing$ then
 6:     return 1
 7: Let $N$ be the names of all the attributes in $A_1 \cup$
$A_2$
 8: for all $a \in N$ do
 9:   if there exists $<a, x> \in A_1$ but no $<a, y> \in A_2$
or
      there exists $<a, x> \in A_2$ but no $<a, y> \in A_1$
10:       $d = d + 1$
11: else
12:       $d = d + dist\ (<a, x>\ ,\ <a, y>)$
     where $<a, x>$ is in one of $A_1$ or $A_2$ and $<a, y>$
     is in the other
13:   end if
14: end for
15: return $d\ /\ |N|$
```

Figure 4.2: Algorithm 1: distance between two attribute sets.

an attribute α' in A_2 having the same name as α, the distance is $dist(\alpha, \alpha')$. Similarly we determine the distance from each attribute in A_2 to the collection A_1. Then we add together all these individual distances and normalize by the total number of distinct attribute names.

4.3.4 Distance between Two Elements and Two Element Sets

The distance between two elements is determined by their attributes and nested subelements. The algorithm for determining the distance between two elements thus requires determining the distance between their two sets of subelements. In turn, the algorithm for determining the distance between two sets of elements requires determining the distance between two elements. Thus these two algorithms must call each other recursively. The algorithm for determining the distance between two elements e_1 and e_2 is given in Figure 4.3. An XML element can have at most one attribute of the same type, but can have multiple subelements

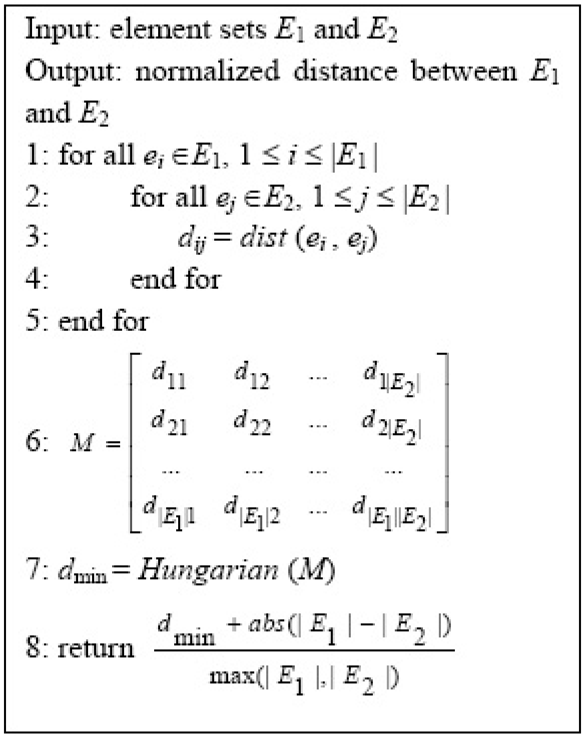


```
Input: $e_1 = <n_1, A_1, E_1>$ and $e_2 = <n_2, A_2, E_2>$
Output: normalized distance between $e_1$ and $e_2$
 1: if $n_1 \neq n_2$ then
 2:       return 1;
 3: end if
 4: if $E_1 = \varnothing$ and $E_2 = \varnothing$ then
 5:    return $dist(A_1, A_2)$;
 6: end if
 7: if $E_1 = \varnothing$ or $E_2 = \varnothing$ then
 8:    return $(dist(A_1, A_2) + 1) / 2$
 9: end if
10: return $(dist(A_1, A_2) + dist(E_1, E_2)) / 2$
```

Figure 4.3: Algorithm 2: distance between two elements.

of the same type. For example, in XML-1, the subelements of items are two elements of type *item*. Given two element sets, as in this case, it is possible that one will have more instances to the same element type than the other. Suppose we know the distance between any two elements (as determined by the foregoing algorithm). The problem of determining the distance between two element sets can be transformed into a maximal matching problem analogous to the classical problem of assigning m workers to n jobs, where each worker has a possibly different cost to finish each of the n jobs. The objective is to find an assignment with minimal overall cost. This is known to be achievable by the Hungarian algorithm [59]. Here the members of one element set play the role of workers, the members of the other element set play the role of jobs, and the distances between members play the role of costs.

Given m workers and n jobs, the Hungarian algorithm is applied to the $m \times n$ matrix representing the costs for each worker-job pair, and it assigns at most one worker to each job and at most one job to each worker. Thus if there are more workers than jobs, some workers will be unemployed, and if there are a more jobs than workers, some jobs will not get done. For the purposes of our distance measure, it is desired that any such unmatched elements contribute also to the overall distance between the two element sets. To this end, we add some 'virtual' elements to the smaller set, so that both sets have the same size, and for each such virtual element, we let its distance to each element in the opposite set be 1. Thus, if



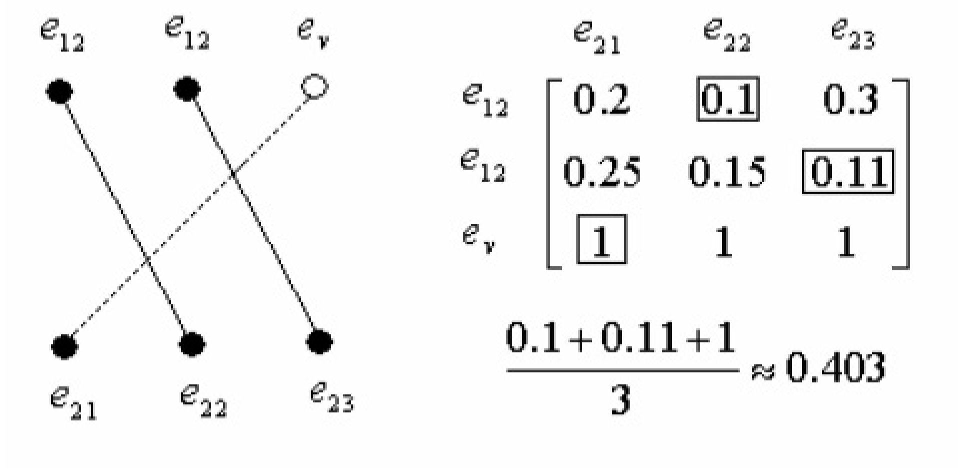
```
Input: element sets $E_1$ and $E_2$
Output: normalized distance between $E_1$ and $E_2$
1: for all $e_i \in E_1$, $1 \le i \le |E_1|$
2:         for all $e_j \in E_2$, $1 \le j \le |E_2|$
3:                 $d_{ij} = dist\ (e_i\ ,\ e_j)$
4:         end for
5: end for
6: $M = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1|E_2|} \\ d_{21} & d_{22} & \cdots & d_{2|E_2|} \\ \cdots & \cdots & \cdots & \cdots \\ d_{|E_1|1} & d_{|E_1|2} & \cdots & d_{|E_1||E_2|} \end{bmatrix}$
7: $d_{\min} = Hungarian\ (M)$
8: return $\dfrac{d_{\min} + abs(|E_1| - |E_2|)}{\max(|E_1|, |E_2|)}$
```

Figure 4.4: A distance matrix and its maximal matching with a virtual element

the original $m \times n$ matrix is M, and if $m > n$, the resulting matrix M' will be $m \times m$ and have the $m - n$ additional rows filled with 1's. The distance between the two element sets is then defined as $Hungarian(M)/m$.

If m is much larger than n, then M' will be much larger than M, and applying the Hungarian algorithm to the former will incur a much greater cost. It turns out, however, that under the above assumptions $Hungarian(M')$ can be computed more simply as $Hungarian(M) + m - n$. This is because adding a virtual element to one of the sets, and defining its distance to the elements in the other sets to be 1, means that, however that virtual element is matched with an element from the other set, this always adds exactly 1 to the overall cost.

To illustrate, consider two element sets $E = \{e_1, e_2\}$ and $E = \{e'_1, e'_2, e'_3\}$, with their distance matrix as given in Figure 4.5. The Hungarian algorithm matches e_1 with e'_2 and e_2 with e'_3 yielding the minimal sum of distances as $0.10 + 0.11$. The unmatched e'_1, is then matched with a virtual element, as shown on the right side of Figure 4.5, and its



$$\begin{array}{cccc} & e_{21} & e_{22} & e_{23} \\ e_{12} & 0.2 & \boxed{0.1} & 0.3 \\ e_{12} & 0.25 & 0.15 & \boxed{0.11} \\ e_v & \boxed{1} & 1 & 1 \end{array}$$

$$\frac{0.1+0.11+1}{3} \approx 0.403$$

Figure 4.5: Algorithm 3: distance between two element sets

distance to the virtual element is given as 1. The distance between these two sets is thus $dist(E, E) = (0.10+0.11+1)/3 = 0.40$. A general algorithm for finding the distance between two element sets is given in Figure 4.4. Here *abs* denotes the absolute value.

4.3.5 Distance between Two XML Objects

The distance between two XML objects is formulated as the distance between their two root elements. In order to show how to compute this distance, XML-3 (Figure 4.6) is given to compare with XML-1. These documents have the same type of root element, *purchaseOrder*, but differ in content. We represent XML-3 as a set of 3-tuples as follows.

$e'_{purchaseOrder} =< purchaseOrder, \{< orderDate, 2002-10-20 >\}, \{e'_{shipTo}, e'_{items}\} >$

$e'_{shipTo} =< shipTo, \{< country, US\}, \{e'_{name}, e'_{street}, e'_{city}, e'_{state}, e'_{zip}\} >$

$e'_{name} =< name, \{< name, JohnDoe >\}, \emptyset\} >$

$e'_{street} =< street, \{< street, ComputerScience >\}, \emptyset >$

$e'_{city} =< city, \{< city, Tallahassee >\}, \emptyset\} >$

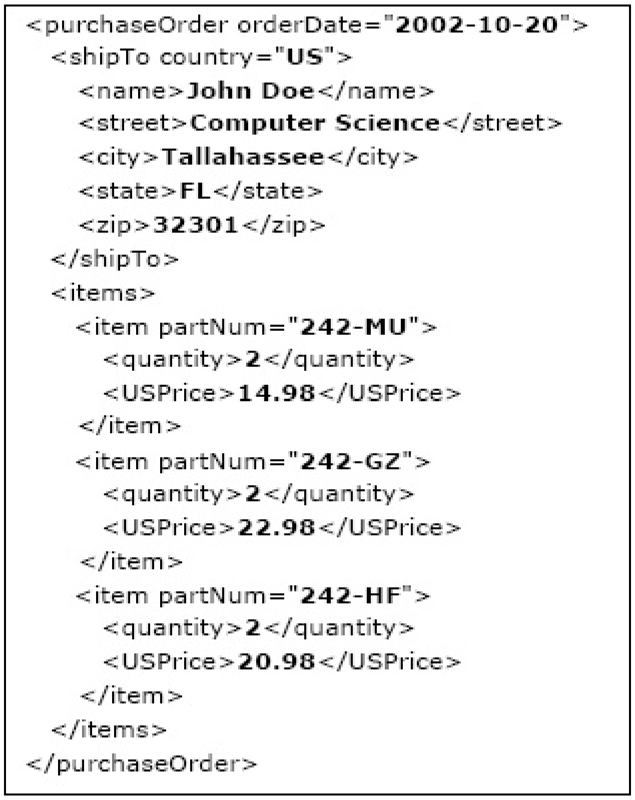
$e'_{state} =< state, \{< state, FL >\}, \emptyset\} >$

$e'_{zip} =< zip, \{< zip, 32301 >\}, \emptyset\} >$

$e'_{items} =< items, \emptyset, \{e'_{item_1}, e'_{item_2}, e'_{item_3}\} >$

$e'_{item_1} =< item, \{< partNum, 242-MU >\}, \{e'_{quantity_1}, e'_{USPrice_1}\} >$

$e'_{item_2} =< item, \{< partNum, 242-GZ >\}, \{e'_{quantity_2}, e'_{USPrice_2}\} >$



```
<purchaseOrder orderDate="2002-10-20">
  <shipTo country="US">
    <name>John Doe</name>
    <street>Computer Science</street>
    <city>Tallahassee</city>
    <state>FL</state>
    <zip>32301</zip>
  </shipTo>
  <items>
    <item partNum="242-MU">
      <quantity>2</quantity>
      <USPrice>14.98</USPrice>
    </item>
    <item partNum="242-GZ">
      <quantity>2</quantity>
      <USPrice>22.98</USPrice>
    </item>
    <item partNum="242-HF">
      <quantity>2</quantity>
      <USPrice>20.98</USPrice>
    </item>
  </items>
</purchaseOrder>
```

Figure 4.6: XML-3

$e'_{item_3} =< item, \{< partNum, 242 - HF >\}, \{e'_{quantity_3}, e'_{USPrice_3}\} >$
$e'_{quantity_1} =< quantity, \{< quantity, 2 >\}, \emptyset >$
$e'_{quantity_2} =< quantity, \{< quantity, 2 >\}, \emptyset >$
$e'_{quantity_2} =< quantity, \{< quantity, 2 >\}, \emptyset >$
$e'_{USPrice_1} =< USPrice, \{< USPrice, 19.98 >\}, \emptyset >$
$e'_{USPrice_2} =< USPrice, \{< USPrice, 22.98 >\}, \emptyset >$
$e'_{USPrice_3} =< USPrice, \{< USPrice, 20.98 >\}, \emptyset >$

Suppose that the attribute *quantity* is of type numerical with range $[1, 10]$, the attribute *USPrice* is numerical with range $[0, 100]$, and all other attributes are non-numerical. The distance between XML-1 and XML-3 is given as $dist(e_{purchaseOrder}, e'_{purchaseOrder})$. Because $e_{purchaseOrder}$ and $e'_{purchaseOrder}$ have attributes with the same name, i.e., *orderDate*, but these attributes take different values, the distance between their attribute sets is 1, and so, according to Algorithm 3,

$$dist(e_{purchaseOrder}, e'_{purchaseOrder}) = (1 + dist(e_{shipTo}, e_{items}, e'_{shipTo}, e'_{items}))/2 \tag{4.1}$$

The distance matrix for the two element sets e_{shipTo}, e_{items} and e'_{shipTo}, e'_{items} is comprised of the four distances $dist(e_{shipTo}, e'_{shipTo})$, $dist(e_{shipTo}, e'_{items})$, $dist(e_{items}, e'_{shipTo})$, and $dist(e_{items}, e'_{items})$. For the first of these, since e_{shipTo} and e'_{shipTo} have identical attributes, the distance between their attribute sets is 0, and since they differ on only one of their five sub-elements, the distance between their element sets is $1/5(= 0.2)$. Hence, by Algorithm 3,

$$dist(e_{shipTo}, e'_{shipTo}) = (0 + 0.2)/2 = 0.1 \tag{4.2}$$

The second and third of are given by Algorithm 3 simply as

$$dist(e_{shipTo}, e'_{items}) = 1 \tag{4.3}$$

$$dist(e_{items}, e'_{shipTo}) = 1 \tag{4.4}$$

For the fourth, Algorithm 3 says that we need to compute

$$dist(e_{items}, e'_{items}) = (0 + dist(e_{item_1}, e_{item2}, e_{item_1}, e'_{item_2}, e'_{item_3}))/2 \tag{4.5}$$

This requires that we apply Algorithm 2 to compute the distance between the element sets e_{item_1}, e_{item_2} and $e'_{item_1}, e'_{item_2}, e'_{item_3}$, which in turn requires computing a 2 by 3 matrix representing the pairwise distances between their elements. To illustrate, again by Algorithm 3, $dist(e_{item_1}, e'_{item_1}) = (0 + dist(e_{quantity_1}, e_{USPrice_1}, e'_{quantity_1}, e'_{USPrice_1}))/2$. Since the type of attribute quantity is numerical with range $[1, 10]$, and the type of attribute *USPrice* is numerical with range $[0, 100]$, $e_{quantity_1}$ has distance $(3 - 2)/10 = 0.1$ from $e'_{quantity_1}$ and distance 1 from $e'_{USPrice_1}$, $e_{USPrice_1}$ has distance 1 from $e'_{quantity_1}$ and distance $(19.98 - 15.98)/100 = 0.05$ from $e'_{USPrice_1}$. Thus, $dist(e_{item_1}, e'_{item_1}) = (0+(0.1+0.05)/2)/2 = 0.0375$. The distance between the other pairs can be obtained similarly. This gives the distance matrix for the two element sets as

$$\begin{bmatrix} 0.0375 & 0.6 & 0.555 \\ 0.59 & 0.0375 & 0.585 \end{bmatrix}$$

From this, Algorithm 2 gives the distance between the two element sets as

$$dist(e_{item_1}, e_{item_2}, e'_{item_1}, e'_{item_2}, e'_{item_3}) = (0.0375 + 0.0375 + 1)/3 = 0.583 \tag{4.6}$$

Thus, from 4.5 and 4.6,

$$dist(e_{items}, e'_{items}) = (0 + 0.583)/2 = 0.292 \tag{4.7}$$

Then, from 4.2, 4.3, 4.4, and 4.7, the distance matrix for e_{shipTo}, e_{items} and e'_{shipTo}, e'_{items} is

$$\begin{bmatrix} 0.1 & 1 \\ 1 & 0.292 \end{bmatrix}$$

Finally, 4.1 yields the overall distance

$dist(e_{purchaseOrder}, e'_{purchaseOrder}) = (1 + (0.1 + 0.292)/2)/2 = 0.675$

4.4 A Supervised Clustering Algorithm

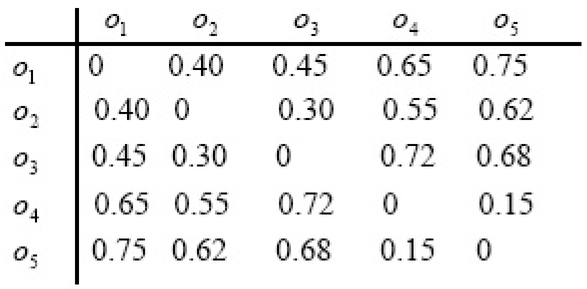
Supervised clustering is a novel data mining technique. It differs from traditional clustering in that the examples to be clustered are already classified. The goal of supervised clustering is to identify clusters that have high probability density with respect to a single class.

Finding clustering that is guaranteed to be optimal in terms of a chosen quality measure is often a difficult task, since it requires an exhaustive search of all possible groupings. Hence, distance-based clustering algorithms normally use heuristic strategies, of which there are several varieties, e.g., bottom-up agglomerative clustering and k-means clustering [13]. We have adopted the former approach, since it is relatively easy to implement and is known to be quite effective. Because objects to be clustered in our domain are XML documents, our clustering algorithm applies the XML distance measure introduced in [5] as the underlying similarity measure. Our supervised algorithm is actually a variation of a traditional clustering algorithm. In general, the result of applying a clustering algorithm to a collection of objects produces a collection C of clusters c. The objects in the clusters may be of different types, depending on their identifying features. In any given cluster, the *majority type* is the one that has the most objects. If there are two types with the same largest number of objects, we simply choose one of them as the majority type. All other types of objects in the cluster are *minority types*. A *minority object* is one that belongs to a minority type.

Given this terminology, we can define a fitness function to evaluate the quality of our clustering results C, $quality(C) = |C| + \sum_{c \in C} minority(c)$, where $minority(c)$ denotes the number of minority objects in a cluster c. Lower values for $q(C)$ indicate a better solution. The objective of the clustering algorithm is thus to find a clustering with the lowest value of $q(C)$. The construction of the hierarchical clustering proceeds in a bottom-up manner. It starts with each object forming its own cluster. Two clusters that are closest to each other are merged into a new cluster. This same procedure is repeated until finally all objects are in the same cluster. The algorithm needs to keep track of the value of the fitness function and its corresponding clustering in each iteration, and it returns the clustering with the lowest value of fitness function as the final solution.

From the description above, the clustering algorithm has to measure the distance between two clusters in each iteration in order to find the closest pair. In our implementation, the distance between two clusters is defined as the maximal distance between an object in one cluster and an object in the other. More specifically, where $C = o_1, o_2, \ldots, o_n$, $C' = o'_1, o'_2, \ldots, o'_m$ are two clusters of objects, $dist(C, C') = max\{dist(o_i, o'_j), 1 \leq i \leq n, 1 \leq j \leq m\}$. It should be noted that the distance between clusters in the beginning is simply the distance between two objects, as each object forms its own cluster. In general there are various ways of defining the distance between two clusters. For example, in addition to the maximal distance discussed above, it can be defined as the minimal distance, $dist(C, C') = min\{dist(o_i, o'_j), 1 \leq i \leq n, 1 \leq j \leq m\}$, or it could be defined as the averaged sum of distances, $dist(C, C') = \frac{\sum_i \sum_j dist(o_i, o_j)}{|C| \times |C'|}$. The maximal distance does not encourage combining two clusters since it tries to enlarge the distance between two clusters. Minimal distance does the opposite of maximal distance and the averaged sum of distances is intermediate between the maximal distance and minimal distance. We choose the maximal distance because it avoids making minority objects. According to the defined fitness function, the cost of combining two clusters is # increased minority objects - 1 (# clusters decreased by 1). Minority objects contribute most to the cost. We hope each combination of two clusters can decrease the value of fitness function. It is obvious that a good combination can decrease fitness function at most by one. However, a poor one can make all objects in one cluster become minority objects with regard to the objects in the other cluster and, moreover, largely increase the value of fitness function. Hence, maximal distance generally plays with less risk than minimal distance or averaged sum of distances according to the defined fitness function.

An example is given here to illustrate how it works. Suppose we start with five objects



	o_1	o_2	o_3	o_4	o_5
o_1	0	0.40	0.45	0.65	0.75
o_2	0.40	0	0.30	0.55	0.62
o_3	0.45	0.30	0	0.72	0.68
o_4	0.65	0.55	0.72	0	0.15
o_5	0.75	0.62	0.68	0.15	0

Figure 4.7: A Sample Distance Matrix

$o_1, o_2, \ldots, o_5$, where o_1,o_2,and o_3 belong to type A; o_4 and o_5 belong to type B; their distance matrix is given in Figure 4.7.

Table 4.1: Overall results of the first experiment on DARPA 1998 data

Clustering Results	MIN distance between clusters	$q(C)$
$C_1 = \{(o_1), (o_2), (o_3), (o_4), (o_5)\}$	0.15	5
$C_2 = \{(o_1), (o_2), (o_3), (o_4, o_5)\}$	0.30	4
$C_3 = \{(o_1), (o_2, o_3), (o_4, o_5)\}$	0.45	3
$C_4 = \{(o_1, o_2, o_3), (o_4, o_5)\}^*$	0.75	2
$C_5 = \{(o_1, o_2, o_3, o_4, o_5)\}$	-	3

4.5 Evaluation of Our Data Mining Techniques with Snort

Snort is well known for triggering large numbers of false alerts [70]. In addition, it fires alerts on a packet basis and misses the context of attacks, which makes it hard for users to identify the attacks and even harder when real alerts are mixed with false alerts. In this section we present a method for applying our data mining techniques to handle Snort alerts more effectively. Central to this approach is the representation of alerts using the Intrusion Detection Message Exchange Forma (IDMEF), which is defined in XML. All the alerts for each network session are assembled into a single XML document, thereby representing a pattern of alerts. Our experiments with the MIT 1998 DARPA data sets using our data mining techniques demonstrate that the XML distance measure can find a reasonable distance for two XML documents and the clustering algorithm based on the XML distance measure can well distinguish various types of patterns of alerts rising from different normal sessions as well as different attack sessions. The results of the clustering provide useful information that can be further applied to reduce the number of false positives and identify attacks on a session basis.

4.5.1 Related Work

Many approaches have been taken toward improving Snort. Wu and Chen [71] proposed a framework for Snort to make it able to detect sequential attack behaviors. That is, whereas Snort looks at only one packet at a time, that work tries to identify sequences of packets that represent attacks and to augment Snort with rules that fire when such sequences are detected. To this end, they apply data mining techniques to extract attack signatures and convert them into Snort rules. The Snort detection engine is then accompanied by their intrusion behavior detection engine, which triggers an alert when a series of incoming packets matches the signatures representing sequential intrusion scenarios.

Coit et al. [72], Sourdis et al. [73], Liu et al. [74], and Yu et al. [75] apply pattern-matching techniques to improve the detection speed of Snort. The issue of detection accuracy, however, was not addressed in any of these works. Efforts to deal with this issue have nonetheless been explored, using artificial intelligence techniques. In particular, Chavan et al. [76] use neural networks and a fuzzy inference system, and Schwartz et al. [34] briefly discuss the possibility of improving the efficacy of Snort through shifting from a rule-based to a case-based reasoning system. The present paper is an outgrowth from the latter work.

Clustering intrusion detection alerts has also been proved successful in other studies. Cuppens and Miege [7][13] apply this in a multi-sensor intrusion detection environment, where the sensors are IDSs of various types and may be either host-based or network-based. The clustering is based on a distance measure studied by Valdes and Skinner [10]. This multi-sensor IDS tries to group alerts from the various sensors so that the alerts in any one group all pertain to the same attack. In this way, it offers a more accurate view of the system's status and, in particular, makes it easier to distinguish false alerts from real ones. Inasmuch as this operates in real-time, it can be viewed as a real-time clustering system. Our approach makes the assumption that all alerts in a particular network session pertain to the same attack. Thus, alert correlation is accomplished simply by looking for identical network session identifiers, source IP and port, target IP and port.

Julisch [23][77] applied clustering to intrusion detection alerts to support root cause analysis. Root causes may be of many different types. Two examples are malfunctioning software and normal behavior, both of which can cause false alerts. That work was motivated by the observation that 90% of all alerts stem from a few dozen root causes. A novel alert-clustering algorithm was proposed to support a human analyst in identifying such root causes. Alerts in the same cluster tend to have the same cause. Thus, if a cluster represents a set of false alerts, the frequency of such alerts can be reduced if the cause can be removed.

Ning et al. [26][78] have introduced a technique that constructs attack scenarios by correlating alerts on the basis of prerequisites and consequences of attacks. The attack scenario is represented by a hyper-alerts alert correlation graph. In order to facilitate the investigation of large sets of alerts, several interactive analysis utilities were introduced. Clustering analysis, one of those utilities, is applied to a set of hyper-alerts correlation graphs or a hyper-alerts correlation graph (as graph decomposition) to find the common features. The present research uses clustering for still another purpose. Here we focus on network sessions, i.e., the entire collection of packets that occur in each such session, and aim to identify patterns of alerts that are associated with real attacks, as well as patterns that characterize false alerts raised during normal sessions. Our style of clustering, as well as the underlying similarity measure, is also different from those employed in the previous efforts.

4.5.2 Experimental Results

Our work was based on the 1998 MIT DARPA data sets. More specifically, we used the network training data sets, which contain tcp dump files for each weekday over a 7-week period, for a total of 35 days. The tcp dump files contain all packets that entered or left the network during the given day. In addition, we used the associated tcp dump "list" files, which, for each day, summarize all the network sessions represented by the tcp traffic during that day. For each such session, this gives the source IP address and port, the destination IP address and port, the session's start time, the session's duration, a flag indicating whether the session contains a real attack, and, if it does, the name of the attack. The sessions throughout each day are enumerated in chronological order, with this number serving as a session ID.

The tcp dump data was fed through Snort and all the alerts were captured. Also as mentioned, we used a Snort output plug-in to convert the alerts into IDMEF. The details of this format, to the extent that it has been employed in the present experiments, are given in Chapter 6.

The XML representations of the alerts were then loaded into a database using XML-DBMS (http://www.rpbourret.com/xmldbms/). This is a middleware program that employs a special language for mapping the elements of an XML document into tables in a relational database. The program uses this mapping to automatically read the XML documents and insert their components into their associated database tables. It can also work in reverse, to extract information from the database and output it in the original XML format.

Given this database, the alerts associated with each network session were extracted and put into a file. This was accomplished by going through the tcp dump list files and, for each network session, executing an SQL select statement based on the session's source IP address, source port, destination IP address, destination port, and the session's duration. The resulting XML document thus represents the entire collection of alerts that were raised during the given session, listed in chronological order. In addition, when each such document was created, it was given a file name that contains the session ID, the information whether the session was normal or contained a real attack, and, in the case of the latter, the name of the attack. This made it easy to group together all the files for sessions containing the same kind of attack, as well as to distinguish those files containing false alerts (i.e., representing normal sessions).

In total, there were 21 different kinds of attacks detected by Snort. For the most part, these all occurred in either telnet, http, of ftp sessions. In addition, there were many of these same types of sessions that were normal, but for which Snort generated false alerts.

From the database, session alert files were extracted for a maximum of 100 sessions for each kind of attack and, as well, for each of the 3 kinds of normal sessions having false alerts. The exact counts for each of these are shown in the last row of Table 4.2. For example, for attack type 'anomaly' there were only 9 sessions containing this attack, and for attack type 'back' there were 100 or more. In total, 749 such session alert files were selected.

This collection of XML files was then sent to our clustering program. It yielded 45 clusters as shown in Table 4.2 with a minimal distance of 0.3 among clusters when the fitness function is optimal. Altogether, 45 clusters were generated. The 35 clusters having more than one object (XML document) are shown in rows 0 through 34. The clusters having only one object each are counted in the Singletons as shown in the second row to last in Table 4.2. A cluster is regarded as pure if all objects in it are of the same type; otherwise, it is impure. As shown in Table 4.2, clusters 14, 31, and 34 are impure, while all others are pure.

The clustering algorithm shows excellent performance for grouping certain types of sessions. For example, all 100 'back' attack sessions go to pure cluster 32. This indicates that our clustering algorithm can perfectly distinguish this type from other types. There are five other such types that have been perfectly clustered, namely, 'format_clear' (1 session), 'ffb' (5 sessions), 'ffb_clear' (1 session), 'land' (17 sessions), and 'phf' (2 sessions). Since the size of most of these clusters is small, there is limited confidence that these same results will be obtained with larger tests. Nonetheless, the present results are promising inasmuch as the clustering is pure and these attacks are clearly distinguished from other types.

The clustering algorithm also demonstrates almost perfect performance for grouping other session types. In particular, 'ftp_write' has its 4 sessions distributed over 2 pure clusters having 2 sessions each. In this case, further analysis shows that this may be attributed to the types of the sessions. As shown in Table 4.3, one cluster results from ftp sessions, and the other from login sessions. Similarly, the 100 'nmap' sessions are distributed over 2 pure clusters (7 and 8). There does not appear to be any obvious reason in this case for having 2 clusters. The same applies to 'normal http', which has its 100 sessions distributed over pure 15 pure clusters.

Moreover, that algorithm shows good performance in several cases where the clustering is impure. There are two variations. First is the situation where most of the sessions go into pure clusters, and a few go into impure clusters. These are 'guest' (49 out of 50, giving 98% correctly clustered), 'multihop' (2 out of 3, giving 67% correctly clustered), 'satan' (35 out of 40, giving 88% correctly clustered), and 'normal ftp' (97 out 100, giving 97% correctly clustered). In these cases, the percentage gives the probability that a session of the given type will be correctly identified by the clustering algorithm. For the 'satan' attacks, the distribution of sessions over several clusters can be explained similarly as for 'ftp_write'. This is summarized in Table 4.4, which shows the various services with which the clusters are associated. Here it may be noted that the clustering algorithm effectively identifies the four named services (eco-i, etc.). The services grouped under 'others' and put into Cluster 29 are unnamed services running on infrequently used ports.

The second variation for impure sessions is the situation where all, or most, of the sessions are in impure clusters, but the relative numbers of other types of sessions in those same clusters is small. These are ‘warezclient’ (100 sessions in a cluster of 104, giving 96% accuracy) and ‘normal telnet’ (93 sessions in a cluster of 118, giving an accuracy of 79%). In these cases, the percentage gives the probability that a session in the given cluster will be of the given session type. It is noteworthy also that all the ‘warez’ variety of sessions (warez, warezmaster, warezclient) went into the same cluster (giving 98% accuracy for sessions of this variety).

4.5.3 Discussion

Three points are worthy of mention relative to this research. First is that, the results of the supervised clustering algorithm suggest a few approaches to improve the detection performance for Snort. As the algorithm identifies clusters that have high probability density with respect to a single class, some large clusters, such as clusters 31, 23, 11 and 16 in Table 4.2, containing similar patterns of alerts frequently raised in normal sessions. Those patterns turn out to be raised by a small set of Snort rules. Hence, just as Julisch has proposed in [77][23], we can change the snort configuration by blocking those rules to avoid extensive false positives. However, blocking rules may cause some attacks to be undetectable by Snort. Instead of doing that, we can apply a higher-level decision making method, such as case-based reasoning, to record the representative patterns of false alerts in normal sessions and compare them with runtime patterns of alerts. If they are sufficiently similar to each other, the new patterns can be regarded as representing possible false alerts and thus be ignorable. Similarly, representative patterns of alerts can also be extracted from clusters representing real attacks. Because different types of attacks were well distinguished from each other with regard to their patterns of alerts, their representatives can be used to identify those types of attacks in runtime alerts. As patterns of alerts were constructed for sessions, using them basically extends Snort’s ability to detect attacks on a session basis.

A second point is that our study makes use of the 1998 DARPA data sets created at MIT’s Lincoln Lab. Here it should be mentioned that this data has the limitations (i) the total number of real attacks for which Snort generates alerts is rather small, (ii) among these, most of the attacks are only of four or five different kinds, and (iii) the data is somewhat outdated inasmuch as there nowadays are many new kinds of possible attacks. For these reasons, our results, although promising, should be regarded only as preliminary. In order to establish the broader efficacy of our approach, further experiments of the present type will be warranted when newer such data becomes available.

Third is that, whereas the XML distance measure was created specifically for its use in the present experiment, i.e., dealing with documents representing patterns of alerts, it could serve generally for measuring the distance between any types of objects in terms of their XML representations. Thus this same measure is quite general inasmuch as it could be used in a wide variety of different kinds of clustering algorithms for performing clustering on a wide variety of different kinds of objects. In addition, this measure could be similarly useful in other domains where distance measures are employed, e.g., case-based reasoning [52] and predictive learning [79]. Even more generally, given that XML provides sufficient expressive power to represent any kind of object, this same measure can be used for determining the distance between objects of any type in terms of their XML representations

Table 4.2: Clustering results on DARPA 1998 data

	anomaly	back	eject	format_clear	format	format_fail	ffb	ffb_clear	ftp_write	guest	land	loadmodule	nmap	multihop	phf	perl_clear	satan	spy	warez	warezmater	warezclient	normal telnet	normal http	normal ftp
0									2															
1																	4							
2																	4							
3																	4							
4																							2	
5																			2					
6																	4							
7													8											
8													92											
9									2															
10							5																	
11																							24	
12											17													
13																	4							
14						1											4					2		
15																							6	
16																							15	
17																							2	
18																							2	
19																							3	
20																							4	
21																	5							
22																							7	
23																								96
24																							7	
25																							11	
26																	4							
27																							15	
28																						5		
29																	4							
30																	1							
31	9		7		1					1		2		1		1	1	2				93		
32		100																						
33										49														
34																			1	1	100			2
S			1					1				1		2			1						3	1
T	9	100	7	1	1	1	5	1	4	50	17	3	100	3	3	1	40	2	1	1	100	100	100	100

Table 4.3: Distribution of 'ftp-write' sessions over assigned pure clusters

ftp-write	ftp	login
Cluster-0	2	0
Cluster-9	0	2

Table 4.4: Distribution of 'satan' sessions over assigned pure clusters

satan	eco-i	finger	u	sunrpr	others
Cluster-1	4				
Cluster-2		4			
Cluster-3		4			
Cluster-6		4			
Cluster-13			4		
Cluster-21				5	
Cluster-26			4		
Cluster-29					4

CHAPTER 5

Alert Correlation

Traditional IDSs concentrate on low-level attacks or anomalies, and raise alerts independently, though there may be logical connections between them. In situations where there are intensive intrusions, not only will actual alerts be mixed with false alerts, but the amount of alerts will also become unmanageable. As a result, it is difficult for human experts or intrusion response systems to understand the alerts and take appropriate actions. Meta intrusion detection, on the other hand, is a high-level intrusion detection framework that incorporates a variety of sensors complementing each other for the purpose of making global decisions and discarding false alerts. As the data sources for meta intrusion detection are alert streams (in contrast to network traffic or host audit trails), the key for meta intrusion detection is alert correlation, ie., determining which alerts from the various sensors are caused by the same attack. Due to the diversity of both attacks and detection approaches, there so far has not been found an effective alert correlation approach that can be generally applied. In most current approaches, alert correlation is accomplished by using well-defined knowledge such as rules or predicates that can reveal the deep relations of alerts. But in case-based reasoning systems, the expertise is embodied in a library of past cases rather than being encoded in classical rules. Since no rules are available, we have explored various approaches and have proposed two basic kinds for case-based meta intrusion detection. One kind works in situations when alerts provide sufficient information and simply correlates alerts through explicit information that reveals the connections between alerts. We call this explicit alert correlation. The other works in an implicit manner. It takes advantage of the patterns of alerts in cases and correlates alerts from those patterns. We call this case-oriented alert correlation. It is more general than the first kind and can be used for situations when alerts provide either poor or rich information. It should be noted that alert correlation in case-based intrusion detection is actually the process of formulating problems from the point of case-based reasoning process. The correlated alerts form a pattern of alerts that is the description of a problem situation and will be compared with cases in the case library.

5.1 Related Work

There have been several proposals that address the problem of alert correlation. Approaches, such as probabilistic alert correlation [10] and alert clustering methods [7][23], do not use predefined knowledge other than alerts themselves and are based on the similarity between alert attributes. Although they are effective in finding similar alerts (e.g. alerts with the same source and destination IP addresses), approaches of this type are criticized for not being able to discover deep logical connections between alerts.

Some approaches [24][25] correlate alerts according to the attack scenarios specified by human users or learned from training data by knowledge discovery approaches (e.g. data mining and machine learning). A limitation of these methods is that they are restricted to known attacks or variants of known attacks.

Approaches based on prerequisites and consequences of attacks [13][26][9] may discover novel attack scenarios. Intuitively, the prerequisite of an attack is the necessary condition for the attack to be successful, while the consequence of an attack is the possible outcome of the attack. However, in practice, it is impossible to predefine the complete set of prerequisites and consequences. In fact, some relationships cannot be expressed naturally with the given set of terms.

Some approaches (e.g., [27][8]) apply additional information sources, such as firewalls, and system states in alert correlation. In particular, Morin et al. [27] have proposed a formal model, M2M2, for alert correlation using multiple information sources. Due to the multiple information sources used in alert correlation, the method can potentially lead to better results than those looking at alerts only. However, it invites more human involvement and makes the development of the intrusion detection system more time-consuming and error-prone in practice.

5.2 Explicit Alert Correlation

Although alert correlation is a complicated problem, alerts sometimes can be correlated simply through session information namely, source IP address and port and destination IP address and port. In fact, session information has been considered as key attributes in most intrusion detection approaches used for building intrusion models, as well as for some other purposes. For example, Wenke Lee [38] has viewed attributes such as source IP and port and target IP and port as essential features in describing data items and argue that 'relevant' association rules found by data mining should describe patterns related to essential features. Likewise, the effectiveness of the first class of alert correlation methods introduced in the previous section relies heavily on session information. Those methods correlate alerts based on the similarity between alerts attributes. Alerts with the same IP addresses and ports are inclined to be similar to each other and thus be correlated.

Our explicit alert correlation assumes that alerts with the same session information are caused by the same attack. Hence, it employs a straightforward application of CBR. If the runtime alerts all contain the necessary session information, then they can be sorted according to their sessions, and then, within sessions, be sorted according to the sensors that produced them. Thus, for each session, we get a pattern of alerts that can be directly matched with the alert patterns appearing in the cases in the case library, and the detection process then amounts to retrieving the case that is most similar. The attack described by the retrieved case is assumed to have taken place. As the explicit approach correlates alerts without using any information in cases, it seems independent of the cases at first glance. But, in order to make it work, the cases must be constructed on a session basis first. Even if a known attack used for building cases spans over multiple sessions, it must be divided into multiple phases or parts, each of which occurs only in a single session and has a corresponding case in the case library.

This must be qualified, however, to cover the situation that no cases in the library are sufficiently similar to the given set to warrant this conclusion. To this end we add a requirement that the distance between the given set and the pattern in the case must be below some threshold. In practice, such threshold can be learned by knowledge discovery approaches or specified by sophisticated users. For instance, in our experiments (Chapter 6), we applied a supervised hierarchical clustering algorithm to refine the case library. This algorithm ends with a set of clusters when a predefined fitness function for resulting clusters reaches the optimal value. Cases in the same cluster have either identical, or different but similar, patterns of alerts. Apart from some small amount of possible noise, they should stand for the same type of attack. As a result, the minimal distance between those final clusters can be used as the threshold for reasoning since it can best distinguish different types of cases.

Cases along with the set threshold form an accepting zone for unknown patterns being similar to known ones. Patterns falling outside of the accepting zone are treated as normal behaviors. However, the case library can never cover all possible attacks; outside patterns can be instances of unknown attacks or variants of old attacks. Ignoring them causes the generation of false negatives. Generally, the solution suggested by a CBR system will be tested for correctness in the working environment. If it is not good enough to solve the problem, a new solution may be devised, thus forming a new case to be entered into case library for future reference. This is how a CBR system enriches itself. For the case-based meta intrusion detection, we can save outside patterns for later investigation by human experts, thereby getting humans involved only when necessary.

5.3 Case-Oriented Alert Correlation

The other type of alert correlation approach we have developed is conducted in an implicit manner. Although cases are not as well-defined knowledge as classical rules, they at least are examples of correlated alerts in previously experienced attacks. The basic idea behind implicit approaches is correlating alerts from examples, namely, cases. For this reason, it is called case-oriented alert correlation. In some sense, this approach is an extension of the classical concept of case-based reasoning. A typical CBR system formulates problems from environment data without using any case. Cases are used when they are compared with formulated problems. The case-based alert correlation, however, takes both cases and environment data in consideration at the same time. Cases are involved early in problem formulation rather than during reasoning. We believe that this is somewhat general and deserves exploration in other application domains of case-based reasoning.

5.3.1 Problems with Explicit Alert Correlation

Explicit alert correlation is a straightforward solution. It does have drawbacks though. The important prerequisite to make this approach work is that the alerts themselves contain the session information. Unfortunately, this cannot be satisfied in the following circumstances.

- Data sources do not contain session information. If they are used for intrusion detection, sensors are not able to be associated attacks with sessions. For example, antivirus software can act as a host-based sensor. It works on the binary executables and may fire alerts when virus patterns are found. But executables cannot be associated with sessions. Viruses embedded in executables can be evoked in an unpredictable manner. Other typical data sources without session information include application logs and system logs that only record high-level data such as user commands.

- Sensors do not provide session information. The sensor capabilities are different. Depending on the environment, one may install a lightweight sensor that provides little information in its alerts, or a more complex sensor that will have a greater impact on the running system but provide more detailed alert information. Thus, even with the data source having session information, the sensor may not use it.

- Session information is implicit in data sources. Even in network traffic, session information sometimes is not apparent. IP packets must include both IP address and ports. However, packets in protocols lower than the TCP/IP protocol may not have complete session in formation. For example, ICMP packets only have IP addresses. An attacker may first perform an "IP sweep" (sending a series of ICMP packets) to find live machines in a particular network and then follow with attack sessions to break into those hosts by exploiting some vulnerabilities. In such cases, the ICMP packets can only be associated with sessions from the attack context.

- Not every intrusion occurs over a network. The system can be compromised by insiders, such as attacks involving an abuse of privileges or previously installed backdoor software.

The drawbacks of explicit alert correlation drove us to look for a more general approach. This lead to the idea of the case-based alert correlation.

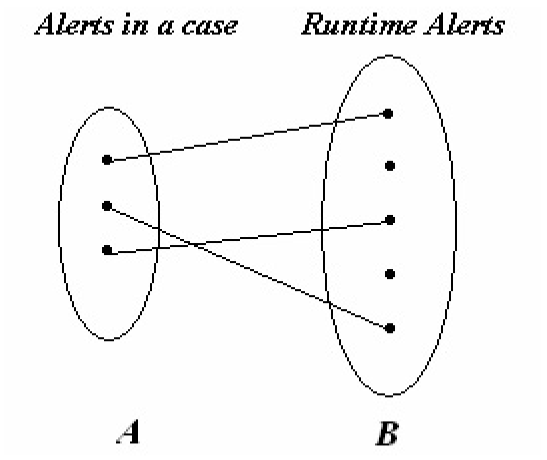


Figure 5.1: Reduction of alert correlation to matching problems

5.3.2 The Method

This approach is called case-oriented alert correlation because it makes use of the case library. Here we assume that there is a case library that contains descriptions of known attacks, where, as before, cases are identified by their patterns of alerts represented in XML. The underlying idea is, given some collection of alerts raised by the various sensors during some given time frame, to find those cases in the case library whose alert patterns are sufficiently similar to some subset of the given set to suggest that the attack described by that case has taken place.

To be more specific, the problem of alert correlation has been reduced to matching problems as follows. The pattern of alerts in a given case and the collection of runtime alerts form two sets of alerts like the example shown in Figure 5.1. An alert in one set can be matched with up to one alert in the other set. The distance between these two alerts is associated with that match. Suppose all the pair-wise distances between alerts in these two sets are available. A matching problem can be defined as one of looking for matches with the overall minimal sum of their distances. The resulting matching determines a pattern of alerts that can be found from the runtime alerts and is most similar to one in the given case. Given a set of runtime alerts, a matching problem is formulated for each case in the case library.

The manner in which these subsets are identified and applied is exemplarily illustrated in

Figure 5.2. A fragment of some given set of alerts is shown in the middle column, where the alerts are organized according to the sensors that produced them. Here there are 6 alerts, with 3 coming from sensor $S1$, denoted B_{11}, B_{12}, and B_{13}, and 3 from sensor $S2$, denoted B_{21}, B_{22}, and B_{23}.

On the left is shown the alert pattern from some case in the case library. For the purpose of the illustration, this pattern also involves the same sensors, S1 and S2, although in general this need not be true. Some cases might involve other sensors, and might not involve either S1 or S2. The pattern of alerts in the given case consists of 4 alerts, A_{11} and A_{12} from S1, A_{21} and A_{22} from S2.

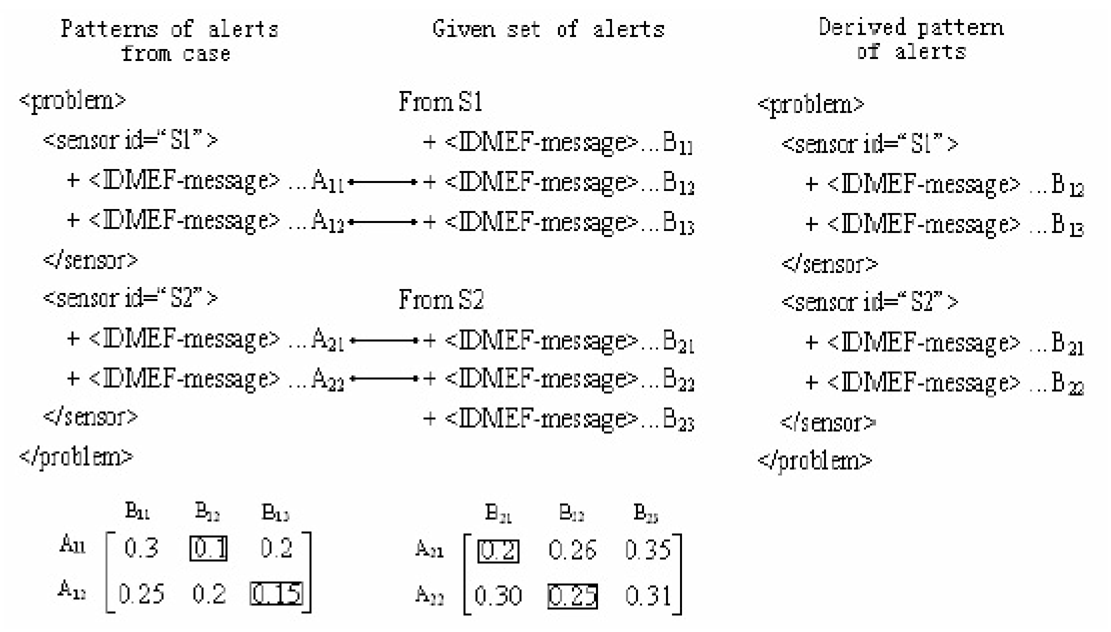
We use the pattern in the case to extract a subset of the alerts in the given set and build a new (derived) pattern as shown on the right. This is done as follows. First, for each sensor, we compute the pair-wise distance between the alerts in the case and those in the given set, using the distance measure described in Chapter 4. These are then recorded in a distance matrix as shown at the bottom of Figure 5.2.

To these matrices we next apply the Hungarian algorithm described in the previous chapter, to find the optimal matching, i.e., the set of alert pairs with the minimum sum of the distances. These are shown in the boxes in the matrices. The matched alerts from the given set are then extracted to form the derived set. Last we apply the distance measure again to find the distance between the derived pattern and the pattern of the case. If this distance is below some specified threshold, we take this is as meaning what was explained above, i.e., that alerts extracted from the given set are correlated in the same manner as their corresponding alerts in the case and that the attack represented by the case is likely to have occurred.

Again, this is done for every case in the case library. Thus it is possible that more than one case will be matched in this manner, indicating that possibly more than one attack has taken place.

5.4 Enhanced Case-Oriented Alert Correlation

The alerts in a given case are sorted in a chronological order within sensors as we have described before. The runtime alerts can also be sorted by time. The enhanced alert correlation takes the ordering into consideration and looks for what we call the "ordered maximal weighted matching" rather than the maximal weighted matching. The ordered



$$\begin{array}{c|ccc} & B_{11} & B_{12} & B_{13} \\ \hline A_{11} & 0.3 & \boxed{0.1} & 0.2 \\ A_{12} & 0.25 & 0.2 & \boxed{0.15} \end{array} \qquad \begin{array}{c|ccc} & B_{21} & B_{22} & B_{23} \\ \hline A_{21} & \boxed{0.2} & 0.26 & 0.35 \\ A_{22} & 0.30 & \boxed{0.25} & 0.31 \end{array}$$

Figure 5.2: An example of case-oriented alert correlation

maximal weight matching puts a sequence constraint on conventional assignment problems. It has not been studied before. We found it can be solved with dynamic programming.

5.4.1 Problem with Original Case-Oriented Alert Correlation

The basic idea behind the original case-oriented alert correlation is it models alert correlation as a series of plain assignment problems. The alerts in a given case are sorted in chronological order. Runtime alerts also occur in some sequence. However, the pattern of alerts derived from the given case by finding the maximal weighted matching may break the sequence of runtime alerts. For example, in Figure 5.3, if elements in sets are ordered, (a) and (b) show two matchings where the ordering has been broken, while (c) and (d) are two matchings where the ordering is preserved.

5.4.2 Formal Definition of Ordered Matching of Minimal Cost

Let $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$ be two sets of objects that we wish to compare. (In our matching problem, the objects will be alerts.) Let be $G = \{V, E\}$ an undirected graph where $V = X \cup Y$ is the set of nodes and $E \subseteq X \times Y$ is a set of edges.

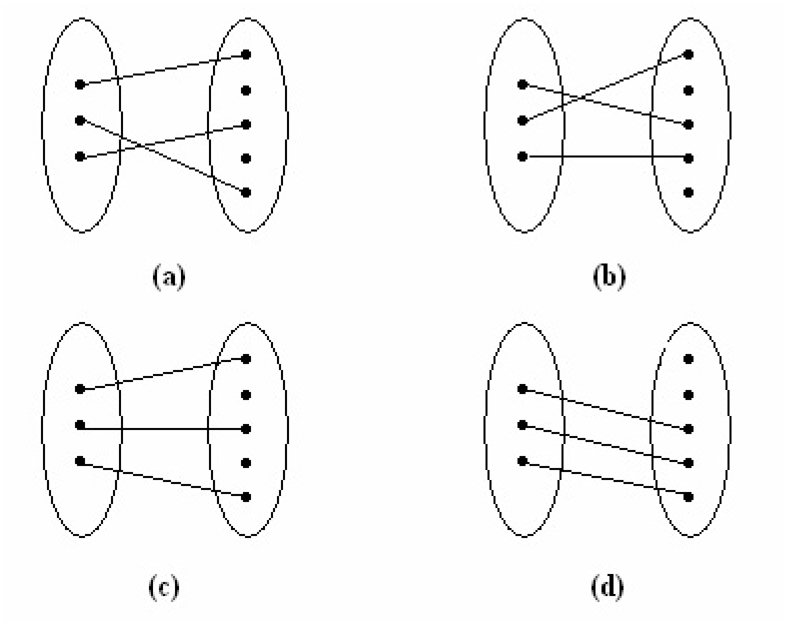


Figure 5.3: arbitrary matching (a) and (b) and Ordered matching (c) and (d).

Because each edge connects a node in X to a node in Y, G is bipartite. A matching in G is a subset M of the edges E such that, among the edges (x_i, y_i) in M, each x_i is connected to a unique y_j and, conversely, each y_j is connected to a unique x_i. Let $c : E \rightarrow R_+$, where R_+ is the nonnegative reals, be an assignment of a numeric cost to each edge in E. (In our application, this cost will be the distance between alerts.) Then we define the cost of M to be the sum of the costs of the edges in M. A maximum matching in a graph G is a matching having the maximum possible cardinality (number of edges). The assignment problem is to find a maximum matching having minimal cost.

In particular, G is complete bipartite graph, i.e., every node in X has a link to every node in Y; the edge set E can be more specifically denoted as $\{(x, y)|x \in X, y \in Y\}$; the cost function can be given as $c : X \times Y \rightarrow R_+$; and the values of c can be put into a $n \times m$ matrix C , where element at the ith row and jth column is the cost for edge (x_i, y_j). Since all the assignment problems we meet in our application domain will be totally connected bipartite graphs, our discussion assumes that such a cost matrix C is given for every assignment problem. Moreover, in a complete bipartite graph, the cardinality of a maximum matching will always be just $min\{|X|, |Y|\}$.

Given a complete bipartite graph G with associated cost matrix C , a maximal matching having minimal cost can be obtained by applying the classical Hungarian algorithm (Appendix A) to C. However, we wish to impose an additional constraint when looking for the matchings. Suppose that the members of the node sets X and Y, associated with G, are ordered, i.e., so that $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$ may be regarded as sequences. For the sake of this discussion, let us assume that $|X| < |Y|$. In this case a matching M for G determines a function $f : X \to Y$. We shall say that the matching is ordered if, for any two elements x_i and x_j from $|X|$ with $i < j$, their matches $f(x_i) = y_k$ and $f(x_j) = y_l$ in Y are such that $k < l$.

If the solution for the assignment problem is restricted to ordered matchings, the goal of the assignment problem may be described as that of finding an ordered maximum matching with minimal cost.

5.4.3 Dynamic Programming

A certain class of optimization problems can be solved through a process of recursive decomposition. The given problem is decomposed into a collection of sub-problems, which in turn are further decomposed into sub-problems, and so on until no further decomposition is possible. Sometimes the compositions form a tree, and sometimes they comprise more complex networks or grids. Given such a decomposition, the solutions to the sub-problems are then determined in reverse order. One first solves the problems at the lowest level, uses these to contribute to solutions for the problems at the next higher level, and so on, until reaching back to the original problem. At each such step, the solution at any given node can be made optimal based on the solutions to its subproblems. Thus when one finally in this manner arrives back to the original problem, the optimal solution has been found.

During such a decomposition, it typically happens that the same subproblems may re-occur in many different paths. Thus a brute force computational approach would entail considerable redundancy in re-computing each such subproblem each time it occurs.

Dynamic programming is a problem solving technique that applies the above methodology, but eliminates these redundant computations by caching sub-problem solutions. It refers to a large class of algorithms. Examples of dynamic programming include finding the longest common subsequence on strings [80], the knapsack problem, and shortest path algorithms on graphs [81].

We will show that the problem of finding an ordered matching of minimal cost, discussed in the preceding section, can be solved using this technique. Two approaches will be described: one where the matching problem is decomposed along a tree, and where the optimization problem amounts to finding a path through the tree having minimal cost; and one where the tree is mapped onto a graph, and the optimization problem amounts to finding a path through the graph having minimal cost. These two approaches are described for the sake of illustrating the possibilities provided by the theory of dynamic programming. Of these, the first one will be used in the following section as the basis for the algorithm employed in the present application. Note that this section only lays out how the optimal solution can be obtained through recursive problem decomposition. The next following section presents the actual dynamic programming algorithm, which implements the caching of subproblem solutions.

To illustrate the method, suppose we are given two sets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3, y_4, y_5\}$, with cost matrix C,

$$\begin{bmatrix} 2 & 1 & 5 & 4 & 3 \\ 6 & 7 & 8 & 9 & 10 \\ 3 & 6 & 4 & 5 & 2 \end{bmatrix}$$

where the element $c_{i,j}$ is the cost $c(x_i, y_j)$ as described in the previous section. We wish to find a minimal cost matching of the members of X with those in Y that preserves their orderings. The possible matchings for each member of X, given this ordering constraint, are shown in Figure 5.4. The left diagram shows that x_1 can be matched with any of y_1, y_2, or y_3. The reason that x_1 cannot be matched with either y_4 or y_5 is that then there would be no order-preserving matchings for x_2 and x_3. Similar considerations apply to give the possible matchings indicated in the center and right diagrams for x_2 and x_3.

Based on this analysis, one can decompose the optimal matching problem along a tree as shown in 5.5. We consider the possible matchings with the x_i in reverse order. From a root node, first generate nodes that represent the possible matchings with x_3. These are represented in Figure 5.5 as the pairs (x_3, y_3), (x_3, y_4), (x_3, y_5). Then, for each of these, generate nodes that represent the possible matchings with x_2. For example, if x_3 is matched with y_3 as represented by the node (x_3, y_3), there is only one possible order-preserving matching for x_2, namely (x_2, y_2). After delineating the possible order-preserving matchings for x_2, for each of the above possible matchings for x_3, then proceed similarly,

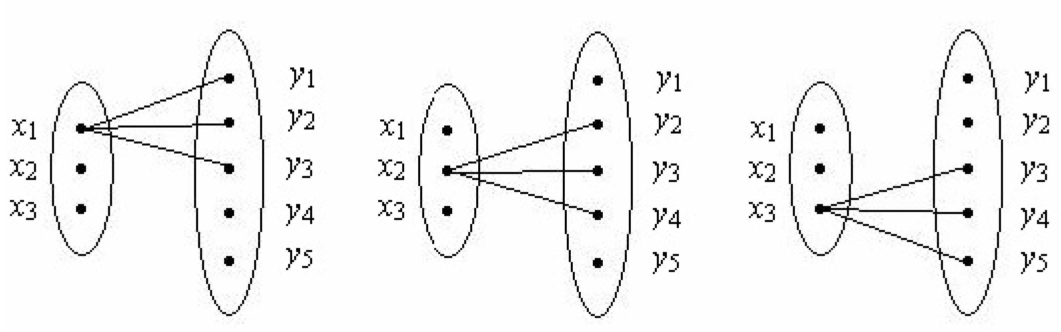


Figure 5.4: Possible matchings for the x_i.

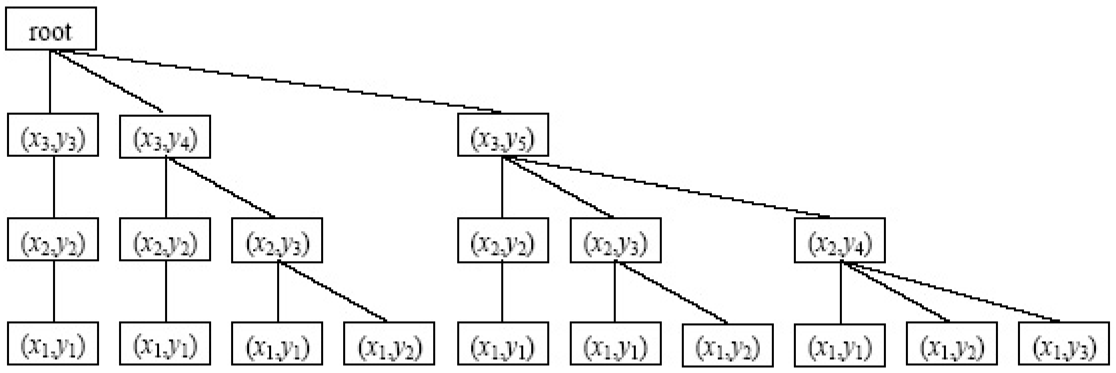


Figure 5.5: Decomposition of optimization problem into subproblems.

for each node at this level, to generate the possible matchings for x_1. This completes the recursive decomposition as shown in the figure.

Now assume that the link in Figure 5.5 connecting (x_i, y_j) to its parent node is labeled with the cost $c_{i,h}$ taken from the cost matrix C. The objective is to select from this tree a path with minimal sum of these costs. In this example there are two such minimal paths, $\{(x_1, y_1), (x_2, y_2), (x_3, y_5)\}$ and $\{(x_1, y_2), (x_2, y_3), (x_3, y_5)\}$, both having total cost 11. Either of these can be taken as the optimal ordered matching.

How one determines this minimal cost is the crux of dynamic programming. The method will first be explained in terms of the foregoing example, and then the general formulation will be presented. We consider the process of determining a set of matchings for the x_i as progressing through a series of stages, where in the present example, one first chooses a matching for x_1, then one for x_2, and finally one for x_3. Figure 5.5 delineates all such possible order-preserving matchings, so the paths upward through the tree represent all the possible such series of stages. The computation of the cost of the minimal-cost order-preserving matching proceeds by computing the minimum cost of getting to each stage along any such path as the cost of the matching performed at that stage plus the minimum of the costs of all paths leading up to that stage. In the language of dynamic programming, if the action at stage s is to match x_s with y_k, then that stage is said to be in state k_s. Given these notations, then the minimum cost of getting to this stage and state is denoted $f_s(k_s)$. These costs for the stages and states in the foregoing example are determined as follows.

$f_1(1) = 2,\ f_1(2) = 1,\ f_1(3) = 5$

$f_2(2) = min\{7 + f_1(1)\} = min\{7 + 2\} = 9$

$f_2(3) = min\{8 + f_1(1), 8 + f_1(2)\} = min\{8 + 2, 8 + 1\} = 9$

$f_2(4) = min\{9 + f_1(1), 9 + f_1(2), 9 + f_1(3)\} = min\{9 + 2, 9 + 1, 9 + 5\} = 10$

$f_3(3) = min\{4 + f_2(2)\} = min\{4 + 9\} = 13$

$f_3(4) = min\{5 + f_2(2), 5 + f_2(3)\} = min\{5 + 9, 5 + 9\} = 14$

$f_3(5) = min\{2 + f_2(2), 2 + f_2(3), 2 + f_2(4)\} = min\{2 + 9, 2 + 9, 2 + 10\} = 11$

Thus the minimal possible overall cost is determined to be the minimum of the values for f_3, namely, 11.

The matchings that provide this solution are not explicitly evident from the computational process, but can be determined as part of the implementation of the dynamic programming algorithm. This will be discussed the section below.

The general case illustrated by this example can now be described as follows. Suppose we are given two sets $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$. For purposes of this discussion, assume that X is the smaller of the two, i.e., that $n \leq m$. Then an element in X can only match with $m - n + 1$ elements in Y ; the candidate matches for the ith element in X are the ith to $(i + m - n)$th elements in Y.

As in the example, we can envision this problem as being decomposed along a tree, where here the tree has n levels. Moreover, again each path upwards through the tree represents a series of stages in constructing a complete matching for the x_i. As above, where s is the stage and y_k is the element matched with x_s at this stage, the stage is said to be in state k_s, and the minimum cost of getting to this stage and state is denoted $f_s(k_s)$. The recursion formulas for dynamic programming, which compute these costs are:

$$f_1(k_1) = c_{1,k_1} \tag{5.1}$$

for all possible values of k_1,and

$$f_i(k_i) = \min_{i-1 \le j < k_i} \{f_{i-1}(j) + c_{i,k_i}\}, \tag{5.2}$$

for $1 < i \le n$

for all possible values of each k_i. The constraint $i - 1 \le j < k_i$ on states over which we minimize in (b) ensures the requirement for ordered matching. The computation produces different costs on different states at the final stage n, namely, $f_n(n)$, $f_n(n+1)$,.., and $f_n(m)$. The solution for the overall problem is the minimal cost, namely, $\min_{n \le k_n \le m} \{f_n(k_n)\}$.

Alternatively, we can reduce the ordered matching problem to a shortest path problem on a graph which is well-known as a dynamic programming problem. The graph is constructed as follows. Given two sets $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_m\}$. The cost of x_i matching y_j is given as $c_{i,j}$. Suppose $n < m$. First we construct n layers with $m - n + 1$ nodes on each. The $m - n + 1$ nodes in the ith layer represent the candidate matches for x_i and are denoted as i, $i + 1$,..., and $i + m - n$. Two additional layers, 0 and $i + 1$, are added before the first layer and after the last layer respectively. Layer 0 has only one node S (source), layer $i + 1$ has only one node D (destination). The node S has a directed link to node k in the first layer with distance equal to the value $c_{1,k}$. Each node in the last layer has a directed link to the end node with distance of zero. There is a directed link from the node p on the ith layer to the q node in $(i + 1)$th layer with distance $c_{i+1,q}$, if x_i matching with y_p and x_{i+1} matching with y_q will not compromise the ordering constraint. Rename the duplicate nodes somehow to make sure a node takes place at most once on the graph, for a graph does not allow duplicate nodes. Then the shortest path from source to destination on the constructed graph is the minimal cost for ordered matching. If we let k denote a node at state i and let $f_i(k)$ be the shortest path from node k to destination D, the recursive formula is

$f_i(k) = \min_{nodezinstagei+1} \{d_{kz} + f_{i+1}(Z)\}$ where d_{kz} denotes the distance from node k to z. An example, if the weight matrix is

$$\begin{bmatrix} 2 & 1 & 5 & 4 & 3 \\ 6 & 7 & 8 & 9 & 10 \\ 3 & 6 & 4 & 5 & 2 \end{bmatrix}$$

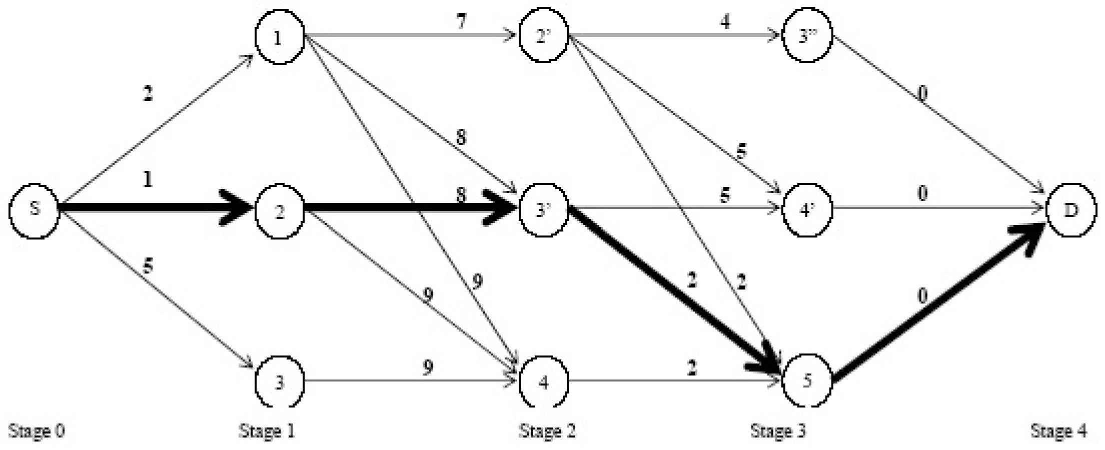


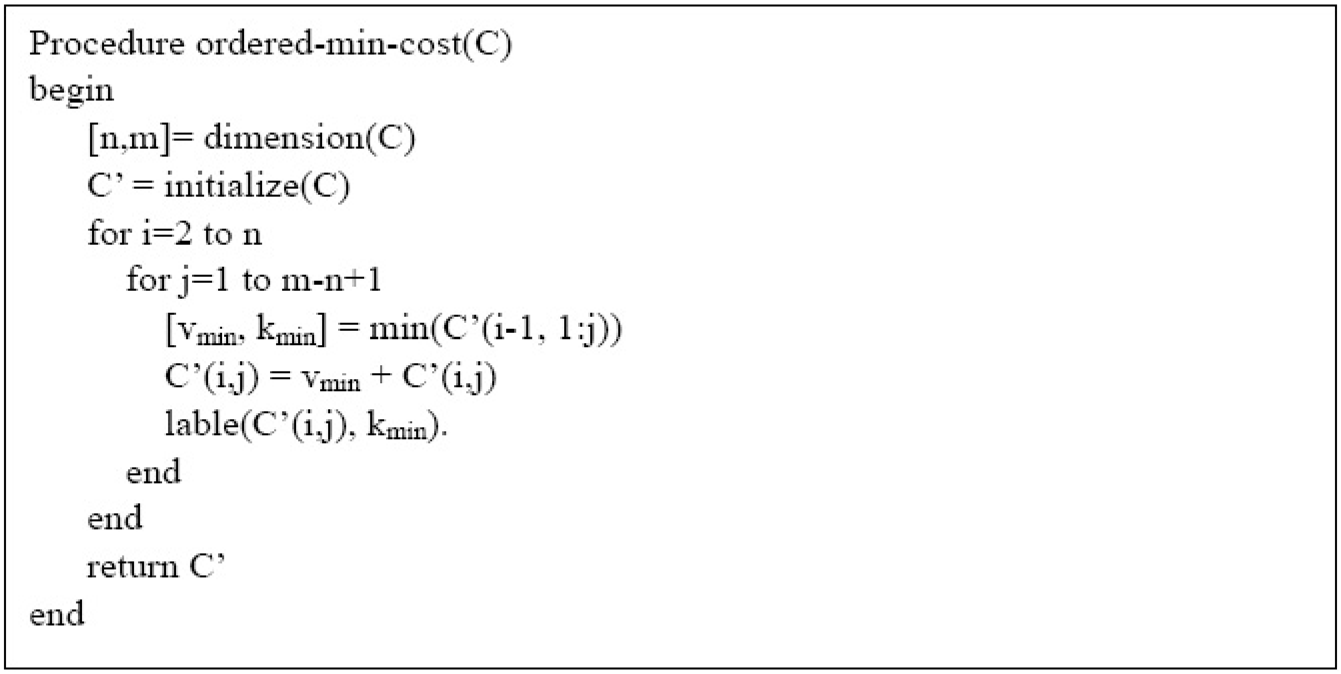
Figure 5.6: An example of reduction of order matching to shortest path on a graph

The graph, shown in Figure 5.6, is constructed according to the procedure described above. The computation starts by setting $f_4(D) = 0$. The shortest distance from S to D is denoted as $f_0(S)$, which is 11. The path is indicated with thick arrows in the graph.

5.4.4 The Algorithm

Although dynamic programming can deal with our problem in two different ways, our algorithm is actually based on the analysis described by formulas 5.1 and 5.2 to avoid constructing graphs. However, Dijkstra's algorithm [81] is the dynamic algorithm that solves the problem of shortest path on a graph. It seems natural there should be a recursive solution for our algorithm. However, the implementation of our dynamic algorithm is by iteration instead of recursion, even though the analysis of a problem naturally suggests a recursive solution. This is because a recursive solution may cause many common sub-problems to be computed repeatedly. For example, in order to compute $f_3(4)$ and $f_3(5)$, some sub-problems, such as $f_2(2)$ and $f_2(3)$ have to be computed more than once. As an optimization, we can compute such sub-problems once and then store the results to read back later. The iteration solution thus is introduced to reduce call overhead. Such iterative implementations of dynamic programming algorithms are quite common in dynamic programming research [82].

Descriptions of dynamic programming algorithms are often harder to understand than



```
Procedure ordered-min-cost(C)
begin
    [n,m]= dimension(C)
    C' = initialize(C)
    for i=2 to n
      for j=1 to m-n+1
        [$v_{min}$, $k_{min}$] = min(C'(i-1, 1:j))
        C'(i,j) = $v_{min}$ + C'(i,j)
        lable(C'(i,j), $k_{min}$).
      end
    end
    return C'
end
```

Figure 5.7: Algorithm of ordered maximal matching

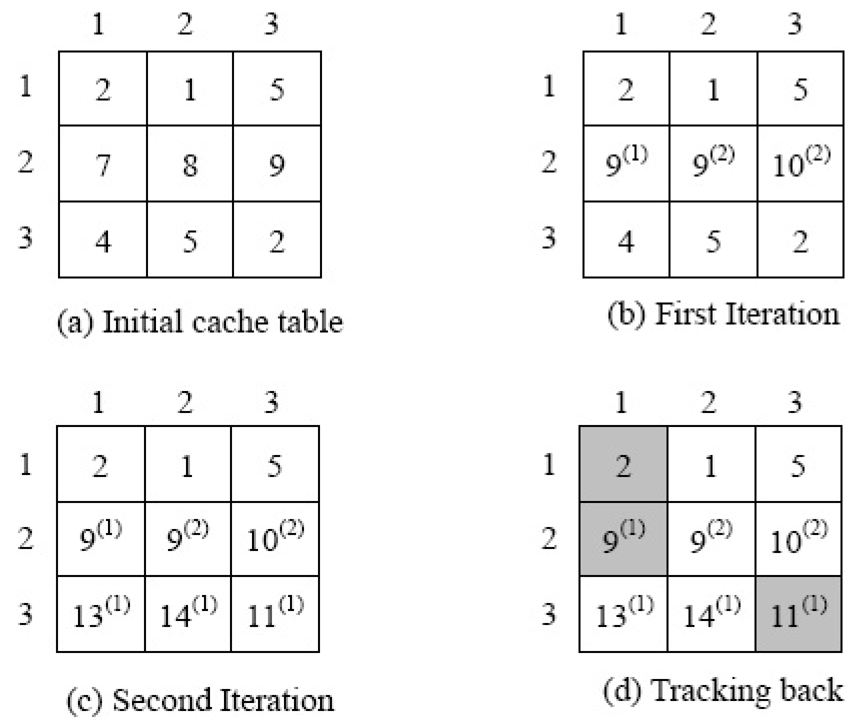
the recursive versions. The essential difference is that dynamic programming keeps its intermediate results whereas recursion does not. This makes a significant difference for performance when a recursive function is called repeatedly with the same arguments. In fact, dynamic programming is nothing more than recursion with the addition of a caching strategy.

The algorithm is presented in Figure 5.7. It consists of two major parts. One part is the initialization of a cache table C' that stores the results to read back later. The latter procedure is presented in Figure 5.8. The initial values of C' are costs of elements in X matching with their candidate matches. Thus, C' has n rows and $(m - n + 1)$ columns. $C'(i, j)$ is the cost of x_i matching y_j. Given the example before, the initialized cache table is shown in (a) of Figure 5.9.

The other part is updating and labeling the cache table, for element $C'(i, j)$ from the 2nd row to the last row in the table C' is updated as and labeled as the value of k. Figure 5.9 (b) and (c), shows the changes of the cache table for the example. Once all elements have been updated, the minimal value of the last row is the cost of the ordered maximal weight matching. The exact matches can be tracked down through the labels in the returned cache

```
Procedure initialize(C)
begin
    [n,m]= dimension(C)
    for i=1 to n
        C'(i,:) = C(i,i:m-n+i)
    end
    return C'
end
```

Figure 5.8: Initialize the cache table



	1	2	3
1	2	1	5
2	7	8	9
3	4	5	2

(a) Initial cache table

	1	2	3
1	2	1	5
2	$9^{(1)}$	$9^{(2)}$	$10^{(2)}$
3	4	5	2

(b) First Iteration

	1	2	3
1	2	1	5
2	$9^{(1)}$	$9^{(2)}$	$10^{(2)}$
3	$13^{(1)}$	$14^{(1)}$	$11^{(1)}$

(c) Second Iteration

	1	2	3
1	2	1	5
2	$9^{(1)}$	$9^{(2)}$	$10^{(2)}$
3	$13^{(1)}$	$14^{(1)}$	$11^{(1)}$

(d) Tracking back

Figure 5.9: An example of changes in a cache table

table C'. The grey boxes in Figure 5.9(d), shows how to identify matching. In this particular example, x_1 matches its first candidate match; x_2 matches its first candidate match; and x_3 matches its second candidate match.

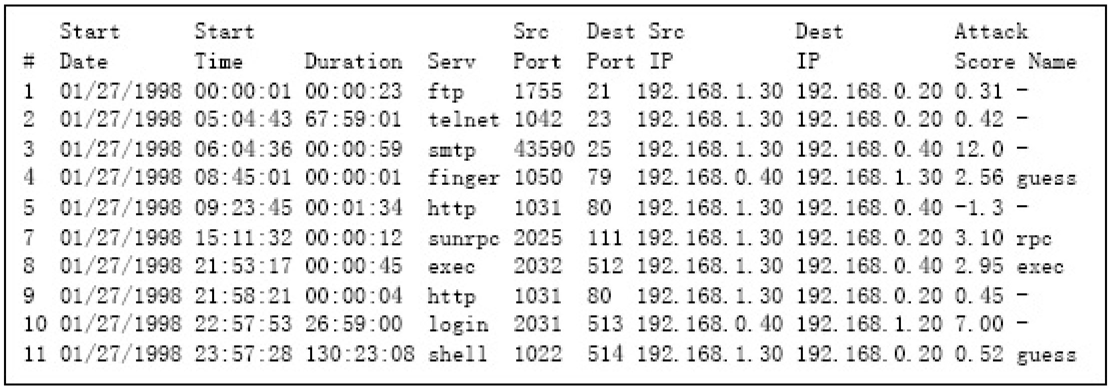
CHAPTER 6

Experimental Evaluation

Training data is important for a knowledge-based IDS in that the detection performance highly depends on the quality of the training data. Lee [39] pointed out that an IDS can not be better than the training data on which it was constructed. However, the creation of training data is a time-consuming and difficult task, because it must be thorough and every piece of data has to be manually labeled as either normal or abnormal. Training data for meta intrusion detection is even more difficult to produce since it should contain multiple data sources plus their correlated information. Hence, we choose to use available datasets for our experiments rather than creating our own data. Besides training, these datasets were used for testing too. In this chapter, we describe our experiments for the evaluation of our case-based meta intrusion detection approach with three different datasets created and distributed by IDS-related projects of the Defense Advanced Research Project Agency (DARPA)[83][84]. As there are few common datasets available for the intrusion detection research, DARPA datasets are relatively popular and broadly considered in this community as the standard corpora for the evaluation of intrusion detection systems. Some institutions, such as SRI, are creating new standard data that contains up-to-date attacks and will become available in near future.

6.1 The Experiment with DARPA 1998 data sets

There are two sensors involved in the experiment with DARPA 1998 datasets, a host-based sensor, STIDE, and a network-based sensor , Snort. As the original STIDE does not generate alerts, it has been modified to act like a real sensor. The cases and problems were constructed on a session basis. In other words, a case or a problem is the pattern of alerts generated during a session. The case library was constructed from DARPA 1998 training data and



```
   Start      Start                          Src   Dest Src          Dest          Attack
#  Date       Time     Duration  Serv        Port  Port IP           IP            Score Name
1  01/27/1998 00:00:01 00:00:23  ftp         1755  21   192.168.1.30 192.168.0.20  0.31  -
2  01/27/1998 05:04:43 67:59:01  telnet      1042  23   192.168.1.30 192.168.0.20  0.42  -
3  01/27/1998 06:04:36 00:00:59  smtp        43590 25   192.168.1.30 192.168.0.40  12.0  -
4  01/27/1998 08:45:01 00:00:01  finger      1050  79   192.168.0.40 192.168.1.30  2.56  guess
5  01/27/1998 09:23:45 00:01:34  http        1031  80   192.168.1.30 192.168.0.40  -1.3  -
7  01/27/1998 15:11:32 00:00:12  sunrpc      2025  111  192.168.1.30 192.168.0.20  3.10  rpc
8  01/27/1998 21:53:17 00:00:45  exec        2032  512  192.168.1.30 192.168.0.40  2.95  exec
9  01/27/1998 21:58:21 00:00:04  http        1031  80   192.168.1.30 192.168.0.20  0.45  -
10 01/27/1998 22:57:53 26:59:00  login       2031  513  192.168.0.40 192.168.1.20  7.00  -
11 01/27/1998 23:57:28 130:23:08 shell       1022  514  192.168.1.30 192.168.0.20  0.52  guess
```

Figure 6.1: the fragment of a list file

refined using our supervised clustering approach. The experimental results have shown that compared with each individual sensor, the meta IDS has improved the overall detection performance.

6.1.1 Description Of the Datasets

The DARPA 1998 datasets provide nine weeks of data, seven weeks for training and two weeks for testing. Training data and testing data are basically the same except they are used for different purposes. Both have two data sources. One is Solaris BSM Audit Data which records system calls taking place on a particular computer. It is a typical data source for a host-based sensor. The other is tcpdump data for a network-based sensor. It contains the network traffic (packets) that have passed a certain location in a network.

The labeled information of training data is given in list files. There are two types of list files, namely, BSM list files and tcpdump list files. Both types employ the same format and include the session information and attack information, as shown in Figure 6.1. Intuitively, BSM list files list all the host sessions in the BSM audit data; the tcpdump list files list all the network sessions in tcpdump data. If a host session and a network session share the same source IP and port and destination IP and port, they are correlated with an unique number as indicated in the first column in Figure 6.1.

6.1.2 Modification of STIDE

STIDE is a host-based anomaly detection program developed by the University of New Mexico to demonstrate that sequences of system calls can be used for anomaly detection [5][6]. The format for STIDE input is a simple table with two columns, namely, process ID and system call ID. Its output is mainly mismatch rates. As the original STIDE does not take the BSM audit data as input or generate IDMEF alerts, a few Perl scripts were written to prepare the STIDE input and the basic alerts information. Figure 6.2 shows the data flows among these Perl programs. In BSM audit data, each system call is described by a set of tokens that contain the details of the system call. The program bsm_filter.pl summarizes the necessary information of each system call in a single line. Its output is called filtered_data and goes to two other Perl programs. Firstly, the bsmlist.pl program looks for patterns identifying sessions of interest in the filtered data and saves the session information (IP address, ports, and ID of starting processes) in list files. Secondly, the process_list.pl program sorts the system calls in filtered_data by processes and saves the results in process files. Finally, for each listed session in the list files, the session.pl program extracts its relevant system calls (including those in the starting process and its children processes) from process files and puts them in the format of STIDE input.

STIDE also produces auxiliary information, such as number of anomalies, hamming distance (HD), and local frame count (LFC). In order to provide the CBR system with the complete analysis detail from STIDE, all possible STIDE outputs together with the session information found using those Perl scripts are represented as IDMEF messages that serve as the alerts from STIDE. IDMEF is a well-designed alert language. According to the specification of IDMEF, it is clear that session information (Source IP and Port and Target IP and Port) should be put into XML elements *Source* or *Target*. It is also flexible enough to accommodate domain specific information using XML elements *AdditionalData*. We extend the IDMEF to include STIDE specific outputs. An example is shown in Figure 6.3.

6.1.3 Experiment Setup

We used the adaptive CBR system [34] as the reasoning engine and selected STIDE and Snort as the sensors to process the BSM Audit data and the tcpdump data respectively. The original STIDE was modified to generate alerts as described in the previous section. As

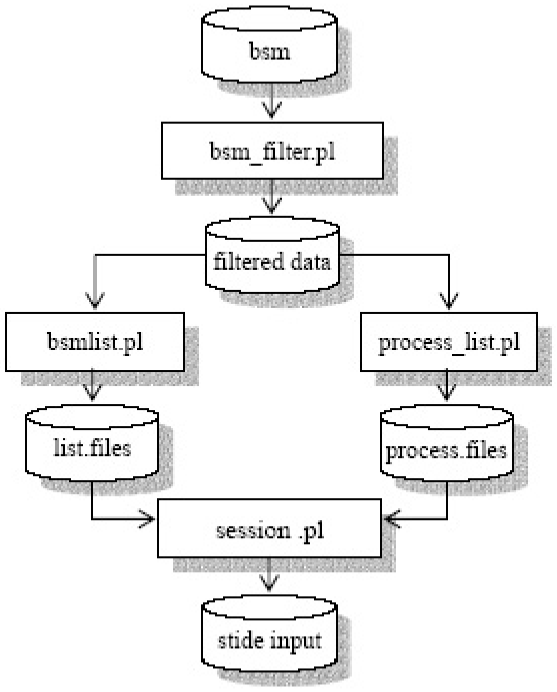
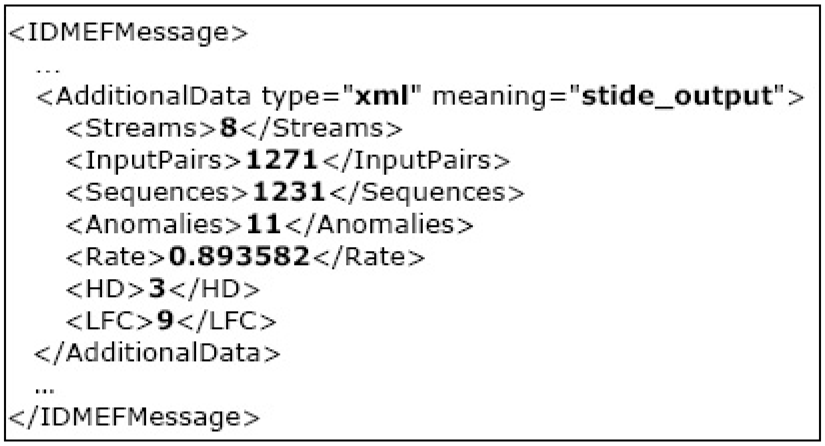


Figure 6.2: The data flow in Preprocess



```
<IDMEFMessage>
  ...
  <AdditionalData type="xml" meaning="stide_output">
    <Streams>8</Streams>
    <InputPairs>1271</InputPairs>
    <Sequences>1231</Sequences>
    <Anomalies>11</Anomalies>
    <Rate>0.893582</Rate>
    <HD>3</HD>
    <LFC>9</LFC>
  </AdditionalData>
  ...
</IDMEFMessage>
```

Figure 6.3: The IDMEF representation of STIDE outputs

STIDE applies anomaly detection. It was trained with the normal sessions of some common services, such as ftp, telnet, and smtp. Most attacks in the DARPA 1998 datasets were launched against these types of services. Snort was configured as using the standard default rules and was set up with an IDMEF output plug-in (http://sourceforge.net/projects/snort-idmef) that can automatically transform default Snort alerts into IDMEF messages. All IDMEF messages are saved in a relational database (we used MySQL) through a middleware called XML2DBMS (http://www.rpbourret.com/xmldbms/) so that we can perform SQL searches to collect alerts of interest.

In training, the attack sessions indicated in the intersection of BSM list and tcpdump list files were used to build the case library. Because tcpdump list files may contain network sessions irrelevant to the host where the BSM audit data was collected, only the intersection of host sessions and network sessions were applied. For each attack session, its session information defines an SQL statement that returns all alerts generated during that attack session. A middleware called DBMS2XML (http://www.rpbourret.com/xmldbms/) transforms these alerts into a pattern of alerts. The pattern of alerts, together with the type of the attack, forms a case. Figure 6.4 represents the overview of the training process.

The testing process is almost the same as the training process except that list files were not used to create SQL statements. In testing, alerts in the database were correlated via session information. Alerts with the same session information were put together and represented in a pattern of alerts using DBMS2XML. Such patterns serve as problems for the CBR system and will be compared with cases in order to find the most similar ones. If a problem and its most similar case are similar enough, the CBR system indicates that an attack whose type is the same as the case may have taken place. The BSM list and tcpdump list files in testing data are used to verify the decisions of the CBR system. The overview of the testing process is presented in Figure 6.5.

6.1.4 Construction of Case Library from Training Data

The library of past cases comprises the knowledge of a CBR system. A good case library should be representative of all possible problems and contain as little redundancy as possible. A three-step procedure has been developed for the construction of the case library from DARPA 1998 training data. These are creating initial cases, clustering initial cases, and selecting final cases. These steps are illustrated in Figure 6.6.

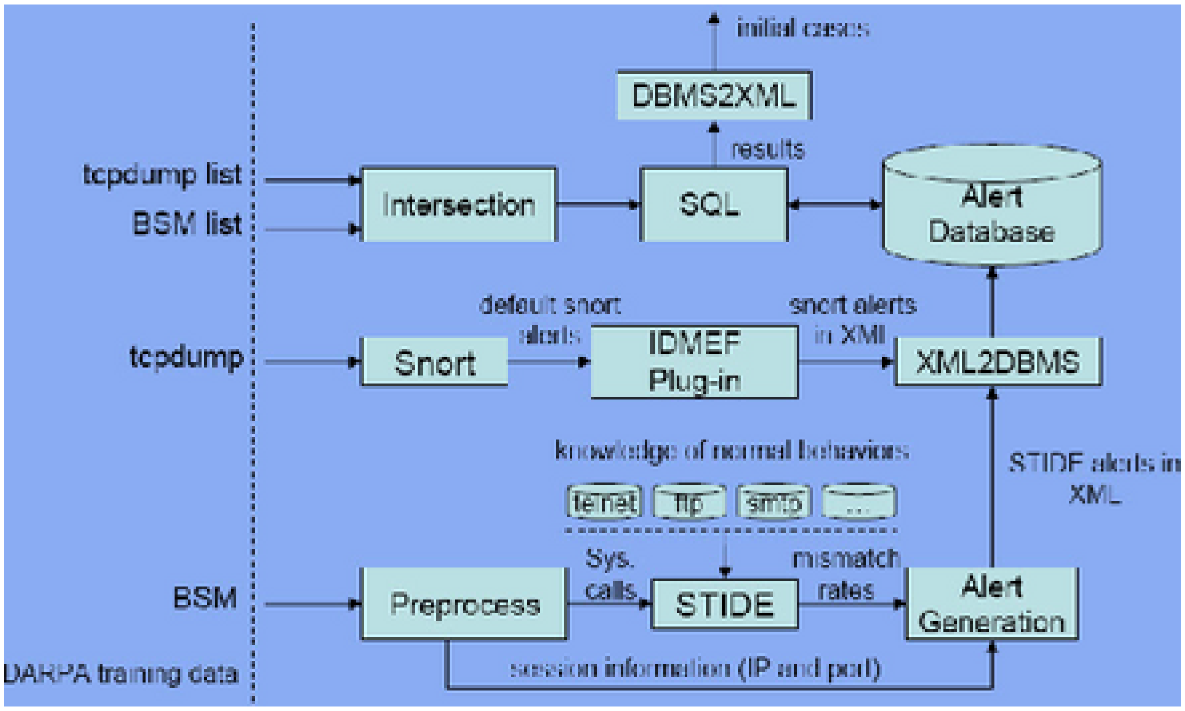


Figure 6.4: the Overview of Training

In the first step, the training data is fed into the sensors. Sensors generate streams of IDMEF alerts. From the labeled information in training data, alerts corresponding to the same attack instance are aggregated into an XML object. Each instance of an attack is associated with an XML object of this kind if there is any alert generated for that instance. Such XML objects collected in the first step are called initial cases. The number of initial cases can be very large. It is possible that many initial cases correspond to the same type of attack and that most of them are very similar to each other. For example, our experiments with the DARPA 1998 datasets produced hundreds of similar cases associated with the type 'warezclient'. If the CBR system had to go through all these cases for every problem, the performance would be very poor.

Obviously, the use of large numbers of similar cases is redundant. In order to remove this redundancy, a clustering approach is applied in the second step. We used the supervised hierarchical clustering algorithm described in Chapter 4. The output of clustering algorithm is a set of clusters.

According to whether or not all objects in a cluster belong to the same type, a cluster

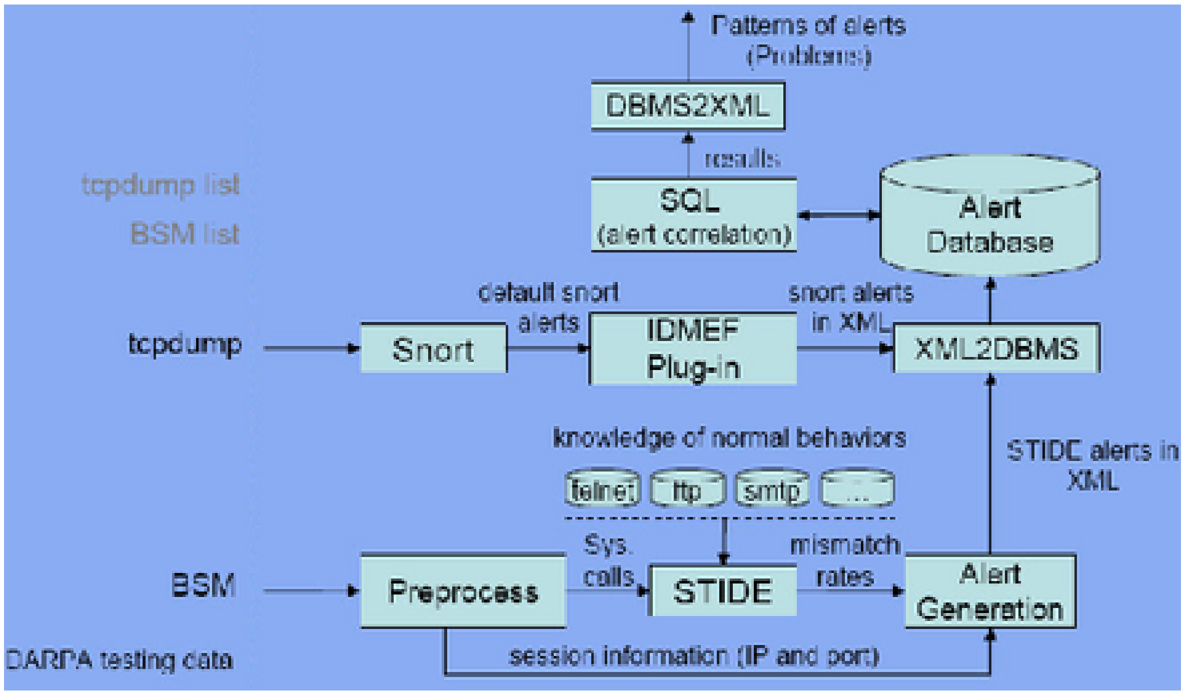


Figure 6.5: the Overview of Testing

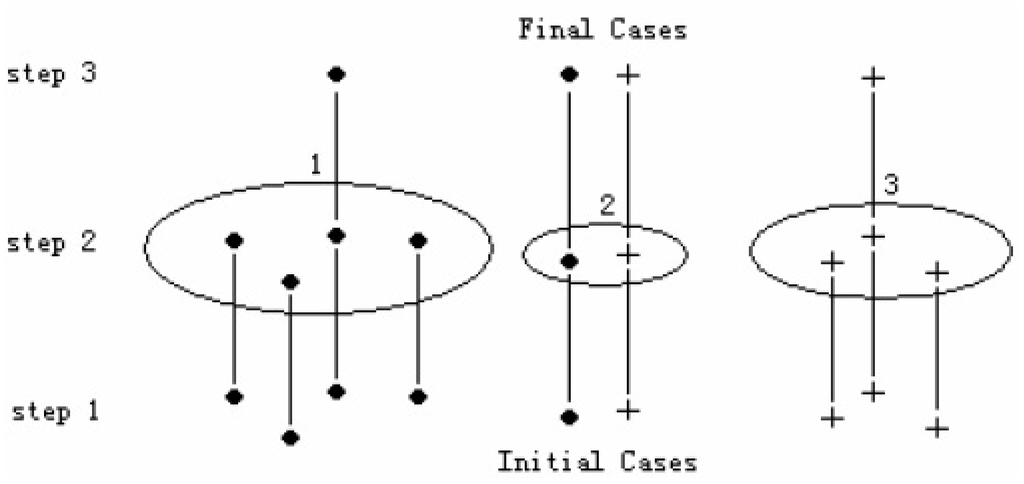


Figure 6.6: The three steps of case library construction.

$$\begin{array}{c|cccc|c|c|}
 & o_1 & o_2 & o_3 & o_4 & & SUM \\
o_1 & 0.0 & 0.1 & 0.3 & 0.15 & & 0.55 \\
o_2 & 0.1 & 0.0 & 0.2 & 0.05 & & 0.35 \\
o_3 & 0.3 & 0.2 & 0.0 & 0.12 & & 0.62 \\
\hline
o_4 & 0.15 & 0.05 & 0.12 & 0.0 & & 0.32 \\
\hline
\end{array}$$

Figure 6.7: An example of selecting the representative

can be marked as either pure or impure. For example, in Figure 6.6, clusters 1 and 3 are pure and cluster 2 is impure. A pure cluster indicates that its objects can be successfully distinguished from objects in other clusters. In contrast, an impure cluster indicates that its objects, although of different types, share some common patterns and cannot be effectively distinguished from each other. Since objects in a pure cluster are similar to each other, there is no need to compare all of them with a problem during reasoning.

Thus, in the third step, a representative of every pure cluster will be selected for entry into the case library. The method we applied to find the representatives is straightforward; we pick the one with minimal sum of distances to all other objects in the cluster. For example, a distance matrix of 4 objects in a pure cluster is given as in Figure 6.7. Because object 4 has the minimal sum of distances to the other objects, it is selected. For impure clusters, we do not make any change; each case in the cluster becomes its own case in the CBR library.

The case library for DARPA 1998 training data was constructed through the three steps described above. First, the initial cases were collected. There are 440 initial cases generated for 13 types or variants of attacks. In the second step, the supervised clustering algorithm produced 20 clusters for the initial cases when the minimal distance between clusters are 0.3. There is 1 impure cluster and 19 pure clusters, including 14 singletons (clusters having only one member). The distribution of initial cases over the clusters is shown in Table 6.1.

Note that instances of the 'warezclient' attack have been grouped into two pure clusters, 3 and 4. According to the method described in the 4, one representative from each of these two clusters was selected and entered into the case library. Two other types of attacks, 'ffb' and 'ftp_write', are processed in a similar way because all the instances of these two types

Table 6.1: Clustering results of the initial cases generated for the DARPA 1998 training data.The last column is singletons.

Type	amount	C1	C2	C3	C4	C5	C6	S
eject	5						5	
ffb	4					3		1
ffb_clear	1							1
format	1						1	
format_fail	1						1	
format_clear	1							1
ftp_write	4	2	2					
loadmodule	3						2	1
satan	8						1	7
spy	2						1	1
warez	1							1
warezclient	408			175	233			
warezmaster	1							1
Final Case	30	1	1	1	1	1	11	14

have also fallen into pure clusters. Finding representatives of pure clusters greatly decreases the number of cases necessary for reasoning. For example, to find out how close a pattern of alerts is to the 'warezclient' attack, only 2, instead of 408, need to be compared. All objects in singletons and impure clusters go directly into the case library. This produced a case library having a total of 30 final cases.

6.1.5 Comparison of Detection Performance

The DARPA 1998 testing data was used to generate problems for the CBR system. The overall results are presented in Table 6.2. The detection rate is the percentage of detected attacks in all attacks. An attack is detected if there is at least one alert generated for it. The false positive rate is the percentage of false alerts in all alerts. Snort may generate more than one alert for a session. We count all Snort alerts generated during normal sessions as false positives and all Snort alerts generated during attack sessions as true alerts. STIDE generates at most one alert for a session. Since it applies anomaly detection, a threshold is normally required to specify the condition of firing an alert. To be conservative, an alert is launched for any amount of detected anomalies. In other words, the threshold is zero for STIDE in the experiment. False negatives are undetected attacks. The number of false negatives can be computed as #attacks - #detected attacks. There are 6550 attack sessions (in a total of 8866 sessions) in the testing data. The CBR system fires 6537 alerts in which there are 6478 true alerts. Thus, the detection rate is $6478/6550 = 98.9\%$; the false positive rate is $(6537 - 6478)/6537 = 0.90\%$; the number of false negatives is $6550 - 6478 = 72$.

In our experiment, the detection threshold is set as 0.3 because this can best distinguish different clusters in training data. Problems that fall outside of the accepting zone are regarded as being normal because they do not match any known case. However, the system may miss a problem caused by a new attack or some variant of a known attack. For example, the first 'ps' attack on Monday of the first week was missed by the system. It happens that there are many types of attacks in the testing data, such as 'processtable' and 'mailbomb', that do not have any instance at all in the training data. Thus the system makes the wrong decision the first time it meets them.

We used this phenomenon to simulate updating the case library over time. A new case was entered into the case library whenever a wrong decision was made. Our experiment assumes that the update is done soon enough that the system will not make the same mistake again for identical problems. Although in practice a sequence of attacks of the same type can occur several times during a short period of time, and no update can be made on such short notice, our experiment was conducted for the purpose of demonstrating the potential of this approach.

Table 6.2: Overall results of the first experiment on DARPA 1998 data

	#alerts	Detection Rate	False Positives rate	#False Negatives
STIDE	7084	99.6%	7.9%	27
Snort	6120	21.8%	76.6%	5118
CBR	6428	98.9%	0.9%	72

From the results in Table 6.2, the system has a much better detection rate than Snort and almost the same detection rate as STIDE. It has a much lower false positives rate than either sensor. Only STIDE has a smaller number of false negatives than the CBR system. The fact that there are a large number of new attacks in the testing data contributes to most of the false negatives of the CBR system. The CBR system first missed a few new types of attacks, but after the case library was updated with these, it caught all subsequent occurrences of these attacks.

The dataset has 4999 ‘mailbomb’ attacks, and snort failed to detect any of them at all, while STIDE only missed a few of them. Thus snort has poor detection rate, while STIDE has very high detection rate.

Moreover, the CBR can indicate the exact types of most detected attacks. This is important in order to take appropriate actions in response to attacks. STIDE applies anomaly detection and thus is inherently unable to provide the detail of attacks. However, the alert correlation used in the CBR process allows the attack information to be obtained from the retrieved case. This is an important augmentation of STIDE.

Snort processes one network packet at a time and misses the context of attacks. Although it applies misuse detection, it tends to generate a large number of false alerts, overwhelming its human observers. Our results show that the CBR system can almost exactly identify the attacks, thus greatly alleviating this problem.

6.2 Experiment with DARPA 2000 datasets

The experiment with DARPA 2000 datasets was done to evaluate the approach of case-oriented alert correlation using session information. Like the experiment with DARPA 1998 datasets, it assumes that the case-based meta IDS is set up to protect a particular host. The results have demonstrated that the alert correlation approach helps detect the given variant of a distributed denial of service (DDoS) attack.

6.2.1 Datasets Description

The 2000 DARPA intrusion detection scenario specific datasets include two sets entitled LLDoS 1.0 and LLDoS 2.0.2. LLDoS 1.0 contains a series of attacks in which an attacker probes, breaks in, installs the components necessary to launch a DDoS attack, and actually launches a DDoS attack against an off-site server. LLDoS 2.0.2 includes a similar sequence of attacks run by an attacker stealthier than the first one. The datasets come with both low-level raw data and mid-level labeled data. The low-level data consists of DMZ tcpdump data, inside tcpdump data, and BSM audit data from two Solaris machines, Pascal and Mill. The mid-level data consists of XML files containing IDMEF alerts generated by sensors. As the experiment is concerned with meta intrusion detection, it was done only with mid-level data, namely IDMEF alerts.

6.2.2 Experiment Setup

In this experiment, LLDoS 1.0 was used to construct the case library. Then the CBR system was tested using the data in LLDoS 2.0.2. This is appropriate since LLDoS 2.0.2 is a variant of LLDoS 1.0.

The LLDoS 1.0 attack scenario is carried out over multiple network and audit sessions. These sessions have been grouped into 5 attack phases, namely IP sweeping, probing, breaking in, installing DDoS software, and launching attacks, as shown in Figure 6.8(http:www.ll.mit.eduISTidevaldata20002000_data_index.html). A case corresponding to each attack phase is created for each of Pascal and Mill. All alerts that took place during an attack phase and were associated with a particular host form a case for the host. The datasets provide each host with IDMEF alerts from three sources: DMZ, inside network, and the host itself. Accordingly, a case may have alerts from at most three sensors. Two case libraries were constructed for Pascal and Mill, respectively. Each has five cases corresponding to the five different attack phases.

6.2.3 Experiment Results

Since there is no redundancy in this experiment, clustering was not performed. The experiment applied case-oriented alert correlation. The correlated alerts for a particular case form a description of a problem representing a potential attack. The results are presented in Table 6.3.

The results have demonstrated that our system is able to effectively detect the attack described by LLDoS 2.0.2, especially on Mill. However, the first phase of the attack may not be easily recognized. The constructed problems for case-phase-1 on both machines have relatively large distances. This is because the attackers in LLDoS 1.0 and LLDoS 2.0.2 used different ways to find victim hosts. The attacker in LLDoS 1.0 performed a scripted IPsweep to find 'live' IP addresses first and then sent 'sadmind' requests to find out possible victim hosts. The attacker in LLDoS 2.0.2 only performed an HINFO query of a public DNS server that directly returned possible victim hosts. Once the victims have been found, the

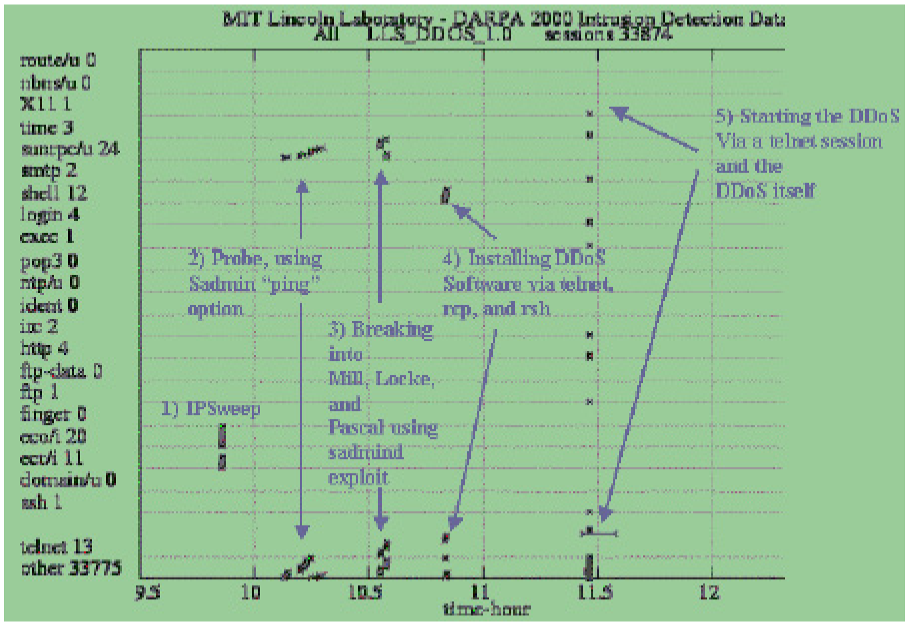


Figure 6.8: Service Plot for DDoS1.0

rest of both attacks is almost the same. This is why the constructed problems show better similarities to their corresponding cases.

There is another important difference between LLDoS 1.0 and LLDoS 2.0.2. In LLDoS 1.0, Pascal and Mill were attacked from outside of the network. As a result, alerts were generated from both the DMZ and inside the network. In contrast, in LLDoS 2.0.2, the attacker broke into Mill and fanned out from there. Mill is inside the network. The attacks from Mill to other hosts inside are invisible to the sensor at the DMZ. So, Pascal didn't accept any alert from the DMZ sensor in LLDoS 2.0.2. This difference made problems on Pascal have less similarity to their cases than problems on Mill. But if the appropriate threshold is set, say 0.4, most of the attack phases can be detected. More specifically, on Mill, all attack phases except phase 1 will be identified, and on Pascal, all attack phases except 1 and 4 will be identified.

6.3 Experiments with DARPA Cyber Panel Program

The Cyber Panel Grand Challenge Problem (GCP)[85] program is an attack simulator. The experiment with it was conducted to evaluate the approach of enhanced case-oriented

Table 6.3: Overall results of the first experiment on DARPA 2000 data

Pascal		Mill	
Cases in the case library	Distance between case and constructed problem	Cases in the case library	Distance between case and constructed problem
$phase_1$	0.67	$phase_1$	0.5
$phase_2$	0.389	$phase_3$	0.139
$phase_3$	0.389	$phase_3$	0.218
$phase_4$	0.425	$phase_4$	0.233
$phase_5$	0.389	$phase_5$	0.141

alert correlation. Unlike the experiments with the DARPA 1998 and 2000 datasets, this experiment assumes the meta IDS is set up to protect a corporation on the large scale. All the cases and problems for the experiment came from the attack scenarios simulated by the program. The experimental results have shown the enhanced case-oriented alert correlation (the dynamic programming solution for order-preserving matching) is a promising approach.

6.3.1 Description of Ground Challenge Problem

The GCP is a notional, cyber-defense scenario that describes a commercial corporation under strategic coordinated attack. The scenario exhibits the primary characteristics of many large, network centric organizations. The role of GCP is to:

- Exercise and validate cyber-defense such as (a) correlation, (b) situation assessment, and (c) response (human-directed and automatic).
- Provide a common tool-suite that can be used to demonstrate opportunities for collaboration and integration, and to achieve repeatable demonstrations. The suite can be expanded and varied over time to explore different adversaries, network topologies, sensor type, sensor replacement, and defensive strategies.
- Provide common data sets and simulation tools that support core capability demonstrations and technology validation.

The GCP scenario describes three strategic, coordinated cyber attacks that are conducted against Global. These attacks exemplify typical attack classes: life-cycle, insider, or distributed denial of service of attacks.

The global corporate structure has 5 different types of network sites, namely Headquarters (HQ), Air Planning Center (APC), corporate transport Ships, Air Transport Hubs (ATH), and Partner Air Transport Hub (PATH). Each site is defended by typical sensors, as well as postulated high-level sensors. There are in total 10 types of sensors in the networks. All the sensors generate the alerts in IDMEF messages. Simulation tools allow researchers to configure a network of arbitrary size and observe the alert streams that would be generated by sensors when attacks are perpetrated.

6.3.2 Experiment Setup

Running an attack scenario with the GCP simulator requires providing the following inputs:

- Specification of the number of sites of type SHIP, ATH, and PATH. This sets the size of the corporation being attacked.
- Three arbitrarily chosen integer, evidently used as seeds for random number generators applied during the attack simulation.
- Specification of the type(s) of attack. There are three choices, A1, A2, and A3. One can select any combination of these, e.g., one can simulate having simultaneous attacks of types A1 and A3.
- Specification of whether the attacks should, or should not, be accompanied with randomly generated background attacks (noise).

We experimented with a number of different site configurations, represented here as triples. To illustrate, $(1, 1, 1)$ indicates one site of each type, SHIP, ATH, and PATH, $(1, 5, 10)$ indicates one of type SHIP, 5 of type ATH, and 10 of type PATH. For each such configuration, we built a simple case library as follows. For each attack type, A1, A2, A3, we ran one attack of this type without background noise. The alerts generated for each attack then became the problem part of a case, and the identity of at the attack type formed the solution part of the case. Here is should be noted further that the simulator allows attacks of type A3 (distributed denial of service) to be generated only for configurations having one site of each type. Thus for configuration $(1, 1, 1)$, the case library has 3 cases, one for each of A1, A2, A3. For all other configurations, the case library has only two cases, one for each of A1 and A2. Different random number seeds were chosen for each run.

Attacks were then run for each configuration, with background noise, and the alert streams were collected. These attacks are referred to as “problems”. The alert streams were then input to our case-based reasoner, once using the Hungarian algorithm, and once using dynamic programming, as the underlying matching technique.

6.3.3 Experimental Results

Table 6.4: Experimental results with (1,1,1)

	C1(A1)		C2(A2)		C3(A3)	
	H	D	H	D	H	D
P1(A1)	0	0	0.5386	0.5484	0.3078	0.3078
P2(A2)	0.7828	0.7847	0.0185	0.0185	0.3078	0.3078
P3(A3)	0.8126	0.8137	0.5697	0.5697	0	0
P4(A1,A2)	0	0	0.0185	0.0185	0.3078	0.3078
P5(A1,A3)	0	0	0.5002	0.51070	0	0
P6(A2,A3)	0.7798	0.7818	0.0185	0.0185	0	0
P7(A1,A2,A3)	0	0	0.0185	0.0185	0	0

Table 6.5: Experimental results with (5,5,5)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0577	0.0595	0.5030	0.5047
P2(A2)	0.8943	0.8951	0.3327	0.3327
P3(A1,A2)	0.0588	0.0576	0.3001	0.3018

The results of these experiments are shown in Tables 6.4 through 6.8. In Table 6.4, the configuration $(1, 1, 1)$ was used. The column heading C1(A1) indicates library case 1, created using an attack of type 1. Similarly for the other columns. The secondary column headings

Table 6.6: Experimental results with (10,10,10)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0298	0.0298	0.5002	0.5107
P2(A2)	0.9449	0.9454	0.3282	0.3293
P3(A1,A2)	0.0298	0.0302	0.2973	0.3079

Table 6.7: Experimental results with (1,5,10)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0	0	0.5002	0.5107
P2(A2)	0.8596	0.8607	0.3282	0.3293
P3(A1,A2)	0	0	0.2973	0.3079

Table 6.8: Experimental results with (10,5,1)

	C1(A1)		C2(A2)	
	H	D	H	D
P1(A1)	0.0563	0.0576	0.5030	0.5047
P2(A2)	0.9304	0.9311	0.0185	0.0185
P3(A1,A2)	0.0822	0.0949	0.2973	0.3079

“H” and “D” indicate results using the Hungarian algorithm and dynamic programming, respectively. The row labels indicate the problems. To illustrate, problem P1 involved an attack of type A1, and problem P4 involved simultaneous attacks of types A1 and A2.

The numbers in the cells are the computed distances between the given problem and case as determined by the indicated matching technique. Thus, the 0’s for both H and D under case 1, for problem 1, represent an exact match, i.e., the problem represented by the case is reported to have occurred. The other cells in this table, as well as those in the other tables, are interpreted similarly.

From these tables, we can draw a few general conclusions. The closeness of the results for the Hungarian algorithm based case-oriented alert correlation and the dynamic programming based cased-oriented alert correlation approaches demonstrate that they are equally good in correlating alerts. However, the dynamic programming approach has much less complexity. The complexity of the Hungarian algorithm is given as $O((m+n)^3)$ for a m by n cost matrix. In contrast, with the same m by n cost matrix (suppose $n < m$), the complexity of the dynamic programming approach is $O(n \times (m-n+1))$ because it only has to fill a cache table with $n \times (m-n+1)$ cells in order to find the solution. Thus, the dynamic programming method is a more efficient solution for case-based alert correlation.

Given a problem and a case, one of two situations is possible, either the problem is represented by the case, or it is not. Whether the problem is represented by the case can be determined by means of a threshold on the distance between that problem and case. If the distance exceeds the threshold, the problem can be determined as not matching the case, and if the the distance is smaller than the threshold, is does match the case, i.e., the attack represented by the case can be assumed to have occurred.

One can determine such a threshold for each case as follows. We illustrate this for case C1 where the matching used the Hungarian algorithm. We scan through all the tables for the smallest distance between a problem and C1, where the problem does not match C1. Given the above tables, this is 0.7798, the distance between problem P6 and C1 in Table 1. We next scan through all the tables for the largest distance between a problem and C1, where the problem does match C1. This is 0.0588, the distance between problem P3 and C1 in Table 6.5.

Thus any value in the range $[0.0588, 0.7798]$ could be an acceptable threshold. It is desirable, however to have this be as unsusceptible as possible to noisy data or variants of known attacks. A straightforward and practical approach is to use the median of the interval as the threshold. Thus, for cases representing attack A1, the threshold is given as $(0.7798 + 0.0588)/2 = 0.4193$.

Using this same technique one can compute a threshold for cases representing attack A2 as $(0.5002 + 0.3283)/2 = 0.4142$, and a threshold for cases representing attack A3 as $(0.3078+0.0)/2 = 0.1539$. Since the table values corresponding to the dynamic programming matching method are essentially identical to those based on the Hungarian algorithm, we will restrict our discussion to just those involving the latter.

As can be seen from the tables, these threshold values effectively identify the attacks contained in all the given problems. Thus these results suggest that the overall technique can be effective. One could, if one wished, add further tables, based on further network configurations, to the mix, but the present choice of configurations is sufficiently varied that such further additions will not significantly change the threshold values from those computed here.

To further validate our method for computing thresholds, one can apply the well-known leave-one-out analysis technique. To illustrate, let us first leave out problem P1 in Table 1. This means that we recompute the thresholds using all the values in the tables except those in the row P1 of Table 6.4. We then see if these values can be used to effectively identify the attack present in P1 of Table 6.4. It will be seen that it does. (Coincidentally, in this case the new computation gives the same three threshold values as computed above.) This is then repeated for every problem (row) in every table. We found that in all cases the method succeeds; the newly computed thresholds effectively identify the attacks present in each problem.

Note that a large threshold tends to make false positives and a small threshold tends to make false negatives. Thus, if in some application it is determined that false positives are less desirable that false negatives, one can lower the threshold values by some appropriate amount. Conversely, if false negatives are less desirable that false positives, one can raise the thresholds.

CHAPTER 7

CONCLUSION

In this chapter, we summarize the thesis, review the thesis contribution, and discuss drawbacks as well as future work.

7.1 Summary

This thesis has described a case-based reasoning framework for meta intrusion detection. The framework defines the cases and problems for this particular domain and makes intrusion detection part of the case-based reasoning process. A distinguishing feature that makes our approach different from other case-based reasoning applications is that cases and problems are described in XML. In addition, this framework includes a distance-based XML similarity measure, data mining approaches, and alert correlation approaches. Using the framework, a case library can be constructed manually or automatically from training data, then refined by data mining approaches, and finally compared with problems formulated via alert correlation approaches for intrusion detection.

We first motivated our thesis by stating the importance of intrusion detection in the overall computer security mechanisms. We provided background on intrusion detection systems, and briefly described representative intrusion detection systems. We pointed out that traditional intrusion detection systems suffer from a large number of mistakes mainly because they apply a single data source and a single detection model. They also lack effectiveness, adaptability, and extensibility. The concept of meta intrusion detection suggests combing the results from other intrusion detection systems or security devices in order to reduce the number of overall mistakes. We observed that meta intrusion detection is very similar to diagnostic problems, which are considered as the most successful application domains of case-based reasoning. The goal of this thesis research is therefore to explore the feasibility if it is also applied to meta intrusion detection.

We studied the common characteristics of case-based reasoning applications. We discussed that for a case-based reasoning application, the most important issue is to find a set of features to describe cases and problems. In the domain of meta intrusion detection, the information for the construction of cases and problems is alerts that are described by IDMEF, an XML language. The inherent flexibility of IDMEF makes it almost impossible to predefine a set of features that can cover all important information in alerts. Therefore, instead of using features, cases and problems are described straightforwardly in XML in order to avoid information loss. Although this design deviates from traditional approaches, it does provide a convenient way to build cases and problems for our domain and thus simplifies knowledge acquisition and representation.

We then studied the problem of similarity measures for case-based reasoning systems. Nearly all traditional similarity measures work with data described by a set of predefined features. As we apply XML representations for cases and problems, traditional similarity measures are not suitable for our needs. Thus, we developed a novel XML distance measure as the similarity measure for our case-based reasoning system. It is able to take any two arbitrary XML trees as input and recursively compute their overall distance from roots to leaves. Our experiments showed that this distance measure could reasonably measure the similarity between XML objects. In fact, our whole case-based approach for meta intrusion detection relies heavily on the effectiveness of this XML distance measure. Besides the similarity measure of the case-based reasoning system, data mining approaches and alert correlation approaches in our framework are all based on the XML distance measure.

We next studied the problem of constructing a case library. A case library embodies the expertise of a case-based reasoning system. It should be representative and precise. As the size of the case library grows over time, it makes the case-based reasoning system slower to respond to problems. We proposed a general three-step approach to construct a case library from training data. In particular, a supervised clustering algorithm was designed to find similar cases. The case library can be refined by replacing similar cases with their representatives. Our experiment on the DARPA 1998 datasets has demonstrated the effectiveness of this approach. The size of the case library was significantly reduced without affecting the detection performance. A useful item of information implicit in the results of the clustering algorithm is the distance that can best distinguish different types of cases in the case library. This can be used as the threshold for case-based reasoning systems for making decisions.

We next studied how to formulate problems for our case-based reasoning system. Problems stand for unknown situations. In our application domain, they are potential attacks and implicit in the alerts from different sensors. The issue is how to find them. This is a common difficulty in meta intrusion detection research and is generally referred to as alert correlation. It is concerned with determining the logical connection between alerts. The adoption of case-based reasoning makes the problem of alert correlation more difficult because knowledge used by experts to solve problems is implicit in cases rather than being encoded as rules. We developed two categories of alert correlation approaches, explicit alert correlation and implicit alert correlation. The former takes advantage of some essential features in alerts, such as IP addresses and ports, which can reveal the connections between alerts. The latter is called case-oriented alert correlation, a more general approach, especially useful when alerts do not contain those essential features for correlation. It correlates runtime alerts according to the patterns of alerts in cases. If the most similar pattern of alerts found from runtime alerts is similar enough to the pattern of alerts in a given case, it is reasonable to believe an attack represented by the case has likely taken place. Our experiments have shown both approaches can effectively formulate problems for case-based reasoning systems. We also devised a variant of the original case-oriented alert correlation approach. The new one is more useful in practice because it is as good as the old one and has much less computational complexity.

We are currently studying the problem of how to implement a real-time meta intrusion detection system on a cluster of processors using our case-based approach. The main idea is using the cluster as the storage and processing center. The cluster is allowed to receive alerts from various sensors simultaneously through parallel channels. By taking advantage of its computation power, some tasks, such as sorting alerts and correlating alerts, are broken into subtasks that can be assigned to different processors running in parallel within the cluster.

7.2 Dissertation Contributions

We recap the dissertation contributions:

- **Case-Based Meta Intrusion Detection** We studied the problem of meta intrusion detection. We modeled it as a diagnosis process on computer system and developed a case-based framework for it.

- **Techniques for Alert Correlation** We studied the common issue, alert correlation in meta intrusion detection and devised a new category of approaches, namely case-oriented alert correlation. Basically, we reduce alert correlation to assignment problems and apply matching algorithms to find best solutions.

- **Utilization of XML Representation for Data in Analysis** We studied the drawbacks of traditional attribute-value representations. We proposed preserving the XML forms of data for analysis in order to avoid information loss. Objects in our case-based reasoning system all take the form of XML. Data analysis algorithms over data in terms of XML representation were developed accordingly.

- **The XML Distance Measure** We studied the problem defining similarity measure between two arbitrary XML documents for the purpose of our case-based reasoning system. A novel distance-based similarity was devised. It plays a significant role in our whole approach. With this distance measure, other tasks of data analysis can be relatively easily carried out.

- **Ordered Minimal Matching** As part of the design of case-oriented alert correlation, we put an ordering constraint to the traditional assignment problems. We developed a new matching algorithm to find ordered minimal matching based on the theory of dynamic programming.

7.3 Drawbacks

As a coin has two sides, there are drawbacks inherent in the adoption of case-based reasoning for intrusion detection.

First, it can not be applied to build attack scenarios. Approaches for building attack scenarios help people better understand attacks so that more sophisticated countermeasures can be taken accordingly. Although case-based reasoning simplifies the knowledge acquisition and representation, the expertise, namely the library of past cases, is not well-defined knowledge anyway. Even thought we can make use of the information about past cases for intrusion detection, those cases can not tell how exactly the attacks took place.

An other drawback is that this approach is subject to changes in the working environments. The case library contains specific information regarding the system and sensors, such as the IP address of the sensors. If the configuration or the topology of the system is changed, the new responses of sensors to attacks may be different from their old ones. The case library then needs to be updated according to sensors' new behaviors, which may make the intrusion detection system unfriendly to use.

7.4 Future Work

There are several interesting and important future directions:

- **Reducing Noise in Cases** A pattern of alerts is the major part of a case. Patterns of alerts in cases are the most important information for intrusion detection. The quality of patterns of alerts is critical for intrusion detection. It is possible that some alerts in the pattern may be false alerts. Finding and removing those false alerts can certainly enhance the quality of the case library.

- **Using Attack Simulator for the Construction of Case Library** We mentioned one drawback in the previous section, that changes in the working environments requires that the case library be updated according to the new responses of sensors to attacks. A good way to construct the case library is to use attack simulator programs that can simulate different network topologies and configurations. If the real network has changed, the simulator programs can simulate reactions of sensors to attacks in the new environment. New cases can be collected for the construction of the new case library.

- **The Design of Case Library for Fast Case Retrieval** Our present approach of retrieving case is like the 1-nearest neighbor algorithm, which needs to compute distances of a given problem to all the cases in order to find the most similar one. If the size of the case library becomes very large, it would be very time-consuming to go through all of them. Even if we can apply the data mining approach to refine the case library, it would be better to organize the cases somehow to speed up the retrieval of cases. A good way to organize cases is by a tree model in which each node represents a certain important feature of cases and where leafs are cases. All cases having the same feature would be collected in the sub-tree of the node representing the feature. Typical features that can be used for this purpose include the number of alerts, the number of sensors, and time frame.

7.5 Closing Remarks

This dissertation documented our research in developing and applying case-based reasoning techniques to the important and challenging problem of meta intrusion detection. A set of algorithms and approaches have been designed, implemented, and evaluated for this dissertation research. While it has shown good promises, there are open problems for future research.

APPENDIX A

Hungarian Algorithm

The Hungarian algorithm is also referred as Munkres' algorithm [59]. The original version is only applied to find solutions in square matrices. Two definitions are given here to help understand the Munkres' algorithm.

- **Definition 1**: Two matrices are equivalent if they have the same optimal assignment solution. It means the algorithm gives the same result.
- **Definition 2**: A set of elements of a matrix are independent if none of them occupies the same row or column.

Munkres' algorithm can be briefly stated as finding a new matrix equivalent to the initial one with n independent zero elements. In the beginning of the algorithm, the initial matrix is transformed into an equivalent matrix by (a) for each row subtracting the value of the smallest element from each element in the row and (b) for each column of the resulting matrix subtracting the value of smallest element from each element. The following steps are:

- **Step 1**: Find a zero, Z, of the matrix. If there is no starred zero in its row or its column, star Z. Repeat for each zero of the matrix. Go to step 2.
- **Step 2**: Cover every column containing a 0*. If all columns are covered, the starred zeros form the desired independent set; Exit. Otherwise, go to step 3.
- **Step 3**: Choose a noncovered zero and prime it; then consider the row containing it. If there is no starred zero Z in this row, go to step 4. If there is a starred zero Z in this row, cover this row and uncover the column of Z. Repeat until all zeros are covered. Go to step 5.

- **Step 4**: There is a sequence of alternating starred and primed zeros constructed as follows: let Z_0 denote the uncovered $0'$. Let Z_1 denote the 0^* in Z_0's column (if any). Let Z_2 denote the $0'$ in Z_l's row. Continue in a similar way until the sequence stops at a $0'$, Z_{2k}, which has no 0^* in its column. Unstar each starred zero of the sequence, and star each primed zero of the sequence. Erase all primes and uncover every line. Return to step 2.

- **Step 5**: Let h denote the smallest noncovered element of the matrix; it will be positive. Add h to each covered row; then subtract h from each uncovered column. Return to step 3 without altering any asterisks, primes, or covered lines.

F. Bourgeois and J. Lassalle developed an extension of this algorithm which permits a solution for rectangular matrices [86]. Specifically, given an arbitrary $n \times m$ matrix, let $k = min(n, m)$. If $n > m$, go to step 0. For each row, subtract the smallest element from each element in the row. If $n < m$, go to step 1.

- Step 0. For each column of the resulting matrix, subtract from each entry the smallest entry in the column.

- Step 1. Remains the same.

- Step 2. Cover every column containing a 0^*. If k columns are covered, the starred zeros form the desired independent set. Otherwise, go to step 3.

- Steps 3 to 5. Remain the same.

REFERENCES

[1] A. Maccabe R. Heady, G. Luger and M. Servilla. The architeture of a network level intrusion detection system. Technical report, University of New Mexico, Computer Science Department, Auguest 1990. 1

[2] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, 1980. 1

[3] G. Vigna and R. A. Kemmerer. NetSTAT: A network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, 1999. 1

[4] H. Javits and A. Valdes. The NIDES statistical component: Description and j ustification. Technical Report SRI Anual Report A010, SRI International, Computer Science Laboratory, March 1993. 1

[5] A. Somayaji S. Forrest, S. A. Hofmeyr and T. A. Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, Los Alamitos, California, May 1996. IEEE Computer Society Press. 1, 6.1.2

[6] S. Forrest S. A. Hofmeyr and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998. 1, 6.1.2

[7] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *Proceedings of the 17th Annual Computer Security Application Conference*, page 22. IEEE Computer Society, 2001. 1, 2, 2.4, 4.5.1, 5.1

[8] M. W. Fong P. A. Porras and A. Valdes. A mission-impact-based approach to INFOSEC alarm correlation. In *Lecture Notes in Computer Science, Proceedings Recent Advances in Intrusion Detection*, pages 95–114, Zurich, Switzerland, October 2002. 1, 2.4, 5.1

[9] S. Templeton and K. Levitt. A requires/provides model for computer attacks. In *New Security Paradigms Workshop, Proceedings of the 2000 workshop on New security paradigms*, Ballycotton, County Cork, Ireland, October 2001. ACM Press. 1, 2.4, 5.1

[10] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Lecture Notes In Computer Science, Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, number 2212, pages 54–68. Springer-Verlag, 2001. 1, 2.4, 4.5.1, 5.1

[11] S. Y. Ho. Intrusion detection - systems for today and tomorrow. Technical report, September 2001. 2

[12] P. Loshin. New direction in intrusion detection. Technical report, 2001. 2

[13] F. Cuppens and A. Miege. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202, Oakland, CA, 2002. IEEE Computer Society. 2, 2.4, 2.5, 4.5.1, 5.1

[14] D. E. Denning. An intrusion-detection model. *IEEE Transaction on Software Engineering*, SE-13(2):222–232, 1987. 2.1

[15] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999. 2.1

[16] R. G. Bace. *Intrusion Detection*. Macmillan Computer Publishing, Indianapolis, IN, 2000. 2.1

[17] S. Staniford-Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework. Technical report, Orlando, FL, October 1998. 2.1

[18] C. Kahn, P. Porras, S. Staniford-Chen, and B. Tung. A common intrusion detection framework. *Journal of Computer Security*, July 1998. 2.1

[19] T. Lunt. IDES: An intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*, Rome, Italy, November 1990. 2.4

[20] P. Porras and P. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of 20th NIST-NCSC National Information Systems Security Conference*, pages 353–357, 1997. 2.4, 4.2

[21] P. Chan and S. Stolfo. Toward parallel and distributed learning by meta-learning. In *AAAI Workshop in Knowledge Discovery in Database*, pages 227–240, 1993. 2.4

[22] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining*, pages 74–81, 1997. 2.4

[23] K. Julisch. Mining alarm clusters to improve alarm handling efficiency. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 12–21. IEEE Computer Society, 2001. 2.4, 4.5.1, 4.5.3, 5.1

[24] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Lecture Notes In Computer Science, Proceedings of the 4th International Symposium on Recent Advance in Intrusion Detection*, pages 85–103. Springer-Verlag, 2001. 2.4, 3.5, 5.1

[25] B. Morin and D. Debar. Correlation of intrusion symptoms: an application of chronicles. In *Lecture Notes In Computer Science, Proceedings of the 6th International Symposium on Recent Advance in Intrusion Detection*, pages 94–112. Springer-Verlag, 2003. 2.4, 2.5, 5.1

[26] P. Ning, Y. Cui, and D. S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, Washington, D.C., November 2002. 2.4, 2.5, 3.5, 4.5.1, 5.1

[27] B. Morin, L. Me, H. Debar, and M. Ducasse. Correlation of intrusion symptoms: an application of chronicles. In *Lecture Notes In Computer Science, Proceedings of the 5th International Symposium on Recent Advance in Intrusion Detection*, pages 115–137. Springer-Verlag, 2002. 2.4, 3.5, 5.1

[28] 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 *Proceedings of the 19th National Information Systems Security Conference*, pages 361–370, Baltimore, MD, March 1996. 2.4

[29] J. Long, S. Stoecklin, D. G. Schwartz, and M. Patel. Adaptive similarity metrics in case-based reasoning. In *Proceedings of The 6th IASTED International Conference on Intelligent Systems and Control*, pages 260–265, 2004. 3

[30] C. Riesbeck and R. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum, Hillsdale, NJ, 1989. 3.1

[31] K. Althof, E. Auriol, R. Barlette, and M. Manago. A review of industrial case-based reasoning tools. *AI Intelligence*, 1995. 3.1

[32] A. Bundy. *Artificial Intelligence Techniques: A Comprehensive Catalogue*. Springer-Verlag New York, Inc, 4th edition, 1995. 3.1

[33] M. Halasz, F. Dude, R. Orchard, and R. Ferland. The integrated diagnostic system : Remote monitoring and decision support for commercial aircraft - putting theory into practice. In *Proceedings of AAAI '99 Spring Symposium on AI in Equipment Maintenance*, Palo Alto, CA, 1999. NRC. 3.3

[34] D. G. Schwartz, S. Stoecklin, and E. Yilmaz. A case-based approach to network intrusion detection. In *Proceedings of 5th International Conference on Information Infusion*, pages 1084–1089, Annapolis, MD, 2002. 3.4, 4.5.1, 6.1.3

[35] J. W. Yoder, F. Balagular, and R. Johnson. Architecture and design of adaptive object models. In *ACM SIGPLAN Notices*, volume 36, pages 50–60. ACM Press, 2001. 3.4.1

[36] M. Esmaili, B. Balachandran, R. Safavi-Naini, and J. Pieprzyk. Case-based reasoning for intrusion detection. In *Proceedings of the 12th Annual Computer Security Applications Conference*, page 214, Washington, DC, 1996. IEEE Computer Society. 3.5, 4.2

[37] M. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press 1996, 1996. 4

[38] W. Lee. *Ph. D. Thesis: A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems.* Computer Science Department, Columbia University, New York, NY, June 1999. 4.2, 5.2

[39] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of The 2001 IEEE Symposium on Security and Privacy*, page 130, Oakland, CA, 2001. IEEE Computer Society. 4.2, 6

[40] A. Valdes and K. Skinner. Adaptive, model-based monitoring for cyber attack detection. In H. Debar, L. Me, and F. Wu, editors, *Lecture Notes In Computer Science, Proceedings of the 3th International Symposium on Recent Advance in Intrusion Detection.* 4.2

[41] K. Fox, R. Henning, J. Reed, and R. Simonian. A neural network approach towardsintrusion. In *Proceedings 13th NIST-NCSC National Computer Security Conference*, pages 124–134, Baltimore MD, 1990. 4.2

[42] A. K. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against program. In *Proceedings of the 14th Annual Computer Security Applications Conference*, page 259, Baltimore MD, 1998. IEEE Computer Society. 4.2

[43] W. Lee, S. Stolfo, and P. Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997. 4.2

[44] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Lake Taho, CA, 1995. Morgan Kaufmann. 4.2

[45] S. M. Bridges and B. B. Vanghn. Fuzzy data mining and genetic algorithms applied to intrusion detection. In *Proceedings of the 23th National Information System Security Conference*, 2000. 4.2

[46] N. Ye and X. Li. A scalable clustering technique for intrusion signature recognition. In *Proceedings of the 2nd IEEE SMC Information Assurance Workshop*, pages 1–4, West Point, NY, June 2001. 4.2

[47] D. Barbara, Y. Li, J. Couto, J. Lin, and S. Jajodia. Bootstrapping a data mining intrusion detection system. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 421–425, Melbourne, Florida, 2003. ACM Press. 4.2

[48] N. Lavrac and S. Dzeroski. *Inductive Logic Programming Techniques and Applications.* Ellis Horwood, New York, NY, 1994. 4.2

[49] C. Ko. Logic induction of valid behavior specifications for intrusion detection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 142–155. IEEE Computer Society, 2000. 4.2

[50] J. Long, D. G. Schwartz, and S. Stoecklin. An XML distance measure. In *Proceedings of the 2005 International Conference on Data Mining*, pages 119–125, 2005. 4.3

[51] S. Dzeroski and N. Lavrac. *Relational Data Mining*. Springer, Berlin, 2001. 4.3

[52] D. Wettschereck and D. Aha. Weighting features. In *Proceedings of 1st International Conference on Case-Based Reasoning*, pages 347–358, Springer, Berlin, 1995. 4.3, 4.5.3

[53] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley and Sons, 1990. 4.3

[54] U. Bohnebeck, T. Horvath, and S. Wrobel. Term comparisons in first-order similarity measures. In *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 65–79, London, UK, 1998. Springer-Verlag. 4.3

[55] M. Kirsten and S. Wrobel. Relational distance-based clustering. In *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 261–270. Springer-Verlag, 1998. 4.3

[56] M. Kirsten and S. Wrobel. Extending k-means clustering to first-order representations. In *Proceedings of the 10th International Conference on Inductive Logic Programming*, pages 112–129. Springer-Verlag, 2000. 4.3

[57] M. Sebag. Distance induction in first order logic. In *Lecture Notes in Artificial Intelligence, Proceedings of the 7th International workshop on inductive logic programming*, pages 264–272. Springer-Verlag, 1997. 4.3

[58] A. Hutchinson. Metrics on terms and clauses. In *Lecture Notes In Computer Science,Proceedings of the 9th European Conference on Machine Learning*, pages 138–145. Springer-Verlag, 1997. 4.3

[59] J. Munkres. Algorithms for the assignment and transportation problems. *Journal Of SIAM*, 5(1):32–38, March 1957. 4.3, 4.3.4, A

[60] A. Nierman and H. V. Jagadish. Evaluation structural similarity in XML documents. In *Proceedings of the 5th International Workshop on the web and databases*, Madison, WI, 2002. 4.3.1

[61] J. Munkres. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, 1966. 4.3.1

[62] S .Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of ACM SIGMOD*, pages 493–504. ACM Press, 1996. 4.3.1

[63] K. Zhang, R. Stgatman, and D. Shasha. Simple fast algorithm for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989. 4.3.1

[64] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for XML documents. In *Proceedings of the 2003 Workshop on Knowledge and Experience*, 2002. 4.3.1

[65] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between XML documents. In *Proceedings of 5th International Workshop on the Web and Databases*, 2002. 4.3.1

[66] E. Bertino, G. Guerrini, and M. Mesiti. A matching algorithm for measuring the structural similarity between an xml document and a dtd and its applications. *Information Systems*, 29(1):23–46, 2004. 4.3.1

[67] G. Canfora, L. Cerulo, and R. Scognamiglio. Measuring XML document similarity: a case study for evaluating information extraction systems. In *Proceedings 10th International Symposium on Software Metrics*, pages 36–45, 2004. 4.3.1

[68] J. R. Ullman. An algorithm for sub-graph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976. 4.3.1

[69] P. Dopichaj. Exploiting background knowledge for better similarity calculation in XML retrieval. In *21st Annual British National Conference on Databases, Doctoral Consortium*, July 2004. 4.3.1

[70] H. G. Kayacik and A. N. Zincir-Heywood. A case study of three open source security management tools. In *Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management*, pages 101–104, 2003. 4.5

[71] L. Wu and S. Chen. Wbuilding intrusion pattern miner for snort network intrusion detection system. In *In Proceedings IEEE 37th Annual 2003 International Carnahan Conference on Security Technology*, pages 477–484, IEEE Computer Society, 2003. 4.5.1

[72] C. J. Coit, S. Staniford, J., and McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *Proceedings of DARPA Information Survivability Conference and Exposition II*, pages 367–373, Anaheim, CA, 2001. 4.5.1

[73] I. Sourdis and D. Pnevmatikatos. Pre-decoded CAMs for efficient and high-speed NIDS pattern matching. In *Proceedings of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 258–267. IEEE Computer Society, 2004. 4.5.1

[74] R. Liu, N. Huang, C. Kao, C. Chen, and C. Chou. A fast pattern-match engine for network processor-based network intrusion detection system. In *Proceedings of 2004 International Conference on Information Technology: Coding and Computing*, pages 97–101. IEEE Computer Society, 2004. 4.5.1

[75] F. Yu and R. H. Katz. Efficient multi-match packet classification with TCAM. In *Proceedings of the 12th Annual IEEE Symposium on High Performance Interconnects*, pages 28–34. IEEE Computer Society, 2004. 4.5.1

[76] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptive neuro-fuzzy intrusion detection systems. In *Proceedings of Information Technology: Coding and Computing*, pages 70–74, Las Vegas, NE, 2004. IEEE Computer Society. 4.5.1

[77] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, (4):443–471, 2003. 4.5.1, 4.5.3

[78] P. Ning, Y. Cui, D. Reeves, and D. D. Xu. Techniques and tools for analyzing intrusion alerts. *ACM Transactions on Information and system security*, 7(2):274–318, 2004. 4.5.1

[79] J. McQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on mathematics, Statistics and Probability*, pages 281–298, 1967. 4.5.3

[80] A. Aho, J. Hopcroft, and J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading MA, 1983. 5.4.3

[81] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math*, 1:269–271, 1959. 5.4.3, 5.4.4

[82] R. Luus. *Iterative Dynamic Programming*. CRC Press, 2000. 5.4.4

[83] J. Long, D. G. Schwartz, and S. Stoecklin. Application of case-based reasoning to multi-sensor network intrusion detection. In *Proceedings of WSEAS/IASMEInternational Conference on Computational Intelligence, Man-Machine Systems, and Cybernetics*, pages 260–269, 2005. 6

[84] J. Long, D. G. Schwartz, and S. Stoecklin. Multi-sensor network intrusion detection: a case-based approach. *WSEAS Transactions on Computers*, 4(12):1768–1776, 2005. 6

[85] Cyber panel grand challenge problem specification version 4.1. Technical report, June 2004. 6.3

[86] F. Bourgeois and J. Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, 14(12), 1971. A

BIOGRAPHICAL SKETCH

Jidong Long

Jidong Long was born on September 19th, 1976, in Tongnan, Chongqing, P.R. China. From August 1995 to April 2002, he completed both his Bachelors degree and his Masters degree in Computer Science at the University Of Electronic Science and Technology of China. He enrolled in the Ph.D program in Computer Science at Florida State University in the fall of 2002. Jidong's research interests include data mining, machine learning, and intrusion detection.