AN INTEGRATED PATTERN RECOGNITION APPROACH FOR ANOMALY
DETECTION IN A DISTRIBUTED SYSTEM

by

AMOD PANDIT

A THESIS

Presented to the faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirement for the Degree

MASTER OF SCIENCE IN COMPUTER ENGINEERING

2003

Approved by

Dr. R Joe Stanley, Advisor
Dr. Bruce McMillin

Dr. Ann Miller
ABSTRACT

Intrusion detection systems (IDS) attempt to address the vulnerability of computer-based systems for abuse by insiders and to penetration by outsiders. An IDS is often required to examine an enormous amount of data generated by computer networks to assist in the abuse detection process. It is therefore required to develop automated tools that address these requirements to assist system operators in the detection of violations of existing security policies. In this research, an automated IDS is proposed for insider threats in a distributed system. The proposed IDS functions as an anomaly detector for insider system operations based on the analysis of the system’s log files. The approach integrates dynamic programming and adaptive resonance theory (ART1) clustering. The integrated approach aligns sequences of log events with prototypical sequences of events for performing tasks and classifies the aligned sequences for anomaly detection and task discrimination. The system examined for this research is a Boots System for controlling the movement of boots from one place to another under specific security restrictions related to the boot orders. The integrated pattern recognition technique, experiments performed using the data from the Boots system and experimental results are presented and discussed.
ACKNOWLEDGEMENT

The supervisor of this work was Dr. Stanley; I am grateful for his invaluable support and academic guidance during my two years of research work. He has shown me extraordinary patience and understanding when times were rough. I also appreciate the help I received from the committee members Dr. Bruce McMillin and Dr. Ann Miller, the multi-level study of the Boots system would have been impossible to accomplish without their help.

The two years research work in the insider threat group provided an innovative and challenging atmosphere created by all the group members. Teamwork made the topic of this thesis both magnetic and stimulating. I want to thank all my co-workers in the Insider Threat Project: Krishnaprasad Balasubramanian, Ian Baird, Sule Simsek, Geoff Roth, and Benjamin Doering for their valuable feedback and their suggestions during several research meetings.

Finally, I am greatly indebted and grateful to my family and friends, for being so loving, supportive and patient during all my years in graduate school.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENT</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>LIST OF ILLUSTRATIONS</td>
<td>vii</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>1</td>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1. OVERVIEW OF THESIS</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2. LITERATURE REVIEW</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.1. NEURAL NETWORKS APPLIED IN INTRUSION DETECTION</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2.2. COMPUTER HOST BASED ANOMALY DETECTION SYSTEM USING SELF-ORGANIZING MAPS</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>3. OVERVIEW OF RESEARCH</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.1. OVERVIEW OF INVENTORY CONTROL SYSTEM</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>3.2. THE INVENTORY CONTROL SYSTEM</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3.3. THE BOOTS SYSTEM</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>3.4. SECURITY REQUIREMENTS</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.5. HISTORIES OF EVENTS</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>3.6. UNIVERSITY OF MISSOURI-ROLLA ONGOING RESEARCH FOR INSIDER THREAT DETECTION IN INVENTORY CONTROL SYSTEM</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>3.7. ONGOING RESEARCH IN PATTERN RECOGNITION</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>3.8. OVERVIEW OF INTEGRATED PATTERN RECOGNITION APPROACH FOR ANOMALY DETECTION AND TASK DISCRIMINATION</td>
<td>25</td>
</tr>
</tbody>
</table>
4. ALGORITHM USED FOR ANOMALY DETECTION AND TASK DISCRIMINATION

4.1. LIMITATIONS OF DATA AND DATA CONVERSION

4.2. FREQUENTLY USED TERMS

4.3. TWO-TIERED PATTERN RECOGNITION APPROACH

5. EXPERIMENTS AND RESULTS

5.1. TASK DISCRIMINATION EXPERIMENTS

5.2. TASK DISCRIMINATION RESULTS

5.3. EXPERIMENTS FOR ANOMALY DETECTION

5.4. EXPERIMENTAL RESULTS FOR ANOMALY DETECTION

6. CONCLUSIONS AND FUTURE WORK

REFERENCES

VITA
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1. Security system’s organization</td>
<td>10</td>
</tr>
<tr>
<td>3.1. The structure of the Boots system</td>
<td>16</td>
</tr>
<tr>
<td>4.1. Sequence and example of a vector time stamp or event</td>
<td>30</td>
</tr>
<tr>
<td>4.2. High-level order</td>
<td>30</td>
</tr>
<tr>
<td>4.3. Binary high-level order</td>
<td>31</td>
</tr>
<tr>
<td>4.4. Prototypic order</td>
<td>32</td>
</tr>
<tr>
<td>4.5. Equations for dynamic programming alignment algorithm</td>
<td>36</td>
</tr>
<tr>
<td>5.1. Denial of service attack</td>
<td>43</td>
</tr>
<tr>
<td>5.2. Denial of service attack vector time stamps</td>
<td>44</td>
</tr>
<tr>
<td>5.3. Boots consuming problem</td>
<td>45</td>
</tr>
<tr>
<td>5.4. Boots consuming problem vector time stamps</td>
<td>46</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1. Testing results over 15 iterations for recognizing orders from classes 1 and 2</td>
<td>41</td>
</tr>
<tr>
<td>5.2. Testing results over 15 iterations for recognizing intrusion and orders from classes 1 and 2</td>
<td>47</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

In a distributed computing system, security is an important concern due to the interaction of individual components as well as the easy accessibility of the applications to the outside environment through Internet. A secure system exhibits the characteristics of secrecy, integrity, and reliability. An intrusion detection technique attempts to detect intrusions into a computer or network by observation of actions, security logs, or audit data, or detection of break-ins or attempts either manually or via software expert systems that operate on logs or other information available on the network.

Intrusion detection is largely concerned with the detection of activities that may violate the system’s security. A distributed computing system is inherently vulnerable to security threats due to its need to interact through message passing over an open network. A security policy is a user requirement, much like functional correctness, that states the required protection [2]. The system is designed to enforce its security policy. Our approach integrates dynamic programming and adaptive resonance theory (ART1) clustering to detect unusual access patterns resulting from some abnormality in the system behavior or some insider intrusion. If a sequence of events occurs that violates the security policy, an alarm is raised. Unfortunately, the system may be subverted in ways that were not anticipated by the designers of the security policy.

This research focuses on anomaly detection caused by an insider threat, whereby improper changes are made to the internal system architecture resulting in an anomalous behavior. An example of an insider threat is a Trojan Horse installed by a programmer. A Trojan Horse is a program in which malicious or harmful code is contained inside
apparently harmless programming or data in such a way that it can get control and do its chosen form of damage, such as ruining the file allocation table on your hard disk.

The behavior of a Trojan Horse [2] is carefully constructed to be undetectable by a fixed, *a priori* security policy and may include additional message traffic or unusual patterns of message traffic that fall within the realm of the security policy, but eventually lead to some security problem. In a distributed computing system, the normal and abnormal interleaving of events in a message trace complicates anomaly and misuse detection.

The Boots system is a distributed system for transferring boots from one place to another under given security restrictions [5]. The Boots system is used as a model to generate access patterns to perform specific tasks. In this system events can be labeled for each task, providing the capability to treat tasks independently. In completing a specific task there may be concurrent events in the distributed system causing potential variations in the sequence of events to complete the task. For the distributed environment, acceptable interleaving of events must be separated from unacceptable interleaving profiles. With such profiles available, it is then possible to develop anomaly detection for unauthorized results that have not been previously encountered and misuse detection for known security threats. The two-tier pattern recognition techniques developed in this research detect unusual access patterns related to performing specific tasks using a combination of dynamic programming and Adaptive Resonance Theory clustering [22].

Ongoing research at the University of Missouri-Rolla is exploring insider threat detection techniques for a model multi-level security inventory control application with an associated formally specified security policy. This research will focus on recognizing
unusual access patterns from insider threat activity. Intrusions will be seeded into the test-bed through fault injection to verify how well unusual accesses patterns are detected by the \textit{a priori} security policy and the clustered assertions.

1.1. OVERVIEW OF THESIS

In section 3 an overview of the Boots system is presented. The Boots system controls the movement of footwear by processing orders to move boots from one place to another under given security restrictions. The chapter describes the functionality of the inventory control system, various components of the system and their behavior, security requirements, and creating and updating the histories of log due to communication between different participants of the system. Section 4 is devoted to the main algorithm. The pseudo code for this algorithm is discussed and the complete mathematical formulation of the same is presented in this section. Section 5 describes the experiments and results obtained, based on the Integrated Pattern Recognition algorithm. Results are presented from the Boots system’s history log data, available from the Computer Science Department of University of Missouri-Rolla. Limitations of the input data and the corrective actions taken are also discussed.
2. LITERATURE REVIEW

Insiders are those individuals who work for the target organization or have a relationship with the organization that grants the individual some level of access. A threat is defined as that which, if unchecked, will cause a loss to the organization. Therefore, an insider threat is defined as those who have authorized access to your organization that could possibly cause a loss to the organization if computer security goes unchecked.

The most serious security breach resulting in financial losses occurs through unauthorized access by insiders [8]. Insiders represent the greatest threat to computer security because they understand their organization's business and how the computer systems work. Therefore, an insider attack would be more successful at attacking the systems and extracting critical information. The insider also represents the greatest challenge in securing your network because they are authorized a level of access to your network and are granted a degree of trust.

To test the approach, fault injection attempts to uncover flaws by program state corruption. For example, we can corrupt program memory by using random number selection based on the expected program data type. Altering Boolean conditions in control flow constructs, if those conditions are known or surmised, can corrupt program control flow. Program states can be altered by buffer overrun or “stack smashing” techniques that introduce unknown code into the application.

Computer network security is based on realization of confidentiality, integrity, and availability in a computer network. Depending upon the type of detection, computer
intrusions can be divided into 2 classes, anomaly and misuse detection. Both techniques are discussed below with some models used for intrusion detection.

Anomaly Detection refers to the intrusions that can be detected based on anomalous behavior or use of computer resources. It is based on determining event traces that are known not to be normal. For anomaly detection, numerous approaches have been extended including: 1) threshold detection over a single metric such as signature differences, 2) profile-based detection over several metric [6], 3) statistical-based approaches comparing profiles [3,4], 4) rule-based detection [5], 5) neural network-based detection [6], and 6) others [7]. Predictive pattern generation, neural networks and pattern matching are three techniques that have been applied to anomalous behavior detection.

Predictive pattern generation [9] is a technique of anomaly detection that is based on the hypothesis that sequences of events are not random but follow a discernable pattern. These results in better intrusion detection because it takes into account the interrelationship and ordering among events. The approach of time-based inductive generalization described by Teng and Chen [10] uses time-based rules that characterize normal behavior patterns of users. The rules, generated inductively, are modified dynamically during the learning phase and only “good” rules, i.e., rules with high accuracy of prediction and a high level of confidence remain in the system. A rule has high accuracy of prediction if it is correct most of the time, and it has high level of confidence if it can be successfully applied many times in the observed data. An example of rule generated by Time based Inductive Machine (TIM) [10] may be E1\(\rightarrow\)E2\(\rightarrow\)E3\(\rightarrow\)(E4=95%, E5= 5%), where E1 to E5 are security events. The Time based Inductive Machine is a domain independent methodology for discovering potentially uncertain
temporal patterns from real time observations using the technique of inductive inference. The above lines mean that the probability of E1, E2, E3 followed by E4 is 95% and that of E5 is 5%. A set of rules generated inductively by observing user behavior comprises profiles of that user. A deviation is detected if the observed sequences of events match the left side of rule but deviate significantly from those predicted by the rules.

The time-based inductive generalization approach results in better handling of the user with wide variances of behavior but strong sequential patterns, ability to focus on relevant security events rather than entire login session that has been labeled suspicious, and better sensitivity to detection of violations. Because the meaning of rules [10] is easy to understand, the chance of catching “cheaters” who intend to falsify their normal profiles during the learning period is improved. The only drawback of this approach is that unrecognized patterns, which do not match the left side of sequence of events, may not be recognized as anomalous.

The Neural network approach is to train the neural net on a sequence of information units [9], each of which may be at a more abstract level than an audit record. The input to the neural network consists of the current commands and the past w commands; were w is size of the window of past commands used for predicting the next command. Once the neural network is trained on a set of representative command sequences of the user, the network constitutes the profile of the user and the fraction of incorrectly predicted next events then measures, in some sense, the variance of the user behavior from his profile.

Some advantages of this approach are that neural networks adjust well with noisy data, they automatically account for correlation between various measures that affect the
output, and success does not depend upon any statistical assumptions about the nature of underlying data. Some of the drawbacks are that the topology of the net and weight assigned to each element of the net is determined by trial and error method that may be time consuming; selection of w is another difficulty. If w is too small, then the network will do poorly; on the other hand, if w is too large, then the net will suffer from false positive intrusion detections.

Anomaly detection suffers from major drawback; you have no guarantee that a specific attack will even generate an alarm. If the intrusive activity is close to normal user activity, then the attack will go unnoticed. The next type of detection, i.e. misuse detection, which is discussed in the next section, is immune to such training; moreover, the technique is simpler than anomaly.

Misuse detection is the detection of intrusions by precisely defining them ahead of time. Misuse detection is predicated on identifying specific traces of events. There are also many approaches that have been extended to misuse detection, including: 1) static pattern recognition, 2) expert systems, 3) state transition analysis, and 4) others [3]. Misuse detection attempts to encode knowledge about attacks as well defined patterns and monitors occurrence of these patterns. Common approaches for performing misuse detection include expert systems, model-based reasoning systems, state transition analysis, and keystroke monitoring [11].

Expert system detectors code knowledge about attacks as if-then implications rules. Rules specify the condition requisite for an attack in their if part. When all the conditions on the left side of the rule are satisfied, actions on the right side of the rule are performed that may either trigger firing of more rules or conclude the occurrence of
intrusion. A Model-based reasoning system combines models of misuse with evidential reasoning to support conclusions about its occurrence. There is a database of attack scenarios where each scenario comprises a sequence of behaviors making up the attack. At any moment the system is considering a subset of these attack scenarios as likely ones being experienced by the system. The system verifies them by seeking information in the audit trial to substantiate or refute the attack scenarios. The anticipator generates the next behavior to be verified in the audit trial, based on current active models, and passes these sets to planner. The planner determines how the hypothesized behavior will show up in the audit data and translates it into a system dependent audit trial match. The evidential reasoning calculus built into the system permits one to update the likelihood of occurrence of the attack scenarios in the active model list. In state transition analysis, attacks are represented as state transitions of the monitored system. States in the attack pattern correspond to system states and have Boolean assertions associated with them that must be satisfied to transit to that state. Keystroke monitoring uses user’s keystrokes to determine the occurrence of attacks. The primary means is to pattern match for specific keystroke sequences indicative of an attack.

Pattern matching is based on the notion of events. Events are audible changes in the state of system or changes in the state of some part of system [11]. An event can represent a single action by a user or a system, or it can represent series of actions resulting in a single, observable record. The fundamental requirement of this approach is that matching is done with *follows* semantic rather than immediately follows semantic. For example, pattern AB specifies event A followed by occurrence of event B, and not A followed immediately by event B. Using the follows semantics makes the field of discrete
approximate pattern matching relevant to intrusion detection. A neural network using an unsupervised clustering algorithm can be designed which will detect the occurrence of the events in a particular sequence for follows as well as immediately follows semantics.

2.1. NEURAL NETWORKS APPLIED IN INTRUSION DETECTION

This section describes the model and a prototype of a neural network based security system [12]. The system uses neural networks to find out intruders to register and learn about their mode of operation, and then to provide elements that can help the system administrator to take actions against them. For this system, the agent is placed in a safe machine, one that is logically invisible to others and whose physical access is restricted. The agent passively monitors the network and captures the circulating packets through the use of the network interface in an indiscriminate mode.

The agent is organized in four layers that manage the packet flow in the network and provide a stimulus vector to the neural network. The lowest level only captures the flow of data and passes packets to the second layer. The second layer has 2 modules: Packets Preselection Module, which makes packet filtering that may represent interesting events, and Expert System Module, which analyzes filtered packets. These events are stored in rows and are vectors that include origin & destiny of connections, ports involved, security level and time stamp. The third module, i.e., the connection module, receives the packets and organizes it in a cause-effect relationship, identifying unidirectional data flow. Once identified and ordered, these packets are mapped into a flow vector whose pairs are represented by connection vectors. The semantic analyzer then acts on connection vectors to search for attack vectors. The fourth module, i.e., Post
Processing Module, receives the above information and forms a stimulus vector for the neural network.

The neural network analyses the stimulus vector and tries to attribute a suspicion degree, representing suspicious state of a particular connection. Before the neural network can identify potential attacks, it must be trained with meaningful and large enough amount of stimulus vectors representing behavior of suspicious connections and the legitimate ones. Once trained, the network must use its generalization ability to correctly identify the users who show characteristics similar to those included in the intruding actions used to train it. Figure 2.1 shows the organization of the security system.

Figure 2.1. Security system’s organization
2.2. COMPUTER HOST BASED ANOMALY DETECTION SYSTEM USING SELF-ORGANIZING MAPS

The objective with the self-organizing maps (SOM) test for anomaly is to test if the current behavior of an object is normal or anomalous [13]. The hypothesis tested to be is:

\( H_0 \) – The most recent observation is not anomalous.

\( H_1 \) - The most recent observation is anomalous.

The behavior of the object can be very consistent, which means it is concentrated to one or couple of regions in the feature space. It, on the other hand, can also be more scattered in the feature space, which would signify a more irregular behavior. The anomaly P-value is a measure of degree of anomaly for an observation. On the basis of this value \( H_0 \) is rejected or accepted. The anomaly P-value is calculated using the algorithm below.

The algorithm uses a set of features describing the object. The feature vector describing the object is denoted by \( f \). The normal behavior of object is observed, which means \( n \) measurements \((f_1, f_2,...,f_n)\) of feature vector is collected. A self-organizing map (SOM) with \( m \) neurons is trained, using \((f_1, f_2,...,f_n)\) as training data. Normally \( m < n \), for e.g. \( m = n/10 \); Neurons in SOM that are not the best mapping unit (BMU) for \((f_1, f_2,...,f_n)\) are omitted. The Best Mapping Unit distances for \((f_1, f_2,...,f_n)\) are calculated. These distances are denoted by \((D_1, D_2,...,D_n)\). The objective is to test the most recent observation \( f_{n+1} \) for anomaly. The BMU distance \( D_{n+1} \) for \( f_{n+1} \) is calculated.

Let \( B \) be the number of BMU distances \((D_1, D_2,...,D_n)\) that are bigger than \( D_{n+1} \). The anomaly P-value is then calculated as
If P-value is bigger than anomaly P-value threshold, then $H_0$ is accepted else $H_0$ is rejected. If the test indicates that behavior is anomalous, then $k$ most significantly deviating features can be determined. The $k$ features with the biggest absolute contribution to BMU distance are the $k$ most significantly deviating features. Equation (3) shows how most deviating feature is calculated.

$$F_{n+1, md} = \arg \max_j \{ \text{abs} (f_{n+1,j} - \text{BMU}_j) \} \quad \text{...............} \quad (3)$$

$j$ – Takes value from zero to no. of features.

If the anomaly P-value is smaller than the anomaly P-value threshold then $H_0$ is rejected and an alarm is triggered. For distributed environments, acceptable interleaving of events must be separated from unacceptable interleaving of events. With such profiles available, it is then possible to develop anomaly detection for unauthorized activity that have not been previously encountered and misuse detection for known security threats.
3. OVERVIEW OF RESEARCH

In this research an integrated dynamic programming and adaptive resonance theory clustering approach is investigated for identifying anomalous behavior in a Boots system [5]. Anomalous activity is defined as that which violates the a priori security policy and that which violates learned access patterns of subversive behavior.

3.1. OVERVIEW OF INVENTORY CONTROL SYSTEM

In order to study how executable assertions can be used and what the problems and consequences of such a usage are, in prior work a moderately complex model problem was studied and implemented in a real environment. The CCIS (Command and Control Information System) Boots system [5] is meant to control the movement of footwear by processing orders to move boots from one place to another under given security restrictions [6] regarding the orders. The system records in an event log all the security-relevant messages exchanged between participants. This implementation uses the distributed-processing approach, in which every participant in the system is designed and built as a separate process that is independent of all others and shares data only by exchanging messages with the rest of the processes. Thus, instead of a centrally maintained event log, the event log is diffused between the processes supporting the windows of the Boots application.

The Boots system is programmed in CCSP (C-programmed Communicating Sequential Processes) [1], an in-house developed programming tool that offers CSP-like syntax on top of a C environment. In CCSP concurrent processes can be created to handle
different tasks in the system. Communication mechanisms are available for processes to exchange messages and program states. Processes act as monitors to evaluate assertions embedded in the code and take action if the assertions fail. For temporal and trace-based assertions [7] each process maintains its own discrete clock and a vector clock, allowing security events occurring in different, independent processes to be partially ordered into a log using causality. Histories of events are maintained by clipping them according to the application requirements [3].

For run-time evaluation of the security policy, rather than modeling the actual security attack, any security violation is considered an error (with respect to the security specifications). Thus, the need for a failure model is avoided [4]. In [6] we took the formal security policy from [5], axiomatized it into executable assertions, and implemented the assertions and a distributed monitor forming a Run Time Security Evaluation system (RTSE). RTSE will be used as the base to evaluate the security policy for misuse detection.

3.2. THE INVENTORY CONTROL SYSTEM

A model problem implemented in a distributed environment was considered to study how executable assertions along with pattern recognition can be used to detect security violations and what the problems and consequences of such a usage are. The aim of designing this model problem was to check its behavior from the security point of view. The chosen model problem is based on the movement of footwear under security restrictions, introduced in [6], where the functionality, the participants, and the security requirements are described in conceptual level.
The prototype designed and implemented uses the distributed processing approach and was named The Boots System. The Boots System consisted of 15 participants representing 15 components in which every participant in the system is an independent process, which shares data only by exchanging messages with the other processes.

Elements necessary for RTSE were derived from the security restrictions requirement and were implemented for our prototype. CCSP Programming Language was used to implement the Boots System and the RTSE-related elements. The CCSP permits CSP constructs to be built into C code: for instance, an executable security assertion $SA_{x}$ can be written as `assert(SA_{x})` directly at the code level, and the CCSP support makes possible the evaluation of this assertion at run-time without any other action from the developer or the user.

3.3. THE BOOTS SYSTEM

The Boots System helps to move the boots from one place to another under specific security restrictions by means of orders and control the movement of the footwear by processing these orders. Different participants, each with given security attributes and obligations, perform different operations in the system, acting on behalf of users or group of user. The interaction between these participants during the functioning of the system and the security constraints placed on these interactions makes the Boots System an interesting model problem.

The participants in the system, depicted in Figure 3.1, are: HeadQuarter (HQ), Stock-cells (SH and SL), Transport, Movement, Warehouse (WH), Operator, Auditor and
Security_officer. The HQ issues orders to the Stock-cells to move a certain number of boots to a given destination for a given purpose. The orders are classified as low or high according to their sensitivity.

Figure 3.1. The structure of the Boots system
Of the two classified orders that are sent to stock cell, high order goes to high component of Stock-cell, while the low order goes to low component of Stock-cell. If an overqualified order has to be sent to low component of Stock-cell, then the HQ sends this order to Security_officer for downgrading. The Security_officer regrades it by changing the classification from high to low and sends it to the low component of Stock-cell. Stock-cells receive a purpose code from the HeadQuarter, which gives information about the type of boots, number of boots, source of the boots and the destination where the boots are to be moved. The purpose code is then sent to the Stock_record, which provides information about which of the three warehouses has the required types of boots. The Stock-cell then sends a message to Movement cell giving information about number, source and destination of boots, and also a message to the corresponding warehouse that must supply the required type of boots. Based on the number of boots required, the Movement cell calculates how many trucks are needed for transport of the boots, and sends this information along with the source and destination for boots to Transport. The Transport checks if the trucks are available and also check the truck records to decide which trucks are going to be used.

Whenever there is communication between any two participants in the Boots system, a copy of the message is sent to the Auditor_buffer. All the security relevant messages exchanged between the participants are recorded in the audit trial. The audit trial can be archived upon request using the Operator participant. The Security_officer can inspect the sender and receiver of messages in the audit trial without having access to the content of messages.
3.4. SECURITY REQUIREMENTS

In order to ensure security of the Boots system, some restrictions must be placed to control the movement of the footwear. In the Boots system, the security requirements are given as the restrictions on the possible observations of one or more participants and on each participant’s knowledge about other participant’s action. Five existing security requirements for the Boots system are presented below [6]:

1. Stock Low must not know a high order received from HeadQuarter at Stock_high.
2. The windows Movement-cell, Transport, Operator, and Warehouse must not know the purpose of an order from HQ.
3. The audit trial must not be corrupted.
4. The window Operator must not know the audit trial.
5. The security officer must not know the content of messages passed between the users.

These security requirements for the Boots system are formalized under the calculus for security and the resulting security requirements are labeled SR1 through SR5.

3.5. HISTORIES OF EVENTS

In the Boots System, the events are modeled as messages exchanged between participants. A message event contains a message type, a sending and a receiving process, and the message body. A timestamp given by the vector clock is also a part of the event, together with the message.

In general format for event histories: \( H(P_i) = (((e_{i1}^j, vc_{i1}^j), ... , m_i) \), where \( I \) indicates the process maintaining the history, and \( m_i \) is the current length of the history. An event \( e_{i1}^j \) in the Boots System is recorded as a tuple \( (msg_{type}, sender, receiver, \ldots) \).
\(msg\_body\) and \(vc_i^j\) is the value of the vector clock at the moment when the message was sent. The format and the content of the \(msg\_body\) depends on the type of message; for an order, for instance, the \(msg\_body\) contains the \((R, P, C)\) information, with \(R\) being the request for boots, in form of a \((number, destination)\) tuple, \(P\) is the purpose for the move, and \(C\) is the classification for the order.

The general mechanism for collection and maintenance of the histories is used without change for this format of histories, the only specific component being the consistency check which checks for the format of events.

To give an example of augmented communication in the Boots System, consider the HeadQuarter (HQ) process and process \(Stock\_low\) (stl) and \(Stock\_high\) (sth) with which it communicates. Consider an execution in which HQ first sends an \(order\) to \(Stock\_low\).

At the beginning of the execution all the histories are empty and local vector clocks have initial zero values:

- At HQ:

  \[VC_{HQ} = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\]

  Own H(HQ) = \(\langle\rangle\), Out H(HQ, sth) = Out H(HQ, stl) = \(\langle\rangle\)

- At Stock_low:

  \[VC_{stl} = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\]

  Own H(stl) = \(\langle\rangle\), Out H(stl, HQ) = \(\langle\rangle\)

- At Stock_high:

  \[VC_{sth} = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\]

  Own H(sth) = \(\langle\rangle\), Out H(sth, HQ) = \(\langle\rangle\)
HQ sends the low_order order 1 to Stock_low. For orders we use the (R, P, C) notation given earlier, where R is a request for boots, containing a quantity of boots and destination for boots, P is the purpose, and C is the classification, after the communication the histories and the clocks become:

- At HQ:

\[ VC_{HQ} = [1,0,0,1,0,0,0,0,0,0,0,0,0,0] \]

\[ \text{Own H(HQ)} = \langle ((\text{msg\_type} = \text{order}, \text{sender} = \text{HQ}, \text{receiver} = \text{stl}, \text{name} = \text{order1}, \text{R} = (400, \text{A}), \text{P} = 7, \text{C} = \text{low}), \text{time} = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]) \rangle \]

\[ \text{Out H(HQ, sth)} = \langle ((\text{msg\_type} = \text{order}, \text{sender} = \text{HQ}, \text{receiver} = \text{stl}, \text{name} = \text{order1}, \text{R} = (400, \text{A}), \text{P} = 7, \text{C} = \text{low}), \text{time} = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]) \rangle \]

\[ \text{Out H(HQ, sth)} = \langle \rangle \]

- At Stock_low:

\[ VC_{stl} = [1,0,0,1,0,0,0,0,0,0,0,0,0,0,0] \]

\[ \text{Own H(stl)} = \langle ((\text{msg\_type} = \text{order}, \text{sender} = \text{HQ}, \text{receiver} = \text{stl}, \text{name} = \text{order1}, \text{R} = (400, \text{A}), \text{P} = 7, \text{C} = \text{low}), \text{time} = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]) \rangle \]

\[ \text{Out H(stl, HQ)} = \langle \rangle \]

- At Stock_high:

\[ VC_{sth} = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] \]

\[ \text{Own H(sth)} = \langle \rangle, \text{Out H(sth, HQ)} = \langle \rangle \]
The own-histories for \(HQ\) and \(Stock\_low\) record the order1 message exchanged. \(HQ\) incorporated the same order event into its out-history for \(Stock\_high\). The vector clocks for \(HQ\) and the \(Stock\_low\) are synchronized, while at \(Stock\_low\) no event was observed and its vector clock is unchanged.

3.6. UNIVERSITY OF MISSOURI-ROLLA ONGOING RESEARCH FOR INSIDER THREAT DETECTION IN INVENTORY CONTROL SYSTEM

The research was divided into three tasks, with my task being designing a multi-facet pattern recognition model, which detects unusual access patterns resulting from abnormal behavior of the Inventory control system. The project ensures and controls the movement of the footwear by processing orders to move boots from one place to another under specific security restrictions regarding the orders. Any violation in security restrictions results in an alarm for the security officer. The three tasks are discussed below:

Runtime Systems and Simulation Task 1 includes modifying RTSE (Run Time Security Evaluation) to embed pattern recognition and cluster maintenance software to process the event log as well as design a mechanism for integration of identified clusters (from Task 3) as part of the security policy.

Testing and Intrusion Generation Task 2 involves fault injection, in which anomalies will be inserted within the system. Sample types of faults will include attempted program state corruption, program memory corruption, and program control flow corruption. The techniques for fault injection will utilize known intrusion and denial of service techniques, such as buffer overrun and Trojan Horse insertion. However, it is
expected that we will develop a test suite of more sophisticated attacks as our knowledge of potential insider event traces improves.

Clustering and Pattern Recognition Task 3 is to examine system executions and show error coverage for mapping these executions as erroneous, correct or unknown. The proposed approach integrates dynamic programming and unsupervised clustering. Issues to be developed beyond refining the basic dynamic programming approach include making use of known causality and concurrency of events, as currently the types of acceptable access patterns are known only in the context of the security policy and not in the context of system operation. A method of reliable cluster identification will be developed.

3.7. ONGOING RESEARCH IN PATTERN RECOGNITION

In the statistical classification approach [14], each pattern is represented in terms of $d$ features and measurements and is viewed as a point in a $d$ dimensional space. The goal is to choose those features that allow pattern vectors belonging to different categories to occupy compact and disjoint region in a dimentional feature space. The effectiveness of the representation space is determined by how well patterns from different classes can be seperated. Given a set of training pattern for each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to different classes. In the statistical decision aproach, the decision boundaries are determined by the probability distribution of the patterns belonging to each class, which must either be specified or learned [15,16].
One can also take a discriminant analysis-based approach to classification: first a parametric form of the decision boundary is specified; then the best decision boundary of the specified form is found based on the classification of the training patterns. Such boundaries can be constructed using, for example, mean squared error correction. The direct boundary construction approaches are supported by Vapnik’s philosophy [17]: “If you possess a restricted amount of information for solving some problem, try to solve the problem directly and never solve a more general problem as an intermediate step. It is possible that the available information is sufficient for direct solution but is insufficient for solving a more general intermediate problem.”

In Syntactic or Structural Classification [14] the key idea in structural and syntactic pattern recognition is the representation of patterns by means of symbolic data structures such as strings, trees, and graphs. In order to recognize an unknown pattern, its symbolic representation is compared with a number of prototypes stored in a database. Domain knowledge is required to guide the application of structural techniques for both feature extraction and classification. Consequently, structural pattern recognition has been primarily restricted to domains in which the set of shape-based features to extract is either well known (e.g., established in the literature) or obvious, and the syntactic grammars can be composed by hand.

Neural networks [18] take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach, i.e., the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known, the computer cannot solve the problem. That restricts the problem-solving capability of conventional computers to
problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully, otherwise useful time is wasted or even worse, the network might be functioning incorrectly. The problem is that there is no way of knowing if the system is faulty or not unless an error occurs.

The building block of a neural net is the neuron. An artificial neuron works much the same way the biological one does. It takes many inputs having different weightings and has one output, which depends on the inputs. A biological neuron can either 'fire' or not ‘fire’ (When a neuron fires, it outputs a pulse signal of a few hundred Hz). In an artificial neuron 'firing' is normally represented by a logical one and not 'firing' by a logical zero.

Neural nets are widely used in pattern recognition because of their ability to generalize and to respond to unexpected inputs/patterns. During training, neurons are taught to recognize various specific patterns and whether to fire or not when that pattern is received. If a pattern is received during the execution stage that is not associated with an output, the neuron selects the output that corresponds to the pattern from the set of patterns that it has been taught that is least different from the input. This is called generalization.
3.8. OVERVIEW OF INTEGRATED PATTERN RECOGNITION APPROACH FOR ANOMALY DETECTION AND TASK DISCRIMINATION

This research focuses on the discrimination of access patterns to perform specific tasks. Events will be labeled for each task, providing the capability to treat tasks independently. The technique for event pattern analysis is a hybrid approach based on a combination of dynamic programming and clustering. The scope of applying dynamic programming in this research is similar to the application of Markov chains in [19].

Dynamic programming is employed to align the actual sequence of events, represented as binary time stamp differences, in completing a specific task to a representative sequence of events for performing that specific task. The representative sequence of events is obtained as the most frequently occurring recorded sequence of events from system operation for the particular task. The total cost for aligning the actual sequence becomes a measure for differentiating normal access patterns from abnormal and unknown access patterns. For aligning sequences containing the same number of events, the cost computed reflects the likelihood that the given sequence will perform the specified task in the context of a normal access pattern. An unusual access pattern will generate a relatively high cost for alignment with the representative sequence based on known state transition frequency analysis.

The dynamic programming alignment provides a projection of the residual events to bridge the gap between representative and given sequences of different lengths. These event differences are isolated and examined for introducing unusual access patterns. The overall cost for the alignment provides one measure for evaluating the likelihood that the given sequence represents a normal access pattern. The added and deleted events provide another mechanism to evaluate the normalcy of the access pattern. Post-alignment
subsequence matching operations will be performed on added or deleted events to evaluate the potential presence of unusual access patterns. Due to situations like concurrent events, different sequences of events with different sequence lengths may occur to complete a given task.

In order to evaluate the sequences of events as well as the individual events, vector time stamp differences are taken between corresponding processes constituting consecutive events. The resulting time stamp differences are binary vectors used to characterize the individual events for clustering analysis and for alignment to a reference sequence. The individual events may adhere to the security policy, but the integration of these events into the sequence for completing the specified task may introduce a security violation. Upon detection, these sequences can be integrated into the security policy for future misuse detection. Following the post-processing operations, the aligned sequence of events is inspected on an event-by-event basis relative to the process or task using a clustering-based approach.

Adaptive resonance theory (ART1) [20,21] provides an unsupervised learning approach for the controlled discovery of clusters relating to acceptable and unusual access patterns. ART1 offers two important capabilities. First, new clusters can be determined without affecting the recall of clusters already created. Second, there is online adaptability whereby total retraining is not required every time a new pattern is found. ART1 provides the ability to partition the training data in such a way that acceptable access patterns can be isolated as specific clusters; regions associated with unusual access patterns can be isolated as separate clusters. Unusual access patterns may result from a single ordered set of log events or from a sequence of ordered log events. Dynamic
programming alignment with the prototypical sequence of events to accomplish a task provides the basis for assessing the latter. Cluster analysis using the aligned sequence of events from ART yields the basis for examining the former.
4. ALGORITHM USED FOR ANOMALY DETECTION AND TASK DISCRIMINATION

4.1. LIMITATIONS OF DATA AND DATA CONVERSION

To correctly classify the data obtained from the log files created due to interaction between different departments in the Boots system using a neural network, it was required that the data should be presented in a specified format understandable to the Integrated Pattern Recognition model. Some preprocessing was required to convert the data into the required format while taking care that the no useful information is lost while processing the data. The following section describes the problems encountered with the data and the remedial action taken to convert the data into the required format.

Whenever there is transaction between any two processes in the Boots System, the corresponding local vector stamps of the two processes were incremented by a constant value. This results in an order with events containing decimal numbers that represent the sequence in which the transactions took place to complete a particular task. These orders were further classified and clustered using Adaptive Resonance Theory (ART1), an unsupervised clustering approach to detect any anomalous behavior or intrusions in the system. Since ART1 requires input patterns to be only in binary, some means were required to convert the decimal numbers into binary values. In order to evaluate the sequences of events as well as the individual events, vector time stamp differences were taken between corresponding processes between consecutive events. The resulting time vector stamp differences are binary vectors used to characterize the individual events for dynamic programming matching, alignment and then clustering.
Another problem faced was, a certain task can be performed in number of different ways depending upon the path that is followed for completing a task. This gives orders having a different number of events and hence different lengths. To cluster the orders using ART1, it is required that the orders that are input patterns to the neural network, should have equal dimensions. Dynamic programming is employed to align an actual sequence of events, represented as vector time stamps, to a representative sequence of events for performing a specific task. Any order with length greater or less than the representative sequence is normalized to length equal to that of the representative sequence without losing much of the information stored in the pattern.

4.2. FREQUENTLY USED TERMS

This section defines the most commonly used technical terms in the thesis. Some of the most commonly used terms were event, order, prototype, cost, comparator, intrusion etc. An event is a vector of 15 elements, where each element represents a time stamp for 15 different processes. The participants in the Boots system communicate by exchanging messages that cause an event. Thus an event can be modeled as a message between two processes. An example of an event is shown in Figure 4.1 below. The labels show the order in which the vector time stamps are displayed. For e.g. the position of two ones in the vector stamp shows that the corresponding local vector time stamps of HeadQuarter and Stock_low are incremented by one, thus indicating communication between HQ and the SL.
Different groups of events represent a particular task, referred to as an order. Performing a specific task might require exchanging many messages between the participants of the system. These messages or events when grouped together form an order. Thus, by definition, an order is a group of events that represents a particular task being performed. An example of a high level order is shown in Figure 4.2.

| HQ, Sth, Stl, SO, WR-0, WR-1, WR-2, MV, TR, TR_R, ST_R, AUD, AUD_B, OP, AR |
|---------------------------------|------------------|
| 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0|

**Figure 4.1. Sequence and example of a vector time stamp or event**

| 13 0 1 1 0 0 0 0 0 0 4 0 0 10 |
| 16 1 1 1 0 0 0 0 0 0 4 0 0 10 |
| 13 4 1 1 0 0 0 0 0 0 4 0 0 10 |
| 13 4 7 1 0 0 0 0 0 0 10 4 0 0 10 |
| 13 10 7 1 0 0 0 0 0 0 10 4 0 0 10 |
| 13 13 7 1 0 1 0 0 0 0 10 4 0 0 10 |
| 16 16 7 1 0 1 0 0 0 0 10 16 0 0 28 |
| 16 19 7 1 0 1 0 1 0 0 10 16 0 0 28 |
| 16 16 7 1 0 1 0 7 1 0 10 16 0 0 28 |

**Figure 4.2. High-level order**
Figure 4.3 shows the binary order obtained after taking vector time stamp differences between corresponding processes between consecutive events. This was done to achieve a binary order that has same length as the actual order, after taking the differences between corresponding vector time stamps.

```
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0 0 1 0 0 1
0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

**Figure 4.3. Binary high-level order**

In the context of system operation, there are many types of orders performed, called classes of orders. The classification was based upon the level of the order, whether it is a *high_level* order or a *low_level* order. The sequence of events tells the path followed by the messages while performing a task. The most frequently occurring sequence in a given class of order is called a prototype. Thus a prototype is an actual sequence of events, represented as binary time stamp differences previously described, in completing a specific task. An example of prototype order is shown in Figure 4.4.
A comparator is a sequence of events required to complete a current task. Dynamic programming is used to compare the prototype and comparator sequences to determine the degree of match between the sequences and to align the comparator to have the same number of events, i.e. length, as the prototype. The dynamic programming match cost was assigned such that higher the cost the worse the match between a test (comparator) pattern and a prototype sequence. The total cost for aligning the actual sequence becomes a measure for differentiating normal access patterns from abnormal and unknown access patterns. An intrusion is an attempt made for unauthorized access to a system with the purpose of simply to test the security of the network, use the facility as a launching pad for further attacks on other systems, to modify information or to steal information, etc. An IDS gathers and analyzes information from various areas within a
computer or a network to identify possible security breaches. Such a mechanism or IDS that detects intrusions in a system is discussed in the following section.

4.3. TWO-TIERED PATTERN RECOGNITION APPROACH

The Intrusion detection systems (IDS) functions as an anomaly detector for insider system operations based on the analysis of the system’s log files. The approach integrates dynamic programming and adaptive resonance theory (ART1) clustering to detect unusual access patterns resulting from some abnormality in the system behavior or some insider intrusion [22]. From the 25 fields in the log file, the model used the last 15 elements, the sender and receiver’s process id, and the order id for classification. The discrimination of orders is done on the basis of the sender and the receiver’s process_id, while the order_id indicates different events to be grouped to form an order. Training data over normal system operation is used to develop the dynamic programming and ART1 network components for the pattern recognition model. A most frequently occurring group of events to perform a particular task is chosen as a representative, or prototypical, sequence. The integrated approach aligns sequences of log events with prototypical sequences of events for performing tasks and labels the aligned sequences for anomaly detection and task discrimination. Dynamic programming is employed to align an actual sequence of events, represented as vector time stamps, to a representative sequence of events for performing a specific task. Unusual access patterns may result from a single ordered set of log events or from a sequence of ordered log events. Dynamic programming alignment with the prototypical sequence of events to accomplish
a task provides the basis for assessing the former, while cluster analysis using the aligned sequence of events from ART1 yields the basis for examining the latter.

For the Boots System, an event is a vector of 15 elements, each representing a time stamp for 15 different processes. In order to evaluate the sequences of events as well as the individual events, vector time stamp differences are taken between corresponding processes between consecutive events. The resulting time vector stamp differences are binary vectors used to characterize the individual events for dynamic programming matching and alignment. Let \( Z = (z_{11}, z_{12}, \ldots, z_{1p}; z_{21}, z_{22}, \ldots, z_{2p}; \ldots; z_{r1}, z_{r2}, \ldots, z_{rp}) \) represent a sequence of the binary vector time stamps for \( r \) events with \( p \) processes. Then, each vector \( z_{i1}, z_{i2}, \ldots, z_{ip} \) for \( 1 \leq i \leq r \) represents an event \( E_i \), where each unique event in \( Z \) is denoted \( U_i \). The frequency counts of all unique events \( U_i \) for all sequences in the training data are determined. The frequency count for event \( E_i \) is referenced as \( F_{E_i} \), and the total number of events occurring in the training data is denoted as \( T \).

There are many different groups of events that represent a particular task, referred to as an order. In the context of system operation, there are many types of orders performed, called classes of orders. From the training data for each class of orders, the representative sequence of events is obtained as the most frequently occurring recorded sequence of events from system operation for performing the particular task. Let

\[
P = (y_{11}, y_{12}, \ldots, y_{1p}; y_{21}, y_{22}, \ldots, y_{2p}; \ldots; y_{m1}, y_{m2}, \ldots, y_{mp})
\]

denote the sequence of binary vector time stamps.

\[
Y = (E_{y1}, E_{y2}, \ldots, E_{ym})
\]

denote the corresponding sequence of events for the representative sequence containing \( m \) events.
\[ Q = (x_{11}, x_{12}, \ldots, x_{1p}; x_{21}, x_{22}, \ldots, x_{2p}; \ldots; x_{n1}, x_{n2}, \ldots, x_{np}) \]
denote the number of events recorded in the log.

\[ X = (E_{x1}, E_{x2}, \ldots, E_{xn}) \]
refer to the sequence of events for completing the current task containing \( n \) events.

If \( n \geq m \), then \( Y \) is used as the reference sequence and \( X \) is utilized as the comparator sequence for dynamic programming matching. Otherwise, \( X \) is utilized as the reference sequence, and \( Y \) is used as the comparator sequence. For the dynamic programming model implemented, every element of the reference sequence is used in the alignment and comparison process with the comparator sequence. Costs are assigned to insertions, substitutions, and matches on each of the events in the order. The computation and math involved can be viewed as filling a cost matrix such that a trace from upper left entry to the lower right entry defines an optimal editing of the order. Thus, the output of the dynamic programming is the comparator sequence aligned to have the same dimension as that of the reference or the prototype sequence. The dynamic programming algorithm for aligning and comparing the reference and comparator sequences is given below. It is assumed that \( Y \) is the reference sequence and \( X \) is the comparator sequence. Let \( e_j^* \) denotes the current comparator element. Figure 4.5 shows the equations developed for dynamic programming algorithm.
For $j = 1$:

For $i = j \ldots j+n-m$:

\[ \text{Compute } e_i^* = \sum_{c=1}^{i} \left( \frac{T}{F_{E_{xc}}} \right), \quad f_j = \frac{T}{F_{E_{yj}}}, \quad s_{t,i} = \left| e_i^* - f_j \right| \]

For $j = 2 \ldots m-1$:

For $i = j \ldots j+n-m$:

For $k = j \ldots i$:

\[ \text{Compute } e_i^* = \sum_{c=1}^{i} \left( \frac{T}{F_{E_{xc}}} \right), \quad f_j = \frac{T}{F_{E_{yj}}}, \quad s_{j,i} = \min_k (s_{j-1,k-1} + \left| e_i^* - f_j \right|) \]

For $j = m$:

\[ i = n: \]

For $k = m \ldots n$:

\[ \text{Compute } e_i^* = \sum_{c=1}^{i} \left( \frac{T}{F_{E_{xc}}} \right), \quad f_j = \frac{T}{F_{E_{yj}}}, \quad s_{m,n} = \min_k (s_{m-1,k-1} + \left| e_i^* - f_j \right|) \]

**Figure 4.5. Equations for dynamic programming alignment algorithm**

Here, $s_{m,n}$ is the final cost for matching $X$ to $Y$. For a given event, the algorithm scans through the library of unique events to find its frequency count. If no match is found, it means that the event has never been seen before. This event therefore, can be considered as an abnormal event and the cost of matching this event with the representative sequence should be reasonably high. A frequency count value of 0.2 is assumed for this event, which results into a cost 5 times higher than the normal cost of matching. For the dynamic programming algorithm, a direction matrix is maintained in parallel with the cost matrix to provide the event mappings between the reference and comparator sequences. The direction matrix keeps track of the combination of the elements that represents the cost on the cost matrix for all the multiple cost combinations.
Once the optimum solution of matching the two orders is obtained, the direction matrix has all the information necessary to know which of the elements from the comparator were recombined to get the optimum cost or the best match. Knowing this combination to obtain the best matching sequence is known as back solving. The direction matrix is therefore required for back solving the order alignment problem since the best optimal cost of matching the entire order using dynamic programming is obtained at the very end. For this implementation, when more than one event is combined for mapping with a reference event, the reference event is inserted in place of the combined events of the comparator.

The aligned sequence vector from the dynamic programming algorithm is input to an ART1 network. The ART1 network has been trained using the same binary vector time sequences that were used for generating the frequency counts for the dynamic programming model as well as the same representative sequence. For the training process, the remaining sequences are input to the dynamic programming model for alignment with the representative sequence. The cost from dynamic programming alignment and the aligned sequences are used for unusual access pattern assessment. The aligned sequences are input to the ART1 for clustering. The vigilance parameter ($\rho$) related to relative closeness of the clusters is determined experimentally.

In the testing mode for the integrated pattern recognition approach, the current task is aligned to the representative sequence for the selected order. The dynamic programming cost is evaluated as a possible unusual access pattern. The aligned current task is input to the ART1 network for cluster mapping for the selected order. Because normal or acceptable access patterns for performing specific tasks are used to train the
two-tier pattern recognition technique, tasks are labeled with respect to normal behavior. If the aligned task maps into any cluster, the task is considered normal. A task is considered abnormal if the dynamic programming match cost is greater than or equal to a specified threshold ($T_H$), and the task does not map into any cluster. For this research, $T_H$ is defined as the maximum dynamic programming score from the training data divided by a constant $K$. 
5. EXPERIMENTS AND RESULTS

5.1. TASK DISCRIMINATION EXPERIMENTS

The data from the log files for the Boots System is classified into two main classes, depending on the sender and receiver’s process ids. Class 1 contains all orders sent from *HeadQuarter* to the high component of the Stock-cell, while Class 2 contains those sent from *HeadQuarter* to the low component of the Stock-cell. From collected data, class 1 has 63 different orders, and class 2 has 89 orders. The training data for each class is obtained by randomly picking 80% of the total orders in a class; the remaining 20% constitutes testing data. Training and testing sets are chosen over 15 iterations to check the consistency of the model. For the training data for each class, the following is performed.

First, all the unique vector time stamps from training data of each class and their corresponding frequency counts is found. The frequency count of each vector stamp is used in calculating the probabilistic distribution, which is required for finding the cost of matching in the dynamic programming approach.

Second, the most frequently occurring order in a class is chosen as a prototype for that class. All orders are dynamic programming matched to the prototype for the specified class to yield the match cost and the aligned sequence of events. For the orders from the specified class, the maximum cost is determined and divided by a constant $K$. The value of $K = 1.3$ was chosen empirically from the training data to obtain the best discrimination of normal and abnormal access patterns.
Third, the aligned sequences from dynamic programming matching are input to the ART1 network for clustering. The ART1 network is trained with the aligned sequences obtained from the dynamic program algorithm. The value of the surveillance parameter $\rho$ was selected by trial and error basis to get optimum results. The value found out was 0.9 (90% matching). Once the neural network is trained, it is ready to classify the test data into the clusters.

Fourth, the test data of each class is tested through training data of both the classes (class1 and class2). A test sequence of events for class 1 that either has a dynamic programming match score below K or maps into one of the clusters for class 1 or both is scored as normal. Any deviation from these criteria results in an abnormal access pattern for class 1. Thus the intent is to label all the class 2 tasks abnormal relative to class 1. The same rules apply for class 2.

5.2. TASK DISCRIMINATION RESULTS

Table 5.1 shows the average testing results over 15 iterations obtained when the testing data for each class was passed through the training data of both the classes. From the experiments the value of $K = 1.3$ was used for all 15 training/testing set of iterations. For the experimental procedure, 80% of the data is used for training and rest 20% for testing. Based on the training approach for Dynamic Programming model and ART1 network for each class, 100% of the training is correctly classified. This is true because every order from the training data is contained in an ART1 cluster. For classes 1 and 2 there are 14 and 17 orders, respectively, in the testing set. Column 1 contains the testing set class. Column 2 shows the training data class. Column 3 contains the percentage of
correctly classified test sequence. Column 4 shows the average threshold $T_H$ over the 15 iterations for rejecting aligned sequences that are not contained in any cluster for the specified order class.

Table 5.1. Testing results over 15 iterations for recognizing orders from classes 1 and 2

<table>
<thead>
<tr>
<th>Test Class</th>
<th>Train Class</th>
<th>No Of Orders</th>
<th>Average %Correct Discrimination</th>
<th>Average Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>14</td>
<td>99.5%</td>
<td>850</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>14</td>
<td>91.0%</td>
<td>7850</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>17</td>
<td>98.6%</td>
<td>850</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>17</td>
<td>91.8%</td>
<td>7850</td>
</tr>
</tbody>
</table>

The average of 15 iterations indicates that 99.5% orders of class 1 were correctly called class 1, 98.6% orders from class 2 were correctly classified as belonging to class 2, 91% of orders of class 1 were correctly not labeled as class 2, and 91.8% orders of class 2 were correctly classified as not belonging to class 1. These experimental results show that the integrated approach can be used for differentiating the order types. In discriminating the orders of one class from the other, since the path followed by the messages after stock cells is same for both the classes, some of the orders in one class could contain sequence of events that when passed through training data of other, gives a low cost of matching. This results in incorrect classification of the order. The results below indicate that the approach was successful in discriminating more than 90 % of the orders belonging to different class. The results were quite consistent for different data, and since the two-tier
pattern recognition model was successful in discriminating all the orders in one class that fairly differed from the orders in other, the approach can be said to work as an anomaly detector.

5.3. EXPERIMENTS FOR ANOMALY DETECTION

In the previous section, the two-tier pattern recognition model was tested for tasks belonging to different classes. Experiments were performed under the assumption that orders belonging to class 1 were considered abnormal for those belonging to class 2 and vice versa. Results showed that the model was quite successful in discriminating orders of different classes. In real time, orders belonging to different classes are processed separately and there is not any relation between the two types of orders. Some means were necessary to check the integrity of the model when working in a real time application; for this reason several kinds of anomalies, discussed below, were introduced in the system and were passed through training data of both the classes to check if the model was able to detect those anomalies.

The first anomaly introduced works as follows: The HeadQuarters sends an order to high component of stock cell to transport a given number and type of boots to a particular location. While the message is on its way, Stock_low intercepts the message from Stock_high. It then modifies the message and sends it on to Warehouse-1. The message from Stock_low is received before the message from Stock_high. Because of this, Warehouse-1 discards the original message as a duplicate and only executes the intrusive order. Figure 5.1 shows the graphical representation of the intrusion.
Figure 5.1. Denial of service attack

The first four lines in Figure 5.2 shows that there is proper communication between the HeadQuarter and Stock_high components of the Boots system, the fifth line with S_pid=2 and R_pid=4 indicates that instead of Stock_high sending the message to Warehouse-1 the message is sent by the Stock_low thus resulting into anomalous behavior.
S_pid  R_pid  Vector Stamps
0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  14  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
1  10  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1
10  1  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
2  4  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0
2  7  0  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0
1  14  0  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0
7  8  0  0  0  0  0  0  1  0  0  0  0  0  1
7  14  0  0  0  0  0  0  1  1  0  0  0  0  0  0

**Figure 5.2. Denial of service attack vector time stamps**

The second anomaly works by *Stock_high* sending multiple messages to *Warehouse-0* asking for boots. The *Warehouse-0* runs out of boots due to multiple orders from the high component of stock cells. When *Stock_low* sends an order to *Warehouse-0* asking for boots it does not get any response from *Warehouse-0* as it has run out of boots. Thus a covert channel is created between the high and low component of the stock cells, thus violating the rules of the system. The sequence of events below shows the anomaly, where S_pid and R_pid is sender and receiver’s process ids respectively indicating communication between respective processes. Figure 5.3 shows the diagrammatic representation of the anomaly.
In Figure 5.4 the first line is *HeadQuarters* giving order to *Stock_high* and then in second line sends a copy of message to *Auditor_buffer*, which updates the history records. Third and fourth line is the communication between *Stock_high* and *Stock_records* to obtain the purpose code. The lines with S_pid=1 and R_pid=4 are the multiple messages sent by the *Stock_high* to the *Warehouse-0* indicating anomalous behavior of the system. In this anomaly, even though *Stock_low* interaction is from
different order, the fact that Stock\textsubscript{high} has to send multiple messages to Warehouse-0 to create covert channel between Stock\textsubscript{low} and Stock\textsubscript{high} indicates abnormal behavior.

\begin{verbatim}
S_pid  R_pid  Vector Stamps
0      1      1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0      14     1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1      10     0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
10     1      0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
1      4      0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1      4      0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1      4      0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1      7      0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1      14     0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
7      8      0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
7      14     0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
\end{verbatim}

Figure 5.4. Boots consuming problem vector time stamps

5.4. EXPERIMENTAL RESULTS FOR ANOMALY DETECTION

Table 5.2 shows the average testing results over 15 iterations obtained when the testing data for each class was passed through the training data of both the classes. Similar experiments, as explained in the previous section were performed. The average of 15 iterations indicate, 95% orders of class 1 were correctly called class 1, 96% orders from class 2 were correctly classified as belonging to class 2, 100% of orders of class 1
were correctly not labeled as class 2, and 100% orders of class 2 were correctly classified as not belonging to class 1. Also, the two intrusions, indicated by last two rows of the table, were tested for both the classes. The experimental results show that both the intrusive orders when passed through the training data of class 1 and 2 were detected as abnormal. Thus, the designed model was successful in discriminating different kinds of orders and detecting the intrusive orders.

Table 5.2. Testing results over 15 iterations for recognizing intrusion and orders from classes 1 and 2

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Test Class</th>
<th>Train Class</th>
<th>No of Orders</th>
<th>Average %Correct Discrimination Results</th>
<th>Average Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
<td>1</td>
<td>1</td>
<td>14</td>
<td>95%</td>
<td>854</td>
</tr>
<tr>
<td>Test</td>
<td>1</td>
<td>2</td>
<td>14</td>
<td>100%</td>
<td>2167</td>
</tr>
<tr>
<td>Test</td>
<td>2</td>
<td>2</td>
<td>17</td>
<td>96%</td>
<td>854</td>
</tr>
<tr>
<td>Test</td>
<td>2</td>
<td>1</td>
<td>17</td>
<td>100%</td>
<td>2167</td>
</tr>
<tr>
<td>Intrusion</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>100%</td>
<td>1854</td>
</tr>
<tr>
<td>Intrusion</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>2163</td>
</tr>
</tbody>
</table>
6. CONCLUSIONS AND FUTURE WORK

The need for evaluation of security comes from the likelihood of occurrence of abnormal conditions that may appear in the underlying system. An integrated pattern recognition approach for intrusion detection was introduced to detect unusual access patterns. Unusual access patterns may result from a single ordered set of log events or from a sequence of ordered log events. The integrated approach utilizes dynamic programming alignment with the prototypical sequence of events to accomplish a task for examining and classifying sequences of log events. The aligned sequences of events are examined using cluster analysis from ART1 for identifying unusual access patterns from the single ordered set of log events.

In this research, sequences of log events in the form of vector time stamps for differentiating two types for a Boots Inventory System were examined. The model was tested for two types of order classes under the assumption that orders belonging to class 1 were considered abnormal to those belonging to class 2. Experimental results show that the model was successful in differentiating 91% orders of class 1 and 91.8% orders of class 2, when passed through the other class. The model was further tested for its integrity by introducing two intrusions in the test data of one of the classes. The experimental results show that the model was able differentiate an intrusive order or orders from those considered normal, with 100% accuracy for both the classes. Thus, the architecture for this system is integrated into an overall intrusion detection system and is proved experimentally to be successful in task discrimination and anomaly detection.
Results reported in this research provide a preliminary evaluation of the pattern recognition component for the intrusion detection system.

For differentiating abnormal access patterns from the normal ones, out of the 25 fields in the log file, the two-tier pattern recognition approach uses the last 15 fields, the transaction id, the sender’s process id, and the receiver’s process id. The approach hence suffers from some limitations. All anomalous activities or intrusions that are not reflected in the incorporated fields could not be detected. The future work will consist of extension of the two-tier pattern recognition approach that will involve incorporating other attributes, i.e. the remaining 7 fields from the system log files, in making unusual access pattern assessments.
REFERENCES


[9] Classification and detection of Computer Intrusions by Sandeep Kumar, Purdue University, August 1995. 
http://citeseer.nj.nec.com/kumar95classification.html


http://citeseer.nj.nec.com/kumar94pattern.html


VITA

Amod Pandit was born in Pune, India, on May 1, 1976. In June 1997, he received his B.S. with Honors in Electrical, Electronics and Power from Government Engineering College Aurangabad, India. He did his college project on Programmable Logic Controller (PLC) in Bajaj Auto Ltd during 1996-97. After completing his B.S. he joined Bajaj Auto Ltd as a Section Manager and worked for an additional two and a half years. He was promoted from level O1 to level O2 in March 2000. In December 2002, he received his M.S. Degree in Computer Engineering from the University of Missouri – Rolla, Rolla, Missouri, USA.

He has published a conference paper, which is listed with the references of his research. During his graduate studies he performed teaching activities for a number of (practical) classes. This thesis marks the end of his long trek through graduate school. His current research interests include developing a two-tier pattern recognition model that is able to detect insider threats in a computer network.