2004

A hybrid intrusion detection system

Yanxin Wang
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation
https://lib.dr.iastate.edu/rtd/1129

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A hybrid intrusion detection system

by

Yanxin Wang

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Johnny S. Wong, Major Professor
Andrew S. Miner
Doug W. Jacobson
James A. Davis
Pete J. Boysen

Iowa State University
Ames, Iowa
2004

Copyright © Yanxin Wang, 2004. All rights reserved.
Graduate College
Iowa State University

This is to certify that the doctoral dissertation of
Yanxin Wang
has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor
Signature was redacted for privacy.

For the Major Program
DEDICATION

I would like to dedicate this dissertation to the memory of my father Xingjiu Wang.
ACKNOWLEDGEMENT

I would like to thank my advisor, Dr. Johnny Wong, for his constant guidance and encouragement. During my four years' study at Iowa State University, I get tremendous advice and support from him. Without his support, I would have never finished this research. I would also like to thank Dr. Andrew Miner for his extensive help and comments on my research and papers.

I also owe a great magnitude of thanks to Dr. Doug Jacobson for many useful suggestions on my academic growth, Dr. Pete Boysen for helping me overcome many difficulties and Dr. James Davis for serving on both my MS and PhD committee and offered valuable suggestions.

Finally, I would like to thank my family - my parents and my brother - for their love and constant support, and all the friends who helped me during my four years' study in Iowa State University.
# TABLE OF CONTENTS

ACKNOWLEDGEMENT ............................................................... iv

LIST OF TABLES ................................................................... x

LIST OF FIGURES ................................................................... xii

ABSTRACT ........................................................................... xiii

CHAPTER 1. Introduction .......................................................... 1
  1.1 Intrusion and intrusion detection ........................................... 3
  1.2 Thesis statement ............................................................... 5
  1.3 Contributions ................................................................ 6
  1.4 Roadmap of this dissertation .............................................. 7

CHAPTER 2. Related work .......................................................... 9
  2.1 Misuse intrusion detection system ........................................ 9
    2.1.1 Rule based expert system .......................................... 9
    2.1.2 High level representation of intrusion detection .......... 11
  2.2 Anomaly intrusion detection system .................................... 13
    2.2.1 Statistical model .................................................... 15
    2.2.2 Computer immunology based anomaly detection ......... 20
    2.2.3 Machine learning and data mining methods for anomaly intrusion detection ............................................................... 22
  2.3 Specification based intrusion detection system .................... 29
    2.3.1 FSM based intrusion detection ................................ 30
    2.3.2 Uppuluri's specification based intrusion detection/response system ........ 31
    2.3.3 Ko's specification based approach ............................ 32
4.2.1 STIDE anomaly detector ........................................ 55
4.2.2 STIDE kernel .................................................... 56
4.2.3 Efficient computation of STIDE kernel ..................... 57
4.3 Design and computation of Markov Chain kernel .......... 58
  4.3.1 Markov Chain anomaly detector .......................... 58
  4.3.2 Markov Chain kernel ......................................... 59
  4.3.3 Efficient computation of Markov Chain kernel .......... 60
4.4 Experiments on STIDE and Markov Chain kernel based Support Vector Machine (SVM) anomaly detectors .................. 61
  4.4.1 Experiments on the UNM data ............................. 61
  4.4.2 Experiments on the DARPA Lincoln Lab data .......... 63
4.5 Impact of window size on the coverage of system call sequence based intrusion detection ................................................. 64
  4.5.1 Coverage of STIDE anomaly detector .................. 65
  4.5.2 Coverage of Markov Chain anomaly detector .......... 65
  4.5.3 Impact of window size on SVM anomaly detector ...... 69
4.6 Summary .......................................................... 71

CHAPTER 5. One class SVM for anomaly detection ................. 72
  5.1 Introduction .................................................... 72
    5.1.1 Clustering ................................................ 73
    5.1.2 One class SVM ............................................ 74
  5.2 Experiments on one class SVM based anomaly detectors .... 74
    5.2.1 Experiments on the UNM data .......................... 74
    5.2.2 Experiments on the DARPA Lincoln Lab data ...... 76
  5.3 Discussion of the experiment results ...................... 76
  5.4 Summary ........................................................ 77

CHAPTER 6. Specification assisted anomaly detection ............. 78
  6.1 Introduction .................................................... 78
APPENDIX B. Examples of SFT modeling intrusions and CPN modeling

intrusion detection design template ........................................... 112

B.1 FTP bounce attack ............................................................. 112

B.2 Trinoo Distributed Denial of Service attack (DDoS) ............... 115

BIBLIOGRAPHY ................................................................. 119
LIST OF TABLES

4.1 Comparison of STIDE, Markov Chain, STIDE kernel based, Markov Chain kernel based anomaly detectors on the UNM data ............ 62
4.2 Comparison of polynomial kernel, RBF kernel, STIDE kernel and Markov Chain kernel based anomaly detectors on the UNM data ............ 63
4.3 Comparison of STIDE, Markov Chain, STIDE kernel based and Markov Chain kernel based two class SVM anomaly detectors on the FTP processes in the DARPA Lincoln Lab data .................. 64
4.4 Comparison of STIDE, Markov Chain, STIDE kernel based and Markov Chain kernel based two class SVM anomaly detector on multiple processes in the DARPA Lincoln Lab data .................. 64
4.5 Window size impact on the STIDE kernel based SVM anomaly detectors on the UNM data ........................................ 69
4.6 Window size impact on the STIDE frequency kernel based SVM anomaly detectors on the UNM data ........................................ 70
4.7 Window size impact on the Markov Chain kernel based SVM anomaly detectors on the UNM data ........................................ 70
5.1 Comparison of STIDE, Markov Chain, STIDE kernel based and Markov kernel based SVM anomaly detectors on the UNM data ............ 75
5.2 Comparison of polynomial kernel, RBF kernel, STIDE kernel and Markov Chain kernel based SVM anomaly detectors (ratio threshold = 0.1) on the UNM data ........................................ 76
5.3 Comparison of STIDE, Markov Chain, STIDE kernel based one class
SVM, Markov Chain kernel based SVM anomaly detectors on the
DARPA data ................................................................. 77

6.1 Comparison of FSM, SVM and SVM with FSM filter and feature selection 86
6.2 Comparison of EFSA, SVM and ESFA guided SVM on the FTP pro-
cesses in the DARPA Lincoln Lab data ............................... 90
6.3 Comparison of EFSA, SVM and ESFA guided SVM on multiple pro-
cesses in the DARPA Lincoln Lab data ............................... 91

7.1 Response time of agents .............................................. 104
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>A sample of generic specification [68]</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>A simplified specification for FTP process [68]</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Detection and blind area of STIDE detector</td>
<td>65</td>
</tr>
<tr>
<td>4.2</td>
<td>Markov chain constructed from training data “abcabcbcacabc” using window size 3</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>Claimed coverage</td>
<td>68</td>
</tr>
<tr>
<td>4.4</td>
<td>Actual coverage of Markov detector</td>
<td>68</td>
</tr>
<tr>
<td>6.1</td>
<td>Source code of a piece of program</td>
<td>83</td>
</tr>
<tr>
<td>6.2</td>
<td>FSM translated from the code</td>
<td>83</td>
</tr>
<tr>
<td>6.3</td>
<td>Training phase of the hybrid anomaly detector</td>
<td>88</td>
</tr>
<tr>
<td>6.4</td>
<td>Testing phase of the hybrid anomaly detector</td>
<td>88</td>
</tr>
<tr>
<td>7.1</td>
<td>Generation of anomaly intrusion detection profiles</td>
<td>102</td>
</tr>
<tr>
<td>7.2</td>
<td>The interface of the agent based anomaly detector</td>
<td>105</td>
</tr>
<tr>
<td>A.1</td>
<td>SFT for FTP bounce attack</td>
<td>111</td>
</tr>
<tr>
<td>B.1</td>
<td>Diagram for FTP bounce attack</td>
<td>113</td>
</tr>
<tr>
<td>B.2</td>
<td>Generated CPN for FTP bounce attack</td>
<td>114</td>
</tr>
<tr>
<td>B.3</td>
<td>Diagram for TRINOO DDOS attack</td>
<td>116</td>
</tr>
<tr>
<td>B.4</td>
<td>SFT for TRINOO DDOS attack</td>
<td>117</td>
</tr>
<tr>
<td>B.5</td>
<td>Generated CPN for TRINOO DDOS attack</td>
<td>118</td>
</tr>
</tbody>
</table>
ABSTRACT

Anomaly intrusion detection normally has high false alarm rates, and a high volume of false alarms will prevent system administrators identifying the real attacks. Machine learning methods provide an effective way to decrease the false alarm rate and improve the detection rate of anomaly intrusion detection. In this research, we propose a novel approach using kernel methods and Support Vector Machine (SVM) for improving anomaly intrusion detectors' accuracy. Two kernels, STIDE kernel and Markov Chain kernel, are developed specially for intrusion detection applications. The experiments show the STIDE and Markov Chain kernel based two class SVM anomaly detectors have better accuracy rate than the original STIDE and Markov Chain anomaly detectors.

Generally, anomaly intrusion detection approaches build normal profiles from labeled training data. However, labeled training data for intrusion detection is expensive and not easy to obtain. We propose an anomaly detection approach, using STIDE kernel and Markov Chain kernel based one class SVM, that does not need labeled training data.

To further increase the detection rate and lower the false alarm rate, an approach of integrating specification based intrusion detection with anomaly intrusion detection is also proposed. Anomaly intrusion detectors tend to classify the behavior patterns that do not exist in training data as anomalies. When the training data does not cover all the normal behaviors, there will be a high false alarm rate using anomaly detection. Using specification based method as a filter and feature selection tool, the accuracy rate of anomaly detection intrusion can be further improved.

This research also establish a platform which generates automatically both misuse and anomaly intrusion detection software agents. In our method, a SFT representing an intrusion is automatically converted to a Colored Petri Net (CPNs) representing an intrusion detection
template, subsequently, the CPN is compiled into code for misuse intrusion detection software agents using a compiler and dynamically loaded and launched for misuse intrusion detection. On the other hand, a model representing a normal profile is automatically generated from training data, subsequently, an anomaly intrusion detection agent which carries this model is generated and launched for anomaly intrusion detection. By engaging both misuse and anomaly intrusion detection agents, our system can detect known attacks as well as novel unknown attacks.
CHAPTER 1. Introduction

In the early days of computing, security problems were not a concern for system and software design and implementation. The computer security problem appeared in the 1970s are mainly attackers broke into users' accounts, and private and important information were stolen. From then on, with more and more sharing of information and the thriving of the Internet, computer security problems became a vital concern.

An informal definition of computer security is given by Garfinkel and Spafford in 1996 - A computer is secure if you can depend on it and its software to behave as you expect [19]. This definition is broad and includes hardware damage as well as software bugs. Currently, a more formal definition narrows down the goal of computer security to three categories: confidentiality, integrity and availability [50]. Confidentiality means data are not disclosed to unauthorized users. Integrity means there is no unauthorized modification. Availability means computer resources, information and services are not withheld to authorized users when they are needed. A computer system is secure if the three goals are achieved.

Security policy is a set of rules and principles to guarantee integrity, availability and confidentiality in a computing system. Intrusion prevention includes defining a security policy and implementing the system according to the security policy. A security policy is specified first by system administrators, then it is implemented through appropriate installation and configuration of systems and also enforced using many security measures, such as authentication, encryption, access control, firewall, etc. Computer security measures and controls help an information system to achieve the goals of confidentiality, integrity and availability, and protect against unauthorized information disclosure (information leakage), modification and destruction of information systems and data (information corruption) and denial of service attack (information denial).
The ideal approach for computer security is to establish and implement a security policy that prevents any intrusions using these security measures. However, the preventive measures are not always sufficient because of the following reasons [64]:

- Bug free software is almost impossible. It is an extremely difficult task to design such software, and no one wants to make such a huge effort to design totally bug free software, except maybe in the military organization.

- It is difficult to change users and organizations' behaviors. For example, we can require users and organizations to follow some security measures, such as setting passwords that are not easy to guess, however, there is no guarantee that such policy will be followed by all users.

- Human errors in operations and maintenances are unavoidable. These errors can cause serious security loopholes.

- There are a lot of insecure systems installed all over the world. It is impossible to patch all of them to make them secure.

- The security measures and controls themselves can be compromised. For instance, the cryptographic algorithms can be cracked with sufficient time and computing power.

- It is almost impossible to prevent insider attacks. Insiders have more access to the system than outside attackers. It is very difficult to prevent them from abusing their rights.

- Even though it is possible to set up a totally secure system, the cost of these security measures and efficiency degradation because of these security measures will discourage people from doing this.

Because of these reasons, intrusion detection is prompted as a second line of defense to ensure security of a computer system. Intrusion detection includes monitoring the events happening in the monitored system, and detecting intrusions when they happen. In the following sections, an overview of intrusions and intrusion detection system is given first.
Then, the problems in the field of intrusion detection are identified and the contribution of our research is summarized.

1.1 Intrusion and intrusion detection

An intrusion is a violation of a security policy. For example, if one of the security policies for a system defines that the users' accounts on this system can only be accessed by authorized users, then the action of sneaking into these users' accounts and transferring these users' files is an intrusion. If a security policy of a system defines that some web pages in the system should be available to Internet users, then the action that impedes Internet users from getting these web pages is considered as an intrusion, namely, a denial of service attack.

System and user behaviors need to be monitored and analyzed to detect intrusions. The monitoring is often done through audit data that is generated by operating systems and applications. Operating systems and many applications regularly log an incredibly large volume of data in their log files. In the UNIX operating system, syslogd is used by kernels, daemons and applications to log useful messages at various levels of details. The wtmp and utmp files on UNIX system provide records about users' log-ins, log-outs and sessions information. Some additional audit tools are also provided to record system activities. For example, strace and Basic Security Module (BSM) in UNIX system can record system call traces. The tcpdump program can be used to record all network traffics at the TCP level, while uptime, df and netstat can be used to provide system usages, disk usages and network connections.

Intrusion behaviors can be discovered by analyzing these audit files. However, the large volume of audit files and the complicated nature of intrusions make it impossible to analyze these audit data to find out attacks manually, calling for an automatic approach to analyze audit files and detect intrusions, namely an intrusion detection system.

For intrusion detection, the basic assumption is that intrusion behaviors are different from normal behaviors and are detectable. Followings are six main types of intrusions, and how they can be detected [61]:

1. Attempted break-ins, i.e., attackers try to break into users' accounts or systems, are
detected by atypical behavior profiles or violations of security constraints.

2. Masquerade attacks, i.e., attackers masquerade as normal users, are detected by atypical behavior profiles or violations of security constraints.

3. Penetration of the security control system, i.e., attackers break into security administration or control system such as Intrusion Detection System and Access Control table, is detected by monitoring specific patterns of activities.

4. Leakage, i.e., system resources such as CPU time and hard disk space are stolen, is detected by atypical use of system resources.

5. Denial of service, i.e., services provided are stopped, interrupted or slowed down by attacker, which is detected by atypical use of system resources.

6. Malicious use, i.e., inside attackers abuse their privileges, is detected by atypical behavior profiles, violations of security constraints, or use of special privileges.

Computer attacks are becoming more and more complicated. Temporal and spatial features of an attack must be considered to detect it accurately. One successful attack behavior may have a long time span, which means an intrusion detection system needs to examine a series of behaviors for a long time to detect such attack. One attack also may involve cooperation from several different sites, which means correlation of behaviors from several sites is needed to detect such attack.

Generally speaking, intrusion detection approaches can be categorized into misuse detection, anomaly detection, and specification based detection. Misuse detection works by comparing network traffic, system call sequences, or other measurable quantities to known attack patterns called attack signatures. Although misuse detectors can be quite successful in detecting attacks using known signatures, they cannot detect novel attacks and often fail to detect slight modifications to existing attacks. When a new attack surfaces, an attack signature must be constructed (usually by an expert), and the misuse detector must be reconfigured to include the new signature.
Anomaly detection systems "learn" what constitutes normal behaviors from systems' or users' past behaviors, developing sets of normal profiles that are continually updated. An anomaly detector looks for deviations from the known normal behavior; when the deviation exceeds certain predefined threshold, an alarm is raised. Unlike misuse detectors, anomaly detectors are able to detect previously unseen attacks. However, anomaly detectors can suffer from a high false alarm rate; this can occur due to rare but legitimate behaviors (e.g., actions taken to respond for events of dropped packets or failed servers) that are not covered by the definition of normal system behaviors.

Specification based detection uses specifications, which are derived from system designs and configurations. The system activities are compared to the specifications of the software system. Any activities that do not comply to the specifications are identified as intrusions. For instance, if the specifications define that in FTP daemon, files should be downloaded from the /public directory, then any downloading from other directories are classified as intrusions. Specification based intrusion detection approach can find out novel attacks and does not need a training procedure to build a normal profile. However, the specification of a system can be complicated and needs tremendous human effort to build it.

1.2 Thesis statement

This dissertation addresses the detection accuracy problems involved in intrusion detection and systematic methods for intrusion detection system design. Specifically, it answers these questions:

- How can we improve the detection rate and reduce the false alarm rate of anomaly intrusion detection?

- Is it possible to take advantages of the strengths of various intrusion detection approaches without degrading the overall performance? How can we integrate these intrusion detection approaches?

- How can we efficiently and effectively design and implement an intrusion detection sys-
tem to detect known and novel attacks?

1.3 Contributions

The first contribution of this research is a novel approach using kernel methods and Support Vector Machine (SVM) for improving anomaly intrusion detectors’ accuracy [77]. In previous work, machine learning techniques have been used in the intrusion detection field for learning intrusion signatures, building normal profiles, selecting important features and so on. SVM is a newly emerging method in machine learning research; kernel method has a longer history, but it is recently found to be very effective and efficient when combined with the SVM method. This research develops two kernels specially suitable for intrusion detection applications, namely STIDE and Markov Chain kernels. They are combined with two class SVM for intrusion detection. This approach improves the detection rate and lowers the false alarm rate for anomaly intrusion detection.

The second contribution of this research is an anomaly detection approach, using STIDE kernel and Markov Chain kernel based one class SVM, that does not need labeled training data. Generally, anomaly intrusion detection approaches build normal profiles from labeled training data [76]. However, labeled training data for intrusion detection is expensive and not easy to obtain. One class SVM approach provides a reliable, accurate and efficient way for anomaly detection.

The third contribution of this research is the integration of specification based intrusion detection with anomaly intrusion detection to increase detection rates and lower false alarm rates. Anomaly intrusion detection can detect novel intrusions. However, the main problem of anomaly intrusion detection is its high false alarm rate. Anomaly intrusion detectors tend to classify the behavior patterns that do not exist in training data as anomalies. When the training data does not cover all the normal behaviors, there will be a high false alarm rate using anomaly detection. Specification based intrusion detection system aims to capture all the legitimate behaviors of a system. Thus, a fully developed specification based intrusion detection system does not have false alarms. The main problem of specification based approach
is the design of specifications needs tremendous human effort and is very time consuming. This dissertation proposes an intrusion detection approach that combines the strengths of specification based intrusion detection and anomaly intrusion detection and alleviates their weaknesses.

The fourth contribution of this research is the design and implementation of an intrusion detection system, which generates automatically both misuse and anomaly intrusion detection software agents. In our method, a SFT representing an intrusion is automatically converted to a Colored Petri Net (CPNs) representing an intrusion detection template, subsequently, the CPN is compiled into code for misuse intrusion detection software agents using a compiler and dynamically loaded and launched for misuse intrusion detection [74, 75]. On the other hand, a model representing a normal profile is automatically generated from training data, subsequently, an anomaly intrusion detection agent which carries this model is generated and launched for anomaly intrusion detection. By engaging both misuse and anomaly intrusion detection agents, our system can detect known attacks as well as novel unknown attacks.

1.4 Roadmap of this dissertation

The remainder of the dissertation is organized as follows. Chapter 2 reviews the related work on intrusion detection systems. An overview of the three main methods for intrusion detection, namely, misuse intrusion detection, anomaly intrusion detection and specification based intrusion detection, is given. The arguments of the issues and problems related to these three kinds of intrusion detection methods are presented. The effort of combining various intrusion detection methods, namely, hybrid intrusion detection systems, are also introduced. We also review infrastructure for intrusion detection systems and agent based intrusion detection systems in this chapter.

Chapter 3 introduces the related technology used in this dissertation, such as Support Vector Machine (SVM), Kernel Method, Software Fault Tree, Colored Petri Net and specification based method. These technologies have been developed in research and development in our project.
Chapter 4 presents our kernel approach, combined with two class SVM, for classification of system call sequences. Two kernels, STIDE kernel and Markov Chain kernel are presented. An evaluation is also presented for the two conventional anomaly detectors, the Markov Chain anomaly detector and the STIDE anomaly detector, with comparison to our new two class SVM based anomaly detectors.

Chapter 5 shows how to use one class SVM and kernel methods for unsupervised learning for anomaly intrusion detection. The results are compared with two conventional anomaly detectors, STIDE and Markov Chain anomaly detectors, and two class SVM based anomaly detectors.

Chapter 6 introduces specification guided anomaly detection. An approach to integrate the specification based intrusion detection and anomaly intrusion detection is presented. A technique of using specifications to reduce features for anomaly detection is also introduced.

Chapter 7 presents the automatic generation of misuse and anomaly intrusion detection agents. Design and implementation of an intrusion detection system prototypes will be presented, which automatically generates misuse intrusion detection software agents from the SFT specification of intrusion, using the CPN as design templates for intrusion detection, and automatically generates anomaly intrusion detection software agents from training data.

Chapter 8 offers conclusions and ideas for future work.
CHAPTER 2. Related work

This chapter describes related work about approaches for intrusion detection, including anomaly intrusion detection, misuse intrusion detection, specification based intrusion detection, and hybrid intrusion detection which integrates some or all of these approaches, as well as infrastructures for intrusion detection systems.

2.1 Misuse intrusion detection system

Misuse intrusion detection systems use well defined intrusion patterns (intrusion signatures) to represent attacks and match intrusion patterns with audit trails to detect attacks. The representation of intrusion signatures has evolved from the simple forms such as fixed strings, regular expressions and rules, to scenario models and state transitions modeling complicated attacks.

The first generation of misuse intrusion detection systems uses rule based misuse intrusion detection, by which audit records are matched to expert rules. The second generation of misuse intrusion detection systems either use model based intrusion detection, by which scenario models are constructed to represent the features of intrusion behaviors, or state transition analysis, by which states and state transitions in a system leading from an initial state to a compromise state are modeled.

2.1.1 Rule based expert system

For very simple misuse intrusion patterns, only fixed strings and regular expressions are used to match audit trails and identify intrusions. However, most intrusion patterns are complicated, thus, expert systems are usually integrated with rule based intrusion detection
systems since expert systems provide powerful reasoning ability to reason from facts and rules [58].

2.1.1.1 P-BEST

P-BEST, a general purpose forward chaining expert system, is first used by MIDAS to define rule sets and facts for misuse intrusion detection [59]. By analyzing audit trails, facts related to the misuse intrusion are extracted and asserted in the P-BEST fact base and an inference engine in P-BEST is triggered to derive new facts.

P-BEST rule sets and fact base are precompiled to C modules, and C functions can be called from P-BEST. This feature provides convenience to the system development as C language is very powerful and supported in almost all systems.

2.1.1.2 RUSSEL

RUSSEL is a language designed for searching arbitrary patterns of records in a file when misuse detection system ASAX is developed [48]. A RUSSEL program consists of rule sets and global variables. Records are analyzed sequentially in such a way that when a record is read in, some rules are triggered to alter global variables, raise alarms, write reports, and activate new rules. When all triggered rules are finished, the next records are read in. ASAX also supports C routines, either system defined or user defined, which makes it more flexible.

Unlike MIDAS [59], which needs to rewrite the P-BEST program for different operating systems, ASAX is independent of operating systems by using a format adaptor to translate native file into a NADF (Normalized Audit Data Format) format. This feature means ASAX can analyze data streams of any format. ASAX uses this feature to implement a distributed misuse intrusion detection system to do online analysis and correlation of multiple distributed data streams.

MIDAS and ASAX use rules to represent detection of an attack in the system. These rules are very system dependent. To design and update these rules, security experts are needed, and also extensive knowledge about the system is needed. Furthermore, current rule designs uses
an ad hoc method, which is inefficient. Thus, a high level abstraction of intrusion detection and a more efficient method of designing rule bases are called for.

2.1.2 High level representation of intrusion detection

Compared to misuse intrusion detection rules, high level representation of intrusion detection is more system independent, efficient and easier to design. Model based intrusion detection approaches use high level representation of intrusions, such as State Transition Analysis and Colored Petri Nets.

2.1.2.1 Model based intrusion detection

Garvey et al. proposed model based intrusion detection which verifies attack scenarios from observed activities [20]. Attack scenarios, composed of a sequence of activities making up the attack, are stored in a scenario database. Model based intrusion detection systems use anticipators which predict the next step in the scenario, use planners which translate the prediction to what would occur in the audit trail and use interpreters to actually search for the data in the audit trail. When activities are found leading to an intrusion depicted by the attack scenario, it signals an alarm.

This approach is very efficient because it always searches for activities that are expected from the last scenario and large amounts of noises can be neglected. Also it is possible to take preventive measures when a compromise is predicted.

2.1.2.2 State Transition Analysis

Porras et al. proposed to use State Transition Analysis approach to model penetration stages in a computer system [51]. State Transition Analysis views an intrusion as a sequence of actions that leads from an initial state, through a series of intermediate states, to a final compromised state of a computer system. A State Transition Diagram is used to model the penetration as a sequence of state transitions.

In State Transition Analysis, only the key events which will prevent the intrusions from happening if absent will be modeled. So, the minimum requirements for an intrusion are
identified. By recognizing only the minimum number of key events for a penetration, intrusion detection is more efficient and also, some variants of the analyzed attack can be modeled.

A State Transition Analysis Tool is developed to look for known penetrations in the audit trail. It consists of an Audit Record Processor, a Knowledge Base, an Inference Engine and a Decision Engine. The Inference Engine maintains a state table to keep track of all possible penetrations. Audit trails are analyzed by the Audit Record Processor and relevant information is sent to the Inference Engine to trigger a state transition. The Decision Engine informs a compromise or a danger of compromise, and take actions to prevent it from happening.

By State Transition Analysis, some level of abstraction is achieved over the rule based methods. For the same type of intrusion, the state transition model is always the same in State Transition Analysis for different operating systems, whereas the rules representing the same intrusions can be very different.

The State Transition Analysis method has several advantages:

First, the rules are easily created and updated. Transition rules are intuitive and concise. Thus, unlike rule based intrusion detection, State Transition Analysis methods do not need experts of the rule system and the operating systems for designing and updating rules.

Second, this method focuses on events that cause state transitions instead of low level audit records. So it achieves high level abstraction and is independent of audit records, making it easy to port to other systems.

Third, the next steps in penetration can be predicted, thus, measures can be taken to prevent a compromise from happening.

2.1.2.3 Colored Petri Nets

Kumar and Spafford proposed to use Colored Petri Nets (CPNs) for misuse intrusion detection in their Intrusion Detection in Our Time (IDIOT) project [33]. They considered the misuse intrusion detection as a pattern matching procedure. The transitions through CPN places lead from initial state to final intruded state. CPN is a Turing Machine language which has the features of generality, conceptual simplicity and graphical representability, and it is more powerful than regular expressions.
Used in pattern matching for misuse intrusion detection, CPN is not only expressive but also attains better abstract level. A CPN is more like a virtual machine for matching input events with intrusion signatures, and it does not depend on operating systems. In other words, it represents signatures in an operating system independent format and works on abstract audit trails. Thus, the CPN approach has good portability.

2.2 Anomaly intrusion detection system

An anomaly is a user or system behavior or a system status that deviates from the normal state. Anomaly detection can be described as finding anomalies in a system. In other words, an anomaly detector looks for deviations from the normal; when a deviation exceeds some pre-defined threshold, an alarm is raised. Unlike misuse detectors, anomaly detectors are able to detect previously unseen attacks.

A normal profile that describes the normality of the monitored system is always required for an anomaly detection system. For different kinds of subjects in a system, such as sessions, users, groups, programs and network traffics, different measures are used to describe the normality. For example, log-in time, log-out time, log-in attempts and user commands are used for profiling sessions; user commands, their arguments and key strike speeds are used for profiling users’ behaviors; network outbound and inbound flow byte counts, source and destination IP addresses and connection start and end times are the parameters used in profiling network traffics. An anomaly detection system requires a normal profile because it detects anomalies based on the deviation from the normal profile, i.e. all system states varying from the normal profile by statistically significant amounts are flagged as intrusive attempts.

Dorothy Denning first describes building an activity profile of normal usage by employing a statistical model [14]. After that, many researchers have studied how to establish normal profiles for a monitored system, some of the efforts are as following:

IDES/NIDES The anomaly detector in IDES/NIDES followed Denning’s statistical model [27]. The profiles of individual subjects, such as individual users and user groups, are generated by observing the subjects in a working environment for a period of time.
Then the system is monitored and the observed data are compared to the profile to assess whether the deviation is significant enough to signal an anomaly.

**HAYSTACK** The anomaly detector HAYSTACK, developed at the University of California, Davis, compares a session vector describing session properties, such as files opened for reading, number of connections in one minute and number of log-in attempts, to a standard vector to indicate whether an anomaly occurs [61]. The standard vector functions as a normal profile of sessions, and it describes the normal characteristics of a session.

**Computer immunology (STIDE)** STIDE, developed at the University of New Mexico (UNM), is inspired by an immunology “sense of self” concept, where living organisms distinguish themselves from any foreign particles [17]. This approach was developed for UNIX system programs. The “self sense” of a UNIX program is established by analyzing the system call sequences generated from normal executions of the UNIX system program. Thereafter, the system call sequences generated in a working environment are collected and compared to the normal database of system call sequences. The “self sense” here is essentially another expression for “profile”.

**IBL** Lane and Brodley use Instance Based Learning (IBL) to learn “characteristic sequences” of actions generated by users by forming a user profile [34]. Actions of users are taken to be UNIX shell commands with arguments; and characteristic sequences are used to represent user actions. For each user, a dictionary is formed from characteristic sequences collected from a presumably users’ intruder-free history. For each new sequence, a similarity value is calculated by comparing it to the dictionary and a threshold is used to determine whether it is anomalous.

**JAM** The JAM project in Columbia University utilizes data mining techniques to compute features that are significant to detect intrusions, and uses a decision tree learning software, RIPPER, to learn rules from training data to detect intrusions [36].

**Neural network** Ghosh et al. introduce a back propagation neural network to build profiles of software behaviors [21]. Program inputs and program states are used as the inputs
to the back propagation network, which is first trained using normal and intrusive data. This approach predicts subsequent behaviors from previously known behaviors, and flags all system states having significant deviations from predicted or expected behaviors as intrusive attempts.

**Support Vector Machine** Mukkamala et al. in New Mexico Technology University performed research on anomaly intrusion detection using Support Vector Machine (SVM) \[66\]. They constructed network based and host based features for intrusion detection, represented the audit data in the form of feature vectors, and fed these feature vectors to SVM for training and classification.

As described above, many researchers have made tremendous efforts to explore various approaches for anomaly detection. These approaches for building anomaly detectors can be categorized into two types. First, the statistical model based method, which employs a statistical model to construct a profile for users, groups, programs and systems. The representative methods are IDES/NIDES and HAYSTACK. Second is the machine learning or data mining method. In this method, profiles (in the form of rules) are built from training data. The representative methods are JAM, Neural network, IBL, computer immunology method (STIDE) and Markov Chain method. In the following sections, these approaches are discussed in more detail.

### 2.2.1 Statistical model

#### 2.2.1.1 Denning’s statistical model

Dorothy Denning first described building an “activity profile” of normal usage and flagging all behaviors deviated significantly from the normal profile as anomalies. She also proposed to employ a statistical model to construct a normal profile \[14\]. The concept about using a normal profile for detecting anomalies in a system is essential for anomaly intrusion detection.

There are six main components in Denning’s model:

**Subject** Initiator of activities, such as users, groups of users, or system.
Objects Resources managed by system, such as files, programs, printers and databases.

Audit records Generated by the system when subjects attempt or perform some actions on objects, such as user log-in, command execution, file access, etc.

Profiles Created from observed activities in system, profiles describe the behavior of subjects with respect to objects in terms of statistical metrics and models. Normal profiles are models generated when the system is in normal operation.

Anomaly records Records of detected abnormal behaviors.

Activity rules Rules of conditions triggering actions, such as updating profiles, detecting abnormal behavior, and producing reports.

This subject-object model is analogous to an access control matrix which describes what access permission a subject has over an object. The dissimilarity between this model and an access control matrix is that:

The information (access permission) in an access control matrix is defined by system administrators, so it is a prior knowledge; while the information in the usage matrix (usage statistics) is generated from observed activities, so it is learned knowledge. For instance, an access control matrix can set a permission whether a user X can access a file A or not, while usage matrix describes statistics from past behaviors that the user X has never accessed file A in the past (thus the user X accessing file A will be considered as an anomaly).

Three kinds of statistical metrics used in this model are:

Event counter: number of audit records corresponding to a specified event, such as the number of log-ins during a period, number of password failures during a minute, and the number of commands used during one session.

Interval time: length of time between two audit records which represents two related events. For example, the time period between log-in time and log-out time of one user.

Resource measure: resource usage of certain activities. For example, CPU time consumed by a program and number of files opened during one session.
2.2.1.2 Main issues for statistical model

The essence of the statistical models is to monitor variables changing over time, such as log-in and log-out time, resource usage (CPU, memory, disk, etc.), user commands, command parameters, network inbound and outbound flow byte counts, and report the deviation from the average.

To establish a statistical model based anomaly detector, the main issues to consider are what measures to use in profiling and what statistical model to use for assessing anomaly. For different subjects described in a profile, such as sessions, users, groups, programs and network traffic, different measures are used. For example, log-in time, log-out time, number of log-in attempts, and user commands are normally employed to describe normality of a session. Also, the statistical models for assessing anomaly is different for various measures.

In statistical models, the measures used in profiling can be categorized into four classes: categorical, continuous, intensity, and event distribution [14].

*Categorical measures* use discrete values from a categorical set, such as command name, source and destination host identity and port number.

*Continuous measures* use numeric values or ordinal numbers, such as session time, number of files opened and CPU time used by an application.

*Intensity measures* track intensity of activities. These measures reflect the intensity of the event stream (number of events per unit time) over time intervals. For example, web page access times during one minute and number of bytes transferred in one second.

*Event distributions* are the “meta-distribution” of the measures affected by recent events. For example, an *ls* command in an FTP session affects the directory measure, but does not affect measures related to file transfer.

In Denning’s method [14], deviations are calculated and anomalies are decided using five statistical models: Operational Model, Mean and Standard Deviation Model, Multivariate Model, Markov Process Model and Time Series Model.

*Operational Model* assumes the new observation of a statistical measure $x$ is defined to be an abnormality if it is out of a fixed limit. For example, the number of failed log-in failures
during a specified time interval greater than 10 indicates abnormality.

*Mean and Standard Deviation Model* assumes a new observation of a statistical measure $x$ is defined to be abnormal if it falls outside of $\text{mean} \pm d \ast \text{stdev}$, where mean and stdev is the mean value and standard deviation of past events and $d$ is the confidence parameter decided by an administrator (the probability of a value falling outside of the above range is $1/d^2$ according to Chebyshev’s inequality [31]). For example, suppose we observed the average session time for a user is 80 minutes and the standard deviation of past log-in times is 10 minutes. Assuming $d = 4$, if the session time is less than 40 minutes or more than 120 minutes, it indicates abnormality.

*Multivariate Model* assumes two or more measures $x_1, x_2, \ldots, x_n$, need to be correlated to detect abnormality. This model is based on other models. For example, log-in frequency $x_1$ and session time $x_2$ may be correlated to indicate abnormality since frequent log-ins with short session time is a strong indication of an attack. When log-in frequency and session time are both out of normal range according to operational model or mean and standard deviation model, it suggests abnormality.

*Markov Process Model* assumes a new observation $x$ is abnormal if its probability as determined by the previous state and the transition probability matrix is lower than a fixed threshold. For example, suppose the monitored subject is a program, if the current observed system call from this program is $s$ and the previous system calls from this program are $s_1s_2\ldots s_n$, assuming the anomaly threshold to be 0.2, then abnormality needs to be reported if the probability of generating a system call sequence $s_1s_2\ldots s_n s$ from the program is below 0.2.

*Time Series Model* assumes a new observation $x$ is abnormal if its probability of occurrence at the time is low. Interval timers are used to measure the intervals between events. When the probability of the specific order and interval times of the observation of event series $x_1, x_2, \ldots, x_n$ are below some pre-defined threshold, abnormality should be reported. This model, which combines event counter with interval time, is actually a variation of multivariate model.
2.2.1.3 Intrusion detection systems using statistical model

Intrusion Detection Expert System (IDES), developed in SRI around 1984, uses the anomaly detection models described above [27]. IDES is a real time anomaly intrusion detection system that monitors computer behaviors and learns normal profiles for users and group of users. The learned profiles are updated daily giving more weight to most recent behaviors. For each newly generated audit records, an overall statistical score is calculated. The score summarizes all relevant measures and reflects the similarity with the normal profile. An alarm is reported when the score exceeds a certain pre-defined threshold.

IDES profiles the behaviors of individual users or group of users. To extend the profiling to application, SRI International also conducted Safeguard project to research on application based profiling. Under safeguard model, a statistical score is computed for the operation of applications and represents the degree to which current behavior of the application corresponds to its established patterns of operation.

HAYSTACK, developed for Air Force computing platform at Lawrence Livermore Laboratory and University of California Davis in 1988, is one of the earliest intrusion detection systems [61]. It includes an anomaly detector that monitors and analyzes user activities for anomalies. In HAYSTACK anomaly detector, a vector is used to describe the statistical measures, such as session duration, number of files opened for reading; a threshold vector is used to represent the threshold for each statistical measure; a weight vector is used to give different measure of importance to each attribute for some particular attacks and a Bernouli vector is used to represent the attributes that are out of range compared to the threshold vector.

2.2.1.4 Advantages and disadvantages of statistical model

Using a statistical model for anomaly detection, no prior knowledge about intrusions or systems is needed for establishing an intrusion detection system, which is better than misuse anomaly detection as domain experts for specifying attack signatures and designing rules are needed. Furthermore, using the statistical model, novel intrusions can be detected.

However, the environmental changes in systems impact the effectiveness of the statistical
model based anomaly intrusion detection and causes many false alarms. In a system, new users are added, some services are removed and new machines are hooked up frequently. All these changes impact activities observed in a system. With the environmental changes, false alarm rates raise significantly since anomaly detection systems flag all behaviors with significant deviations from the profile as abnormal. The true alarms are sometimes buried with false alarms and the IDSs become useless. Statistical model based anomaly detection must update the profiles to reflect the environmental changes.

Currently, two approaches are widely used for updating normal profiles. One method is to generate and use a new profile regularly; another method is to merge new observed activities into the old profile, giving more weight to recently observed activities. However, these two methods of updating profiles are not faultless: attackers can slowly insert their intrusive behaviors and train the normal profiles to accept their behaviors as normal. This problem is still an open problem for user based anomaly detection. It is easy for an attacker to disguise as a normal user and slowly insert their intrusive behaviors to misguide an anomaly detection system. However, it is less a problem for system call based anomaly detection since the system call sequences of programs are not easy to be manipulated, i.e., being changed slowly.

2.2.2 Computer immunology based anomaly detection

Natural immune systems can detect foreign objects from self and attack foreign objects when encountered. Forrest et al. from the University of New Mexico proposed to incorporate the mechanisms and algorithms used by natural immune systems in computer intrusion detection [17].

2.2.2.1 Sense of self for Unix process

The first thing to do for any immunology system is to establish a “sense of self”. In UNIX system, a process with root privilege is likely to cause serious damages. Thus, to establish “sense of self” definition of UNIX privileged process is essential for detecting anomalies in UNIX system. Forrest et al proposed to use local ordering of system calls (fixed size system call sequences) observed in execution of processes to define a UNIX process’s “self”. A fixed
size normal system call sequences database needs to be established from the process execution. To test if a process is normal, the process's system call sequences are compared with the normal system call sequence database. If the percentage of abnormal system call sequences is above some predetermined threshold, an abnormality alarm is generated.

Thus, system call sequences from program execution is used as a definition of "self". Usually, a definition too narrow will result in many false positives (false alarms), a definition too broad will result in many false negatives (undetected intrusions). Forrest et al. demonstrated that this notion of behavior identity - using system call sequence to define UNIX process's self - is suitable because of following two reasons:

First, the sequence of system calls executed by a program is locally consistent during normal operation. Conditions and function calls will change the relative orderings of the invoked system calls but not necessarily add variation to short term correlation. The experiment showed that new system call sequences only emerge in the early part, say first 3000 system calls, during the execution of a sendmail process (experimented using window size of 6). The experiment also showed that although the normal system call sequence database formed in this way only covers $5 \times 10^{-5}\%$ of the total possible system call sequences (from analyzing source code of sendmail program), the normal database is very stable for a particular architecture, software configuration, local administrative policies, and usage patterns.

Second, some unusual short sequences of system calls will be executed when a security hole in a program is exploited. Most of the attacks to UNIX privileged process will change some system calls in execution. For example, the syslog attack uses the syslog interface to overflow a buffer in sendmail process, and cause sendmail process to execute a piece of an attacker's code. The attack will vary sendmail's system call sequences by inserting system call sequences from the attacker's code.

Some attacks to UNIX process might not be detected by only examining their system call sequences. One example is race conditions attack, which steals system resources from a privileged program by taking advantage of the time lag between checking the availability of a resource and actually using the resource of a privileged program. As long as the privileged program does not find an error, the system call sequences of the privileged program will not
reflect any abnormality. Those intrusions might be detectable by examining other aspects of a process's behavior, and need a revised definition of self [21].

2.2.2.2 Computer immunology and intrusion detection

Forrest et al. pointed out some computer immunology concepts and methods can be used for intrusion detection [17]:

First one is to use sequence to represent characteristics of "self" and to build a normal sequences database (used for anomaly intrusion detection for UNIX privileged processes). Second one is the similarity value of a characteristic sequence to the normal sequence database, which is used to decide whether a test sequence is abnormal. In the immunology field, exact match, approximation and partial match are all used to calculate the similarity value. In UNIX intrusion detection, only exact match are systematically studied. Another one is that memorial cells in immunology are used to memorize some knowledge about foreign objects to improve the accuracy of the judgement and accelerate the reaction, and intrusion detection system can borrow this idea and use learned rules to memorize some knowledge about attacks, and thus integrating misuse and anomaly detection together.

2.2.3 Machine learning and data mining methods for anomaly intrusion detection

Anomaly detection can be formulated as a learn-by-example machine learning task. A learn-by-example learning procedure generally begins with a set of examples with labels, and generates a model (i.e., classification rules) to classify future examples. In anomaly detection learning, the audit data from previously observed activities serve as examples. Several machine learning approaches are proposed to profile user and program behaviors and recognize previously unseen behaviors.

2.2.3.1 IBL

Lane and Brodley applied Instance Based Learning (IBL) technique to learn a profile for each user from users' input streams [34]. In Denning's statistical based intrusion detection model, a user's behavior can be characterized using resources consumed by the user, user's
typing rate, command issue rate and counts of particular commands employed in one session, whereas Lane and Brodley’s approach emphasizes on one interesting assumption - “a user responds in a similar manner to similar situations, leading to repeated sequences of actions”. This hypothesis holds in general, while there are times when users respond to abruptive events, and cause the input stream injected with unusual user inputs, which may cause false alarms.

Lane and Brodley used sequences to characterize a user’s input stream - including commands and their arguments. File names in the user input are neglected because they can change significantly. Furthermore, these character sequences are segmented into fixed size overlapping characteristic sequences of tokens. Their experiments showed the size of the characteristic sequences had positive correlation with detection rate [34].

The motivation of using segmented sequences for user input profiling is similar to that of using a sliding window on system call sequences for program profiling used in computer immunology approach [17]. The segmented fixed length sequences feature temporal relationships between single events. Furthermore, abnormality is usually temporally localized, which means in an abnormal sequence there always exist some relatively small length sequences that are very dissimilar with normal sequences. Also, by using fixed size sequences, it is possible to reduce false alarm rate since small noises are filtered out because the dissimilarity of system calls in a segmented fixed size sequence needs to exceed some threshold to be classified as an anomaly.

The similarity comparison between sequences is basically a string matching problem. Lane and Brodley defined similarity measures that yield a higher score for more similar sequences and biased toward adjacent identical tokens rather than identical tokens separated by some non-matching intermediate tokens.

A set of instances are used to exemplify the user profile. However, instances library get extremely large, so it needs to be trimmed. A least recently used pruning strategy is employed to keep the instance library to a manageable size; also only representative instances are stored in instances library. However, the least recently used pruning strategy causes this anomaly detection approach to be prone to a slow training attack (attackers slowly insert their behaviors into the normal profile to cause the intrusion detection system to identify their behaviors as
Lane and Brodley’s IBL algorithm is different with computer immunology approach in that it defines a similarity measure instead of doing the exact matching. Lane and Brodley also proposed: in addition to exact matching and similarity measure, any off-the-shelf machine learning algorithm can be applied to anomaly detection based characteristic sequences. Followed Lane and Brodley, Lee et al. experimented on using a detection tree software RIPPER to classify system call sequences [35]; Ghosh proposed to use Neural Network to classify program BSM event stream [21]; Mukkamala used SVM to classify network and host events [66]. In this dissertation, we proposed to use Support Vector Machine (SVM) to classify system call sequences, which has shown to perform very well.

2.2.3.2 JAM

Lee et al. applied a decision tree learning approach RIPPER to learn normal and abnormal patterns of program behaviors in JAM project [35]. Similar to computer immunology approach, Lee’s method is also based on system call sequences. However, instead of rote learning by maintaining a normal system call sequence database like computer immunology method, Lee’s approach employed concise rules to generalize the system call sequence information, and is able to identify unseen intrusions and unseen normal traces.

This generalization ability is what differs machine learning method from other anomaly detection methods. Other methods such as computer immunology method can memorize past behaviors but is unable to generalize from past behaviors. Thus, if the observed normal behaviors are not adequate and can not cover all users' or programs' behaviors, future unseen normal behaviors will be classified as anomalous and false alarm rates will be high. By applying a machine learning approach like RIPPER, knowledge about abnormal and normal patterns of program behaviors are generalized and a concise set of rules for normal and abnormal behaviors are produced.

Compared to computer immunology method, this decision tree based approach also has the advantage of being less sensitive to noise, thus less false alarm rates. A post processing procedure is used after prediction using RIPPER on the input system call sequences.
region of length $2l + 1$, if more than $l$ predictions are anomalous, the region is anomalous. The instinct behind this idea is: “When an intrusion actually occurs, the majority of adjacent system call sequences are abnormal; whereas the prediction errors tend to be isolated and sparse” [35]. Thereafter, this procedure filters out some spurious prediction errors.

Different from the user based method, such as IBL, this method is program based, so it is insusceptible to slow training by the attackers. However, this method still has not overcome the shortcoming of common anomaly detectors - being impacted by training data. RIPPER outputs rules for the minority class and the default class for unmatched instances is the majority class. Thus, if abnormal data is the major instances in the training data, the default class will be abnormal, which means this approach can not classify most unseen normal behaviors correctly; otherwise, the default class in the RIPPER rules will then be normal, which means this approach can not classify most unseen intrusive behaviors correctly.

2.2.3.3 Neural network

Ghosh et al. proposed to use neural network for detecting anomalies [21]. They compared two kinds of neural networks, feed-forward backpropagation neural network and recurrent neural network (Elman network), with simple equality matching algorithm, which is similar to computer immunology method. Their experiments showed neural networks had a better accuracy rate than equality matching algorithm.

Their goal of using neural network was to detect novel attacks, while at the same time, reduce false alarms. Ordinary anomaly detection methods based on exact matching or similarity measures have a high false alarm rate when the observed audit data is incomplete and the future data set includes much previously unseen normal data. Besides learning the associations between inputs and outputs, neural network can generalize from previously seen inputs to predict future unseen inputs. Neural network is a good fit to generalize these incomplete observed data and give a good prediction to future seen and unseen data.

Like decision tree method, which is more sensitive to temporal co-located events and less sensitive to noises, Ghosh et al. uses a leaky bucket on the result of the neural network. The leaky bucket memorizes recent events and clears out the old events, so it is sensitive to a
series of anomalies and neglects occasional anomalies. Since it is well recognized that program anomalies are always located in temporal clusters and isolated anomalies tend to be prediction errors, the leaky bucket approach helps to reduce prediction errors.

The feed-forward backpropagation neural network method uses weights of the network to store generalized mapping from inputs to outputs. The inputs to the neural network are sequences of BSM events. A set of randomly generated input data are first used as abnormal data to train the network. So by default, the neural network classifies data as anomalies, which complies with the anomaly detection concept - unknown data are classified as anomalies. Then observed sequences, which are sequences of BSM events captured during a program's execution, are used to train the neural network. To put into operation, the trained neural network classifies BSM sequences in order and uses a leaky bucket to make the final prediction.

The above feed-forward backpropagation neural network method can not catch "large scale structure" of programs. Elman networks overcome this disadvantage since it maintains some state information using context nodes which only take input from hidden nodes [21]. When training, the sequences of BSM events are input into an Elman network in order; suppose the nth output from the nth input $O_n$ is $I_n$, the difference between nth output $O_n$ and (n+1)th input $I_{n+1}$ is used as the anomaly measure of the sequence $I_{n+1}$ and fed into a leaky bucket. The Elman networks achieve better accuracy rate than the feed-forward backpropagation neural network method since it is also trained to recognize large scale structure of program behaviors [21].

The exact matching method classify all unseen inputs as anomaly, which often causes a high false alarm rate; while neural network method can classify novel inputs from the generalization of seen input thus give a reasonable prediction. Therefore, the neural network approach performs better in reducing false alarms compared to exact matching method.

2.2.3.4 Support Vector Machine

Support Vector Machine (SVM) is a statistical machine learning algorithm that has gained interests in many research areas in recent years [6]. SVM is based on the idea of structural risk minimization, which means finding a learning machine with minimum upper bound on
error on the actual data [6].

Mukkamala et al. in New Mexico Technology University has done research on using SVM for intrusion detection [66]. They use the original two class SVM with standard Radial Basis Function (RBF) kernel for intrusion detection.

They have constructed 31 features for the SVM detector. Among these features, 22 features are network based [66], such as:

**Duration**: Length of connections

**Prototype**: Network protocols, such as TCP, UDP

**Service**: Network services, such as HTTP, TELNET

**Srcbytes**: Number of data bytes from source to destination

**Dstbytes**: Number of data bytes from destination to source

**Flag**: Normal or error status of the connection

**Land**: 1 if connection is from/to the same host/port; 0 if otherwise

**WrongFragment**: Number of “wrong” fragments

and 9 features are host based, such as:

**NumRoot**: Number of “root” access

**SUattempted**: 1 if “su root” command attempted; otherwise 0

**NumFileCreations**: Number of file creation operations

**NumShells**: Number of shell prompts.

**NumAccessFiles**: Number of operations on access control files

**RootShell**: 1 if root shell is obtained; 0 otherwise
There features are constructed manually. Thus, their feature construction and feature selection methods are complicated and time consuming.

Their experiment results on the 1998 DARPA intrusion detection evaluation data are promising, nevertheless, it can be improved since their method only use standard SVMs, i.e., they use the original two class SVM and standard Radial Basis Function (RBF) kernel.

2.2.3.5 Pros and cons of using machine learning for anomaly intrusion detection

Machine learning methods can generalize from observed incomplete data. This generalization ability is what differs machine learning from other methods such as computer immunology approach, which only memorizes past behaviors but is unable to generalize from past behaviors. Using computer immunology approach, if the observed normal behaviors are not adequate and can not cover all users’ or programs’ behaviors, future unseen normal behaviors will be classified as anomalous and false alarm rates will be high. Whereas, using machine learning methods like decision trees, Neural Network and SVM, classifiers are produced to recognize general patterns of program behaviors. This will reduce the false alarm rate which is one of the critical problems in anomaly intrusion detection research.

Although machine learning methods can reduce false alarms by generalizing the observed behaviors, they can not handle the false alarms caused by significant and sudden environmental changes. In general, a classifier learned from a sufficiently large number of representative training examples is likely to accurately classify novel instances from the same space. Nevertheless, if the novel instances have different distribution with the training data because of environmental changes, the classifier learned from previous training data will have high false alarm rates for new audit data.

One of the contributions in our research work is to adapt SVM in the intrusion detection field. We propose two new kernels which are designed for anomaly intrusion detection on system call sequences. In our experiment, these two kernels, combined with SVM, are demonstrated to have high detection rate and low false alarm rate. We also propose to use one
class SVM, which does not need labeled training data, for unsupervised learning for anomaly detection.

### 2.3 Specification based intrusion detection system

Specification based intrusion detection approach uses specifications to define expected behaviors of programs and systems, and observed behaviors deviated from the specifications are identified as intrusions. Specification based intrusion detection also uses derivation from normality as the criteria to identify intrusions. In contrast with anomaly detection, which defines normality from observation of users’ previous normal behaviors, specification based intrusion detection systems define normality using manually designed specifications that describe intended behaviors of programs and systems.

The specification based intrusion detection approach is proposed as an alternative of misuse and anomaly detection systems to detect new intrusions with reduced false alarms. Misuse detection systems can detect known intrusions with zero false alarms assuming the description of intrusions is precise. However, misuse detection systems cannot detect novel intrusions. Anomaly detection systems can detect novel intrusions, whereas their false alarm rate problem is conspicuous. In specification based intrusion detection approaches, no preceding knowledge about intrusions are needed, and intrusions are detected by inspecting the deviation from specifications. Thus, it can detect both known intrusions and novel intrusions.

Furthermore, since the specifications are designed to describe intended or legitimate behaviors of programs or systems, theoretically it will not be impacted by environmental changes, i.e., the legitimate behaviors of a program are the same after users change their behavior patterns. In other words, the illegitimate behaviors deviated from the specifications are always illegitimate assuming the specifications are correctly designed.

There are several research efforts on specification based intrusion detection. Some research work using Finite State Machine (FSM) to specify program's behaviors and detect intrusions [60] are introduced. In this dissertation we propose to integrate FSM with SVM based anomaly detectors.
A specification based intrusion detection system proposed by Uppuluri et al., are introduced [68]. This specification based intrusion detection system uses an event based security relevant behavioral model for intrusion detection and a Behavior Monitoring Specification Language (BMSL) for specifying event based security relevant properties.

Calvin Ko et al.'s work on security relevant specifications are another well known research on specification based intrusion detection system. Calvin Ko et al. employed security relevant specifications for UNIX privileged programs, such as fingerd, sendmail, and rdist, and applied an execution monitor to analyze audit data regarding the specifications [32]. A program policy specification language based predicate logic is used to describe the security relevant specifications in their approach.

2.3.1 FSM based intrusion detection

Sekar et al. proposed to use an automaton to model program behaviors [60]. Automaton can catch the branches and loops of a program's behavior, thus it is expected to generalize a program's behavior quite well. It is intuitive to think of generating a Finite State Machine from program sources, however, program sources are not always available. Thus, Sekar et al. presented a method based on program execution traces to automatically and efficiently characterize program behaviors.

One problem of developing Finite State Machine (FSM) from program execution traces is to obtain state information. System call sequences do not provide state information, so program structures can not be obtained directly from system call sequences. Therefore, Program Counters (PCs) are used to indicate state information in their method. Sequences of pairs of system call and PCs are traced and used for building a FSM, in which a state is indicated by a PC and a transition is associated with a system call. To deal with library functions, information is recorded from the locations where the library functions are called.

The FSM built in this way can recognize branches and loops, thus it can model temporal relationships in program execution traces, either long term or short term. System call sequence method can only model short term relationships, and it can not recognize different traces generated from the same loop if the window size is not very large. Thus, FSM based approach
learns program structures from traces better than the system call sequence method. As a result, FSM based approach converges faster and has less number of false alarms than system call sequence method.

Furthermore, FSM based approach has comparable overhead time. For overhead of time, it depends on the learning process which builds the FSM from source code or execution traces and interception process which intercepts the system calls. As mentioned earlier, the learning process of the FSM based approach takes less time to converge. The interception of system calls are similar for FSM based approach and system call sequence method and depends on the interception method. If using a user-level program like `strace`, overhead time is very high because of the switch between user space and system space; while if using kernel-level interception, less overhead time is introduced because no switch is needed.

FSM based approach also has less space overhead because of FSM's compact representation. FSM can model branches and loops compactly, which contributes to less space overhead for FSM based approach. FSMs actually memorize and generalize arbitrary length sequences compared to approach using system call sequence [60].

FSM based approach is very powerful in detecting the attacks that will change program behavior, such as buffer overflow attack and Trojan Horse attack. Dictionary password brute force attack and DoS attack can be detected by augmenting the transition with frequency of execution of some system calls. Attacks by manipulating system call arguments can be detected by including system call arguments in FSM's transitions. Similar with other program based learning approach, FSM based approach has limitation in detecting attacks that do not change program behaviors, such as race condition attack, while steals some system resources from a privileged program by taking advantage of the time lag between checking the availability of a resource and actually using the resource of a privileged program.

2.3.2 Uppuluri's specification based intrusion detection/response system

Uppuluri et al. proposed an event based security relevant behavioral model for intrusion detection. Knowing that intrusions can be identified by events that deviate from normal, using specifications of normal events that can occur in a system or during a program's execution,
intrusions can be found [68].

A Behavioral Monitoring Specification Language (BMSL) was developed for specifying event based security relevant properties. Similar to the program policy specification language proposed by Ko, BMSL also uses regular expressions to catch security properties of a program; however, the difference is BMSL uses rules to define the response to a security event (the response can be a defensive response or simply set state variables) while Ko's method does not include response rules.

Uppuluri et al. observed that the efforts of developing specifications are normally less than the efforts for obtaining training data and updating the anomaly detector for environmental changes. What makes the specification approach more promising is that specifications are nearly site and installation independent and can be ported to other operating systems easily. Compared to anomaly detection, which needs to be trained on different operating systems, sites and installations, specification based intrusion detection systems are easier to use.

2.3.3 Ko's specification based approach

Calvin Ko et al. proposed to use specifications of security relevant behaviors of privileged programs to detect intrusions on systems [32]. Observing how privileged programs are always exploited by attackers for attaining the privileges on system, Calvin Ko et al. proposed to specify the intended behaviors of privileged programs and label the observed behaviors deviated from the specification as intrusions.

Security specification is a security relevant part of the intended behavior of a program. Although the behavior of privileged program can be very complicated, such as sendmail, the security related part of the program behavior is usually quite simple. For example, for the sendmail program, the security related part is identified as what files sendmail reads, what files sendmail writes and what are the programs being executed from the sendmail program. Thus, despite the complexity of the privileged program, a concise security specification can usually be defined to describe the intended behavior.

From this perspective, security specification is defined based on the knowledge of security vulnerabilities of privileged program, and it seems similar with intrusion signatures in misuse
intrusion detection. However, specification based intrusion detection is different from misuse intrusion detection in that a misuse rule defines the precise pattern that indicate intrusions and only known intrusions can be identified; whereas security specification defines the security policy of a program, thus novel intrusions, which also violate these security policies, can be detected.

Take the sendmail program for example. The security specification can be described as: the sendmail program can read worldly readable files, write to files in mail spooling directory and mail queue directory, write to configuration file of sendmail, bind to sendmail port and execute a predefined program for a user when new mails arrive. Any other manipulation of files and execution of other programs will be identified as intrusions.

Three types of security properties are categorized to be described in security specification for program [32]:

1. Names of objects that a program can access, such as “/etc/passwd”.
2. Conditions that impact the accessibility of an object to a program, such as “if a program creates an object, then this program can access this object”.
3. Abstract states of the program execution that impact what a program can do. They are represented by values bound to variables. For instance, “if a program P creates a temporary file X, the temporary file X can then be read by the program P”.

A program policy specification language is developed for specifying the policy of a program. It employs predicate logics and regular expressions to describe the set of object names, conditions and states allowed for each operation.

A formal framework is designed to translate the security specification to audit trail rules, preprocess the audit trails and associate them with subject identifier such as user and program, and apply audit trail rules to these preprocessed audit trails to find out intrusions.

2.3.4 Discussion of specification based intrusion detection

The major difference between specification based intrusion detection and misuse intrusion detection is: specification based intrusion detection utilizes knowledge pertaining to program
or system behaviors; misuse intrusion detection utilizes knowledge pertaining to attack behaviors.

The main difference between specification based intrusion detection and anomaly intrusion detection is: specification based intrusion detection derives normality from expected or intended behavior of a system or program (as the form of specifications); anomaly intrusion detection derives normality from the observed behavior (as the form of normal profiles).

Specification based intrusion detection has the following advantages:

- Low false alarm rate. Since normal programs or system behavior comply to their specifications, the false alarm is near zero.

- No need for training. Considering the troublesome of training anomaly detectors or adding all the misuse rules regularly, specification based intrusion detection systems are easy to maintain and use.

- It is easy to port to other sites or operating systems. Specifications are usually common for the same kinds of operating systems and programs. Also, they are less environmentally relevant (less related to sites, installations and usage) than profiles in anomaly detection and rules in misuse intrusion detection. Usually, it is not necessary to build new specification when the environmental changes, and it takes less effort to make it work on another site.

Despite these advantages, the specification based approach has its own weaknesses. Detailed specifications require tremendous efforts. There are trade-off between the preciseness of specifications and false positives (undetected attacks). How much effort should be spent on defining precise specification to get satisfactory detection rate is a difficult question for designers. In Uppuluri’s event based security relevant behavior model, this problem is dealt with by developing general specifications first and refining them to a satisfactory level.

Since specification based method does not incur false alarms if specifications are correctly defined, we say that specification describes a superset of normal behaviors. Therefore, how to increase the detection rate of specification based intrusion detection is an interesting research
topic. In the event based security relevant behavioral model, specification is augmented with misuse constraints and detection rate can be improved considerably. However, knowledge about intrusion behavior must be known to integrate misuse intrusion detection.

2.4 Hybrid intrusion detection system

The three types of intrusion detection have different strengths and weaknesses. Considerable research effort has been spent to integrate different kinds of intrusion detection approaches to combine their strengths and alleviate their weakness.

2.4.1 Integration of misuse and anomaly intrusion detection

The integration of misuse and anomaly intrusion detection has been studied since early intrusion detection research. Misuse intrusion detection can detect attacks precisely with a low false alarm rate, however, it can not detect novel attacks; on the other hand, anomaly intrusion detection can detect novel attacks, however, it has a higher false alarm rate. Thus, integrating these two methods to combine their strengths and alleviate their weaknesses may generate better results.

2.4.1.1 NIDES

NIDES has a rule based intrusion detection component and a statistical anomaly detection component [3]. NIDES's rule based intrusion detection uses P-BEST expert system for analyzing audit trails and detecting known intrusions.

NIDES's profiling engine is used to profile users' behaviors and several applications. A statistical score is calculated for each session of a user to measure how this session deviates from the established normal user behavioral model. For an application, statistical measures are customized to measure the proper operation of the application and a statistical score is calculated for each operation of the application.

NIDES's rule-based expert system and statistical anomaly detection system operate and report intrusions separately.
2.4.1.2 EMERALD

EMERALD was developed from NIDES. Different with NIDES, EMERALD employs a highly distributed analysis strategy. In EMERALD, distributed small signature analysis engines and distributed profiler engines are deployed to monitored machines.

The combination of signature analysis and anomaly detection is achieved using a universal resolver, which can invoke real time countermeasures in response to intrusive behaviors. The response criteria is decided by evaluating the results of signature analysis and the anomaly detection engine. The metric representing the deviation of normality and the metric representing the severity of possible attacks are combined together to formulate the resolver’s countermeasure responses, such as close a connection or shut down a program.

2.4.2 Integration of specification and misuse detection

Specification based intrusion detection can detect attack behavior that are deviated from system or program specification. However, there are some attacks that can not be caught using system or program specification. For example, a race condition attack can not be caught based on a program’s specification because system resources stealing in race conditions attack will not change the behaviors of the privileged program monitored. Thus, misuse detection is viewed as a promising method which can compensate for the specification based approach and catch the missed intrusive behaviors based on intrusion signatures.

BMSL, a program specification description language proposed by Uppuluri [68], not only can define normal specifications but can also describe misuse signatures. By augmenting specification based methods with misuse constraints, their experimental results using DARPA evaluation data improve greatly [68].

Specifications can also be used to guide anomaly detection and assist in lowering the false alarm rate of anomaly detection. Integrating specification based approach with anomaly detection approach to combine their strength and alleviate their weakness is one goal of this dissertation.
2.5 Intrusion detection system infrastructure

Generally speaking, most intrusion detection systems follow a hierarchical architecture, which allows scalability and efficient communication. In the hierarchical architecture, the bottom layer is for rendering data in a common format to upper layers; the second layer is for data analysis and reduction; the top layer is usually the user interface.

In order to detect these distributed intrusions, much research focuses on building distributed intrusion detection systems which can detect coordinated intrusions from several sites. Large scale coordinated attacks, such as Distributed Denial of Service (DDoS) attacks, have increased dramatically. Hackers can obtain the tools that can easily launch these sophisticated attacks from the Internet.

Software agents are very suitable for distributed intrusion detection. Software agents can perform tasks autonomously, move between machines, share information with each other and interact with each other to accomplish tasks. Because of their mobility, they can reduce network latency, distribute the load and adapt to environment dynamically.

2.5.1 CIDF

Common Intrusion Detection Framework (CIDF) aims at enabling different intrusion detection and response components to interoperate and share information [55]. The CIDF is a standard proposed by the Information Technology Office of the Defense Advanced Research Projects Agency, University of California-Davis, Information Sciences Institute, Odyssey Research, and others. CIDF views intrusion detection systems as consisting of discrete components that communicate via message passing. CIDF consists of four kinds of intrusion detection system components: Event Generators, Event Analyzers, Event Databases and Response Units.

A specification language, Common Intrusion Specification Language (CISL), is also proposed to allow independently developed intrusion detection and response systems to share information. Using CISL, IDS can disseminate event records, analysis results, and countermeasure directives amongst intrusion detection and response systems.
CIDF is an important step towards getting different intrusion detection systems to inter-operate with each other. Since intrusions are taking on a grander scale in number and type, many attacks can be orchestrated over a wide area network, and over a long period of time. It is very important for IDSs, despite being distributed on many locations and developed by different vendors, to be able to share information, infer possible distributed and coordinated intrusions, and warn others about impending attacks. CIDF and CISL are designed to satisfy this need.

2.5.2 DIDS

Distributed Intrusion Detection System was developed at the University of California-Davis [62]. This system focuses on extending the intrusion detection from a single segment of a network to arbitrarily large networks. The architecture for the system includes a host manager in each host, a LAN manager for monitoring each LAN in the system. The central manager resides at a central location on a secure host, receives event reports from hosts and LAN managers, processes these reports, correlates events and generates intrusion alarms.

In DIDS, the monitoring and analysis tasks are distributed among the hosts and LAN managers. The hosts and LAN managers are responsible for detecting intrusions on one host or one segment of the network and for matching intrusion signatures. The central manager gets distributed audit data and correlates events, so it monitors the system from a coherent point of view and detects intrusions involving multiple hosts. It can also track tagged objects, for example, users or files as they move around the network.

2.5.3 EMERALD

EMERALD, developed from NIDES, employs a highly distributed analysis strategy, and is composed of scalable signature engines, scalable profiling engines and a universal resolver [52].

Distributed small signature analysis engines are deployed to a monitored machine. They scan event streams for possible attacks and report to upper layer. Upper layer signature engines scan these aggregated reports and try to detect more global coordinated attacks.
EMERALD uses a distributed profiler engine for each specific event stream. The anomaly reports from the low level propagate to higher levels and are correlated and merged for an enterprise wide anomaly report.

The universal resolver combines the results from the signature analysis engine and profiler engine and decides if the monitored machines are compromised. The universal resolver can invoke real time countermeasures in response to intrusive behaviors.

2.5.4 AAFID / AAFID 2

The Autonomous Agents for Intrusion Detection (AAFID) project is developed in Purdue’s CERIAS group [63]. AAFID uses a distributed architecture that utilizes agents to detect anomalous or malicious behavior. The first prototype of AAFID was implemented by a combination of programs written in C, Bourne shell, AWK and Perl. Its main objective was to test the initial feasibility of the architecture. The second implementation, AAFID 2, is completely implemented by Perl 5. It is composed of autonomous agents, transceivers, monitors and a user interface. Autonomous agents are used to monitor interesting events on hosts. Transceivers are used to control agents and communicate. Monitors can receive information from several transceivers and correlate events.

2.5.5 JAM

The Java Agents for Meta-Learning (JAM) project at Columbia University uses a secured agent infrastructure for continuous learning of fraud and intrusion patterns [35]. This system uses two kinds of agents: local fraud detection agents, which learn how to detect fraud and provide intrusion detection services with a single corporate information system, and meta-learning agents, which correlate the collective knowledge acquired by individual local fraud detection agents.

2.5.6 DSRIDA

The DoS Resistant Intrusion Detection Architecture (DSRIDA) is a project at the Computer Security Resource Center, National Institute of Standards and Technology (NIST) [43].
Intrusion detection systems easily become attack targets on the Internet. They can disable the IDS first, and then carry out their other activities without being found.

The DoS attack is an attack that is easy to launch and difficult to stop. Attackers can obtain the tools for DoS attacks and easily disable an IDS. The DoS Resistant Intrusion Detection Architecture is an important step in designing an IDS that can resist DoS attacks. In the DSRIDA project, the IDS is protected by hiding IDS components and moving them away from harm using mobile agent technology.

Our hybrid Mobile Agent Intrusion Detection System (MAIDS) use data gathering agents which correspond to Event Generators in CIDF, low-level agents which correspond to Event Analyzers in CIDF, and high-level agents which correspond to the Decision and Response Unit in CIDF. We also use a database backend to store events, which corresponds to the Event Database in CIDF.

Our hybrid Mobile Agent Intrusion Detection System (MAIDS) includes both misuse and anomaly intrusion detection agents. The misuse intrusion agents are automatically generated from Software Fault Tree modeling intrusions, using Colored Petri Net as design template for the IDS. The SVM anomaly intrusion detection agents in our system are automatically built and launched based on training data.

2.6 Summary

Related prior work performed on three kinds of intrusion detection systems (misuse intrusion detection systems, anomaly intrusion detection systems and specification based intrusion detection systems) were examined. An overview of hybrid intrusion detection systems which combine several or all the intrusion detection approach were also given. Intrusion detection system architectures, especially for distributed intrusion detection, were briefly reviewed.

In the next chapter, the related technologies which are being used in our research will be presented, such as Support Vector Machine, Kernel method, Software Fault Tree, Colored Petri Net, and specification based intrusion detection system.
CHAPTER 3. Related technology

This chapter introduces the technology related to this dissertation. We first introduce Support Vector Machine (SVM) and kernel methods, which are the machine learning methods we explore for anomaly intrusion detection. Then, formal methods, Software Fault Tree, Colored Petri Net and the conversion between them, which we use for misuse intrusion detection, are introduced in detail. We use Uppuluri's specification based intrusion detection system to guide the anomaly detection, and their methods are also explained in this chapter.

3.1 Technology related to machine learning methods

3.1.1 Support Vector Machine

SVM was first introduced by Vapnik and has been used with great success for pattern recognition fields such as handwriting processing [4] and protein homology detection [37]. SVMs have viewed the classification problem as a quadratic optimization problem. It has exhibited excellent accuracy on test sets in practice and have strong theoretical foundation in statistical learning theory [70]. In this section, key concepts of SVM, such as margin and generalization capability, will be provided and explained. Then SVM is introduced and its advantages are analyzed.

**Margin** The *margin* of a linear discriminant \((w, b)\) w.r.t. a labeled pattern \((x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}\) is defined as

\[
\gamma_i = y_i(<w, x_i> + b)
\]

If the margin is negative, then the pattern is incorrectly classified; if it is positive then the classifier predicts the correct label.
**Geometric margin** The larger the margin, the further away $x_i$ is from the discriminant. This is made more precise in the notion of the geometric margin

$$\gamma_i = \frac{\gamma_i}{||w||}$$

which measures the Euclidean distance of a point from the decision boundary.

**Functional margin** The functional margin of $(w, b)$ w.r.t. the data set $S = (x_i, y_i)$ is defined as:

$$\gamma = \min_i \gamma_i$$

**Margin of a training set** The margin of a training set $S$ is the maximum functional margin over all hyperplanes.

**Maximal margin hyperplane** A hyperplane realizing this maximum is a maximal margin hyperplane.

**training error** Given distribution $D$ over input space $X$, assume the training and testing data points are drawn randomly from $D$; the training error of a hypothesis $h$ is the fraction of points in the training set being misclassified by the hypothesis $h$.

**testing error (true error)** The testing error of $h$ is the probability that $h$ will make an error on an unseen test example randomly selected from $D$. Obviously, the testing error is of paramount interest, and it is referred to true error of learning machine.

**Generalization capability** Generalization capability is the “Capacity” of the machine - ability to learn any training set without error.

SVM method originates from the perceptron algorithm, which works by adding misclassified positive or subtracting misclassified negative examples to an arbitrary initial weight vector. In perceptron based learning, the learning problem is ill posed: finding one hyperplane that separates the data - many such hyperplanes exist. Certain principles to choose the best possible hyperplane are needed. The generalization capability of a machine is used as the principle for best hyperplane for Support Vector Machine.
The bounds on the classification error suggest the possibility of improving the generalization capacity by maximizing the margin, which can be reduced to minimizing $||w||$. So the learning problem converts to the following optimization problem:

$$\text{Minimize } <w, w> \text{ subject to } y_i(<w, x_i> + b) >= 1$$

Therefore, the SVM algorithm tries to find the best machine for a data set in the sense that it tries to maximize the correctness of the machine regarding the training data set and it also tries to maximize the generalization capability of the machine so it is also good for other unseen data sets. A mathematical optimization method is used to solve the optimization problem and find the best machine.

SVM controls capacity by increasing the margin, not by reducing the number of dimensions (features). The ability to learn is independent of the dimensionality of the feature space. Thus, the accuracy of SVM is not impacted by huge feature spaces, and a feature selection procedure which is used to reduce overfitting is not required for SVM, in the sense of increasing accuracy, although feature selection is critical for the accuracy of traditional machine learning algorithms. In this dissertation, we conduct research on using feature selection for the time and space efficiency of SVM anomaly detection.

3.1.2 Kernel function

In some optimization functions, e.g., dual presentation, the data points appear only as dot products. Thus, a kernel function is defined to return the value of the dot product between the two vectors in high dimension feature space: $K(x1, x2) = <\Phi(x1) \cdot \Phi(x2)>$ where $\Phi(x)$ is a function mapping from original input data space to another high dimensional feature space. Many machine learning algorithms, such as SVM [6], Fisher Discriminant (FD) [44] and Principal Component Analysis (PCA) [45], use kernel functions. There are also many existing algorithms being ‘kernelized’ [56].

Recently, there has been considerable interests in the development of kernels for applications including natural language processing [39], speech recognition [47] and computational biology [37]. Most research effort in the area of application based kernels has focused on string kernels. A family of string kernels for protein classification are introduced, such as spectrum
and mismatch kernels, restricted gappy kernels, substitution kernels and wildcard kernels [71]. In each case, the kernel is defined via an explicit mapping from the space of all finite sequences of an alphabet $Z$ to a vector space indexed by the set of k-length subsequences from $Z$. In the case of a wildcard kernel, $Z$ is augmented by a wildcard character.

Recent research by Cortes et al., also introduced rational kernels, which are a general family of kernels based on weighted transducers or rational relations, that can be used for analysis of variable-length sequences or more generally weighted automata, in application such as spoken-dialog applications [12]. There are two benefits of rational kernels: variable-length sequences can be handled and rational kernels can be computed efficiently using a general algorithm of composition of weighted transducers and a general single-source shortest-distance algorithm. String kernels are proved to be a special case of rational kernels.

For kernel based learning algorithm such as SVM, only dot products of input feature vectors need to be calculated, and the high dimensional feature vectors need not to be calculated, which leads to one of the advantages of kernel method - computational efficiency.

### 3.2 Technology related to specification based methods

Uppuluri proposed a specification based intrusion detection system, which use Behavior Modeling Specification Language (BMSL) to describe specifications and Extended Finite State Automata (EFSA) as a model of Detection Engine [68].

Specifications in BSML consist of rules of this format: $\text{pat} \rightarrow \text{act}$ where $\text{pat}$ is a pattern of event sequence and $\text{act}$ is an action to be launched when the pattern is observed. The patterns in BSML rules are specified using Regular Expression for Events (REEs), which extend Regular Expression with variables. REEs can model system call sequences that are characterized with system call names as well as system call arguments. The actions in BSML rules can be a variable assignment or an external function execution.

The presence of variables in REEs makes them more expressive than traditional Regular Expressions. The simplest REE patterns capture single-event occurrences; the primitive event patterns can be combined to capture properties of event sequences using temporal operators,
which include sequencing (an ordered happening), repetitive (happening multiple times) and alternation (either one happening). A condition operator is also included to specify constraints of a pattern.

Following is an example of BSML rules:

\[
\text{admFiles} = \text{"/etc/utmp", "/etc/passwd"} \\
\text{open}(f, \text{mode}) | (\text{realpath}(f) \notin \text{admFiles} | \text{mode} \neq \text{O.RDONLY}) \rightarrow \text{term()}
\]

This rules represents: If a system call open open a file that is an administration file or is read-only, then call term() function (| is the condition operator).

Extended Finite State Automata (EFSA) is proposed as the runtime model for pattern matching expressed in REEs. EFSA extends traditional Finite State Automata with a finite set of variables for representing system state.

The specification based intrusion detection system proposed by Uppuluri uses BSML to describe specifications for programs such as ftpd. A compiler is used to translate the specifications expressed in BSML to EFSA, convert EFSA to Nondeterministic EFSA (NEFAS) and translate the NEFAS to C++ classes used as Detection Engines. The C++ code output by the compiler is linked with a runtime support system that delivers the events by invoking the corresponding member function on the NEFA class.

in [68], a refining process of generating specification is also proposed:

1. Develop generic specifications for any programs. By grouping system calls with similar functionality and identifying basic security requirement for them (such as system call arguments), generic specifications can be developed and used in different operating systems and different sites after some specific information is defined (site and operating system related information such as location of a file).

2. Strengthen the generic specifications for a class of programs. Similar programs have similar security requirements. For examples, setuid programs are one class of programs with similar security properties.

3. Reinforce the specifications for applications or servers. Some servers and applications, such as an FTP server, may have a special security policy, so their specifications need
Figure 3.1 A sample of generic specification [68] to be refined to accurately catch security events.

4. Customize the specific specifications for a site or a host. Some site or host based security policy can be employed and specified in this step.

5. Add misuse constraints. Although only a small portion, some attacks can not be caught using a specification based approach. BMSL is also developed to catch intrusive events by adding misuse constraints to employ misuse intrusion detection.

As shown above, this procedure of generating specifications is from general to specific. More precise specifications can be built with higher detection rates and lower false alarm rates with more effort spent on developing these specifications. Nevertheless, the experiments showed that the general specifications produced in first step can catch the majority of attacks.

Figure 3.1 is a sample of generic specifications. It defines allowed file access operations, process operations, network calls, privileged calls and setting or changing resource attribute operations.

Based on document of FTP process, a specification for FTP process is also designed. Figure 3.2 is their simplified version of specification for FTP process.

Specification based approaches have the advantages of low false alarm rates. However, in order for high detection rates, big efforts are needed to design detailed and precise specifications. Misuse constraints, representing attack signatures, are augmented with the specification based approach to improve detection accuracy. For example, following misuse constraint is added to the ftp specification for detecting a guess ftp attack.

\[(\text{any})*; \text{execve}(\text{prog}, \text{arg}, \text{env})|\text{guessftp}(\text{prog:get()}) \rightarrow \text{term}();\]
1. `(setreuid)*; setreuid(r, e) -> loggedUser := e
2. `(getpeername)*; getpeername.exit(fd, sa, 1) ->
   clientIP := getIPAddress(sa)
   /* Access limited to certain system calls before user login. */
3. `(setreuid())*; ftpInitBadCall() -> term()
   /* Access limited to certain other set of system calls after user login is completed. */
4. `setreuid(); any() *; ftpAccessBadCall() -> term()
   /* UserSID must be set to that of the logged in user before exec. */
5. `(setuid(loggedUser))*; execve -> term()
   /* Resetting userSID to 0 is permitted only for executing a small subset of system calls. */
6. `setreuid(r, 0); ftpPrivilegedCalls*; (setreuid(r, loggedUser)
   || setuid(loggedUser) || ftpPrivCalls) -> term()
   /* A file opened with root privilege is explicitly closed, or has close-on-exec flag set. */
7. `open.exit(f, fl, md, fd) | geteuid()==0); (!close(fd))*;
   (execve | closeOnExec(fd)) -> term()
8. `connect(s, sa) | ((getIPAddress(sa) == clientIP)
   && (getPort(sa) != ftpAccessedSvcs)) -> term()

Figure 3.2 A simplified specification for FTP process [68]

Uppuluri et al. first systematically implemented a specification based intrusion detection system and experimented on the DARPA 98, 99 evaluation data. Their approach can detect 80% of intrusions with zero false alarm before augmented with misuse constraints, and can detect 100% of intrusions and zero false alarm after augmented with misuse constraints.

3.3 Technology related to formal methods

In this dissertation, formal methods and formal tools are used in the development of the misuse IDS to reduce design and implementation errors. Software Fault Trees (SFTs) are used to model intrusion and Colored Petri Nets (CPNs) are used to model intrusion detection. The relationship between SFTs and CPNs are explored to design a translator, which can take in a SFT description of intrusions and translate to CPN design template for intrusion detection automatically. This section introduces SFTs, CPNs and the previous research work that has been done to convert Fault Trees (FTs) to Petri Nets (PNs).

3.3.1 Software Fault Tree (SFT) for modeling intrusion

SFT is a top-down approach to the identification of process hazards [38]. It is one of the best methods for systematically identifying and graphically displaying the many ways a
system can go wrong. The root of a SFT always represents a hazard. By describing the ways by which the system can reach the unsafe state, SFT can help requirement specification of a system.

SFT for IDS design begins with an intrusion as the root node and traces back all the possible parallel and serial combinations of events that cause such an intrusion. SFT description of intrusions results in a number of benefits. First, SFT enables structured and systematic analysis of intrusions. Second, by describing the ways by which the system can reach the intrusion state, SFT analysis can help the requirement specification for an IDS. At last, SFT helps to identify appropriate countermeasures for an intrusion. The efforts to formalize the use of SFT to model intrusion and assist the development of an IDS are described in other documents [23].

3.3.1.1 Extended SFT for intrusion

To model intrusions precisely, we extend basic SFTs. A basic SFT is static and they cannot represent time orders of events. However, the time relationships among events are critical in the intrusion detection domain.

The effect of adding time constraint nodes may be demonstrated by considering the set $E$ of all combinations of events that make the root node of a basic (unextended) SFT ‘true’. The set $I \subset E$ represent the combinations of events which indicate intrusions and also make the root node of the basic SFT ‘true’. The set $E - I$ represent the combinations of events that make the basic SFT root node ‘true’ and do not indicate intrusions. The constraint nodes added to an extended SFT intend to exclude $E - I$, which will cause ‘false alarms’ in an IDS if they are considered as intrusions by an IDS.

Three kinds of time constraint nodes: ‘Occurs After’, ‘Immediately After’ and ‘Within Time’ are introduced based on interval temporal logic [2]:

- ‘Occurs After’ is the condition where one event is required to start after another event has started.
• 'Immediately After' is the condition where an event must occur after another event in the same context with no intervening events.

• 'WithinTime' is the condition where an event is required to follow another event within some amount of time.

Moreover, basic SFTs lack formal and rigorous semantics. Natural language annotations are used to label and describe events and constraints in a SFT. The semantic meanings of the natural language annotations labelling a node could be ambiguous. To ensure the unambiguous meanings of SFT for the automatic translation, we introduce expressions to label basic event nodes so that the exact meanings of the basic event nodes could be clear.

3.3.2 Colored Petri Net (CPN) for modeling intrusion detection

CPN is a well-documented and frequently used abstraction for modeling complex and distributed systems. It has been applied to a variety of problem domains, including security, network protocols, mutual exclusion algorithms, VLSI chip design and chemical manufacturing systems [28]. CPN combines states and actions into a single diagram through the use of colored tokens, places and transitions. Tokens carry data and the colors of tokens represent the different data types. For a more detailed and strict specification of CPN, please refer to Jensen's book [28].

CPNs are very well suited for designing IDSs because they can describe clearly the complicated interaction, classification and correlation of activities of IDSs. Furthermore, the CPN design of an IDS provides an efficient way to verify the design of the IDS using the CPN tools such as Design/CPN [13]. We can run attack simulations using this tool to verify the correctness of an IDS design. This tool also allow us to save and load the CPN diagrams from XML files. We can use the XML representation of the CPN diagrams in the conversion. Finally, CPNs may be organized in a hierarchical fashion and this allows modular design and reuse of the CPNs.
3.3.3 Conversion between SFT and CPN

Malhotra and Trivedi have proved that Generalized Stochastic Petri Nets (GSPNs) are more powerful than Fault Trees with Repeated Events (FTREs) in the sense that all the scenarios modeled by FTREs can be modeled by GSPNs [40]. They also describe an algorithm to convert a FTRE model into an equivalent GSPN model [41]. A FTRE is a static model and cannot represent sequential events. Hura and Atwood proposed using PNs to analyze coherent FTs, so the analytic properties of Petri Nets, such as, safeness, liveness, concurrency, boundness, can be used to analyze deadlock operations, reachability and controllability of a system [26]. Coherent FTs are FTs constructed using only AND and OR logic operations. Hura and Atwood have also established a relationship between the simple AND and OR logic operations in FTs and their equivalent operations in PNs.

3.3.4 Summary

In this chapter, we introduced related technologies for this dissertation. These techniques included SVM and kernel methods, which are the essential machine learning technologies we use in our research; Uppuluri's specification based intrusion detection system, which we use in guiding the SVM based anomaly detection; and formal methods, which we use for the automatic generation of misuse intrusion detection agents.

In the following chapters, our works on improving anomaly detection accuracy rate using kernel method and SVM will be detailed first. Then, an approach of combining the strength of specification based intrusion detection and anomaly intrusion detection will be presented. A system for automatically generating anomaly intrusion detection software agents will be introduced. Finally, a systematic method of automatically generating misuse intrusion detection software agents using SFT and CPN will be explored.
CHAPTER 4. Kernels for anomaly intrusion detection

Although misuse intrusion detection is the most common technique used in intrusion detection systems, especially for commercial products, anomaly intrusion detection is receiving more and more attention. Many intrusion detection systems are beginning to include an anomaly detection module since misuse intrusion detectors can not detect the new novel attacks emerging on a daily basis.

However, current anomaly detection methods have the problem of high false alarm rates, which creates a serious problem as too many false alarms will bury true alarms, resulting in administrators not distinguishing true alarms from false alarms, rendering the anomaly intrusion detection system useless.

4.1 Analysis of the false alarm problem in anomaly detection

The false alarm problem of anomaly detection is inherent. Anomaly detection defines normality based on the observed normal behaviors. Usually, not all normal behaviors are observed when generating a normality profile. Hence, unseen behaviors are identified as anomalies, which results in false alarms. To alleviate this problem, a new approach is necessary for designing an anomaly intrusion detector that can classify not only the observed normal behaviors but also unseen normal behaviors as normal. Generalization capability (as described in Chapter 3) of an approach is how much that technique recognize or characterize inputs it has never encountered before. It is critical for an anomaly detector to have good generalization capability, thus future unseen behaviors can be classified correctly based on currently available observed behaviors.

There are two kinds of unseen normal behaviors. One is the type of behaviors that are
similar to the observed normal behaviors. To classify these behaviors correctly, the observed behaviors need to be properly generalized to represent normal patterns. The other type is normal behavior that can not be predicted from the observed data. For example, normal behaviors caused by environment changes can not be detected. In anomaly detection, the only way to deal with this kind of normal behaviors is to rebuild the normality model regularly or when the environment changes, such as when new users are added.

Since reducing false alarms in anomaly detection mainly depends on proper generalization of normal patterns from observed (sometimes incomplete) data, much research aims to improve the generalization capability of anomaly detectors. There are three main issues to be considered in designing an anomaly detector with generalization capabilities:

1. What specific generalization method to use?

2. What features to observe?

3. What similarity measure to use for pattern matching?

To answer the first question, we can use the statistical model method, the immunology approach or machine learning method. The statistical model method specifies normal ranges of statistical measures based on their observed values. For instance, if the average number of files opened in a user's sessions is $N$, and the threshold value for deviation is $d$, then the normal range is $N - d$ to $N + d$, and any value that is out of this range is identified as an anomaly. The generalization using the statistical model is intuitive and very simple. However, there is no automatic method to decide the threshold value $d$ in the statistical model. The threshold value is critical for controlling the false alarm rate and the detection rate. As there is no known automatic approach to calculate the appropriate threshold value, so it is usually decided empirically.

The immunology approach proposed by Forrest et al. [17] simply compares new feature values to observed feature values, and it is considered abnormal if the new feature value does not match any observed feature values. This method employs exact matching of feature values, so it could not be considered as generalization at all.
Machine learning algorithms provide good generalization capabilities. Neural Networks and Decision Trees (e.g. RIPPER) are two methods that are widely used for anomaly detection. Neural Networks use a network to generalize previously seen inputs from the network and map future unseen inputs into normal or abnormal outputs. Decision Tree methods use concise rules to generalize the mapping from input feature values to outputs. The generalization capability of machine learning algorithms has been discussed in detail in Chapter 3.

To answer the second question, feature selection is required to select the most relevant features for anomaly detectors. This is very critical for improving the prediction accuracy since the generalization capability of a method also depends on feature selection. Good generalization capability can be achieved by proper feature reduction. For the statistical model, the features (variables) to use are decided using an ad hoc method by domain experts. For the immunology approach, all fixed size sequences that appeared in program execution trails are used as features, and there is no feature selection procedure. For machine learning methods, feature selection is considered very critical to achieve accuracy. Much research work has studied feature ranking and feature selection [25, 65].

To answer the third question, currently there are two kinds of similarity measures. One is value based, which is the difference between two values. For example, a value \( a \) is compared to value \( b \); the smaller their difference, the more similar they are. Most statistical models use value based similarity measures. Another one is string based. There are many similarity measures of this kind, and the most simple one is to count the difference in the corresponding positions of two same size strings. The immunology model uses string based measures and compares system call sequences (in string format of monitored privileged programs) to the system call sequences observed in their previous normal execution. An appropriate similarity measure will give better prediction for unseen data based on observed data.

The above three questions must be answered in designing a high performance anomaly intrusion detector. In the following sections, we analyze how SVM based anomaly detectors answer these three questions and be suitable for anomaly intrusion detection.
4.1.1 Generalization capability of a machine learning algorithm

The best generalization capability can be achieved when the true error is minimized while the accuracy of classifying training data set is maximized. The learning task is to search for a hypothesis \( h \) which can guarantee the best generalization capability. SVM is a new emerging machine learning algorithm which uses an optimization approach to search for a hypothesis which minimize true error with a bound on classification correctness on training data [6] (see Chapter 2 for detail). In order to achieve good generalization capability, SVM minimizes the bounds on the error of classification by maximizing the margin. The risk of overfitting, which means classifiers drawn from the training data behave considerably better on the training instances than the testing instances, can also be minimized by choosing the maximal margin hyperplane in the feature space. Therefore, we propose to use SVM for anomaly intrusion detection.

4.1.2 Feature selection

Feature selection is very important for anomaly intrusion detection. Because irrelevant features can create overfitting problem, which means the derived anomaly intrusion detector performs much better on training data than on testing data. Many researchers have proposed methods to rank features by relevance, and get rid of irrelevant features [25, 65]. SVM has a special characteristic that it achieves the best generalization capability without feature selection.

4.1.3 Similarity measure

Similarity measure is critical for pattern matching. In SVM, the similarity measure is represented by kernel functions, which is the dot product of two vectors. Thus, selection of suitable kernel plug-ins becomes critical for applying SVM on anomaly detection.

From the above analysis, the motivation for using SVM are summarized as follows:

- SVM provides very good generalization capability and alleviates the overfitting problem
in anomaly detection.

- The complex feature selection procedure is no longer needed.

- SVM uses kernels to represent the similarity measure and the kernels act like a plugin for the SVM algorithm. They can be replaced, changed and tuned, which makes the SVM method more flexible. This chapter proposed two kernels that are suitable for anomaly detection.

### 4.2 Design and computation of STIDE kernel

Any kernel based learning is composed of two modules: a general purpose learning machine module and a problem specific kernel module. It has been discovered that many machine learning algorithms that only use dot product of input data can be rewritten using a kernel module and a learning machine module.

The STIDE anomaly detector developed at the University of New Mexico is a well-known anomaly intrusion detector. We analyzed the STIDE anomaly detector, and modeled it as a kernel based learning algorithm which is composed of a threshold function and a problem specific kernel function. Then, we kept the kernel function, and used a SVM algorithm, which has theoretically and empirically excellent generalization capability, to substitute the threshold function used in the STIDE anomaly detector. The result is an SVM anomaly detector based on STIDE kernel. Our experiments demonstrated that it is better than the existing STIDE anomaly detector.

#### 4.2.1 STIDE anomaly detector

In anomaly detection based on system call sequences, an alphabet is used to represent system calls; a sequence of alphabet corresponds to a sequence of system calls, and sequences are collected using the trace utilities in UNIX. Normal sequences are obtained from normal executions of the privileged programs and abnormal sequences are generated from abnormal executions.
The STIDE anomaly detector uses a fixed-length sliding window of size \( k \) to generate all the subsequences with size \( k \) from the training data to form a normal \( k \)-size system call sequence database [17]. Given a sequence of testing data, anomalies are detected by sliding a window of size \( k \) along the testing data: if the number of subsequences of size \( k \) not existing in the normal database is greater than a certain predetermined anomaly threshold, \( \delta \), then the detector declares that an anomaly has occurred, and the current position in the testing data can be used to provide information about where the anomaly occurred.

4.2.2 STIDE kernel

Let alphabet set \( Z \) represent all the possible system calls that appear in the UNIX system program and \( k \) be the sliding window size used by the STIDE model. The input space \( Z^* \) is a space of sequences of characters from the alphabet \( Z \) (also known as traces). We define a \( k \)-STIDE kernel mapping from \( Z^* \) to a \( |Z|^k \)-dimensional feature space space. The feature map from \( Z^* \) to a \( |Z|^k \)-dimensional feature space is defined as follows:

\[
\Phi_k(s) = (e(s_k))_{s_k \in A}
\]

where \( s \in Z^* \); \( A \subseteq Z^k \); \( e(s_k) \) is 1 when \( s_k \) is a subsequence of \( s \) or 0 otherwise. Thus, \( e(s_k) \) is a vector of 0 and 1 with dimension \( |A| \), and \( A \) represents the feature set, which is a subset of all subsequences with size \( k \).

The \( k \)-STIDE kernel is then:

\[
K_k(s_1, s_2) = \Phi_k(s_1) \cdot \Phi_k(s_2)
\]

With respect to the kernel definition, the original STIDE anomaly detector [17] is expressed using this kernel and a threshold function as follows. In the training phase of STIDE, a vector \( v_0 \), which represents all the normal subsequences, is calculated:

\[
v_0 = (e(s_k))_{s_k \in A}
\]

where \( e(s_k) \) is 1 when \( s_k \) is a \( k \)-length subsequence in the training data set or 0 otherwise. \( \tilde{v}_0 \) (the complement of \( v_0 \)), which represents all the abnormal subsequences, is also calculated. In the testing phase, for each sequence \( s \) in testing data, a vector \( v \) is calculated as \( v = \Phi_k(s) \).

Then, the dot product of \( v \) and the vector \( \tilde{v}_0 \) is compared against an anomaly threshold \( \delta \). When the dot product is greater than or equal to the anomaly threshold, an anomaly alarm is raised, i.e., the linear threshold function for anomaly detection is: \( v \cdot \tilde{v}_0 \geq \delta, \text{anomaly}; v \cdot \tilde{v}_0 < \delta, \text{normal} \).
Using the STIDE anomaly detector, consider the following example:

alphabet={a, b, c}, window size k=2

Original feature space: \( Z^* \)

New feature space: \( Z^k = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\} \)

**Training phase:**

Training data of normal processes sequence \( Z^* \): abbbabb, bcccc

Thus, \( N=\{ab, ba, bb, bc, cc\} \) is the normal system call sequence database built from the training data.

According to the definition above, we get, \( \mathbf{v}_0 = (0, 1, 0, 1, 1, 0, 1, 0, 1) \) and \( \mathbf{v}_0 = (1, 0, 1, 0, 0, 1, 1, 0) \).

If we assume anomaly threshold to be 2 and \( \mathbf{v} = \Phi_k(s) \) where \( s \) is the testing data, then the linear threshold function should be: \( \mathbf{v} \cdot \mathbf{v}_0 \geq \delta \), anomaly; otherwise, normal.

**Testing phase:**

Is the sequence \( abcca \) normal?

The corresponding vector to this sequence is: \( \mathbf{v} = (0, 1, 0, 0, 0, 1, 1, 0, 1) \).

Using the linear threshold function we got in the training phase, \( \mathbf{v} \cdot \mathbf{v}_0 = 1 \); it is normal.

Is the sequence \( aacca \) normal?

The corresponding vector to this sequence is: \( \mathbf{v} = (1, 0, 1, 0, 0, 0, 1, 0, 1) \).

Using the linear threshold function we got in the training phase, \( \mathbf{v} \cdot \mathbf{v}_0 = 3 \); it is abnormal.

The STIDE anomaly detector compares every new instance with a standard abnormal vector \( \mathbf{v}_0 \), and when the number of matches is greater than a predetermined threshold, \( \delta \), an alarm is raised.

**4.2.3 Efficient computation of STIDE kernel**

Since the system call sequence length is \( k \) and the alphabet is \( Z \), there can be \(|Z|^k\) permutations of all possible system call sequences, which means, the vector length is \(|Z|^k\). To calculate the STIDE kernel, the whole vector needs to be calculated. The algorithm complexity is \( O(|Z|^k) \), which increases exponentially with a sliding window size \( k \) and is impractical.
to be calculated when $k$ becomes greater than 4 with alphabet size above 100.

To calculate a $k$-STIDE kernel efficiently, all the subsequences of size $k$ that do not exist in the log file of the privileged system program can be neglected. This saves tremendous space and computation time. Thus, the complexity of the $k$-STIDE kernel method is reduced to $O(|N|)$ where $N$ is the set of all the subsequences with size $k$ that exist in the traces of the privileged system program if it is already a priori knowledge. Otherwise, it is hard to find all the sequences without peeking into both the training and testing data. One way is to build the vector from the training data and ignore those new features found in the testing data. Another approach will be described in Chapter 6, which uses specification guided method to find all the relevant features of the privileged system program.

4.3 Design and computation of Markov Chain kernel

Markov Chain models have been used for anomaly detection for system call sequences of privileged system programs in UNIX [42]. We define a Markov Chain kernel in this section.

4.3.1 Markov Chain anomaly detector

The Markov-based detector described in [42] uses a fixed-size detector window of size $k$ to build a discrete-time Markov Chain as follows. Each state of the Markov Chain is a sequence with size $k$ appearing in the training data. The probability of a transition from state $s = (a_1, a_2, \ldots, a_k)$ to state $s' = (a_2, \ldots, a_k, a_{k+1})$ is computed as $p(s, s') = F(s, s')/F(s)$, where $F(s, s')$ is the number of times the sequence $(a_1, \ldots, a_{k+1})$ appears in the training data, and $F(s)$ is the number of times the sequence $(a_1, \ldots, a_k)$ appears in the training data. Given a sequence of testing data, anomalies are detected by sliding a window of size $k$ along the testing data: the current state $s$ is given by the current $k$ symbols, the next state $s'$ is given by the $k$ symbols after sliding the window by one position, and a “surprise factor” is calculated as $1 - p(s, s')$. If the surprise factor is above a pre-defined anomaly threshold, then the detector declares that an anomaly has occurred, and the current position in the testing data can be used to provide information about where the anomaly occurred.
4.3.2 Markov Chain kernel

Let alphabet set $Z$ represent all the possible system calls that appear in the privileged system program and $k$ be the sliding window size used by the Markov Chain model. The input space $Z^*$ is a space of sequences of characters from the alphabet $Z$. The k-Markov Chain kernel maps from $Z^*$ to a $(|Z|^{k+1})$-dimensional feature vector space.

The feature map is defined as follows: $\Phi_k(s) = (p(s_k, s_k'), s_k, s_k' \in A)$, where $s \in Z^*$; $s_k$ and $s_k'$ are two states in a Markov Chain, each of them represents a k-size sequence from alphabet $Z$, and $A$ is the set of states of the Markov Chain; $p(s_k, s_k')$ is the probability of transition from state $s_k$ to state $s_k'$ in the sequence $s$.

The k-Markov Chain kernel is:

$$K_k(s_1, s_2) = \Phi_k(s_1) \cdot \Phi_k(s_2)$$

The original Markov Chain anomaly detector can be described using the feature mapping we defined above to calculate a probability vector $(p(s, a'))_{a' \in A}$ where $s$ is a state that represents a k-size sequence from the alphabet $Z$ that exists in the privileged daemons and $p(s, a')$ is the probability of transition from the state $s = (a_1, a_2, \ldots, a_k)$ to the state $s' = (a_2, \ldots, a_k, a')$.

The Markov Chain anomaly detector uses a simple linear threshold function. In the training phase, a probability vector $v_0$, representing the probability of each transition, is calculated: $v_0 = (p(s, a'))_{a' \in Z}$, where $s$ is a state that represents a k-size sequence from the alphabet $Z$ that exists in the program execution; $p(s, a')$ is the probability of a transition from state $s = (a_1, a_2, \ldots, a_k)$ to state $s' = (a_2, \ldots, a_k, a')$. Since the Markov Chain anomaly detector identifies a sequence as an abnormal sequence when any of the transition probability in the trace is lower than an anomaly threshold, we represent the probability vector $v_0$ by a family of probability vectors $v_1, v_2, \ldots, v_n$, with $v_0 = v_1 + v_2 + \ldots + v_n$. Each vector in the family of vectors has one and only one feature with non-zero value and represents probability of one transition. For example, the vector $v_0 = (0.1, 0, 0.3, 0, 0, 0.9)$ will be represented by $(0.1, 0, 0, 0, 0, 0) + (0, 0, 0.3, 0, 0, 0) + (0, 0, 0, 0, 0, 0.9)$.

In the testing phase, for each system sequence $s$ in the testing data, a probability vector $v$, which represents the features of the subsequence transition probability in $s$, is calculated
using the feature mapping: \( \mathbf{v} = \Phi_k(s) \) as described above. \( \mathbf{v} \) is then mapped to \( \mathbf{v}' \) where each non-zero value in \( \mathbf{v} \) is mapped to 1 in \( \mathbf{v}' \) and each zero value in \( \mathbf{v} \) keeps the same in \( \mathbf{v}' \).

The dot product of \( \mathbf{v}' \) and each vector \( \mathbf{v}_i \) in the family of the probability vectors is compared against a predefined threshold \( \delta \). When there is a dot product value less than or equal to the threshold \( \delta \), an anomaly is reported, i.e., the linear threshold function for anomaly detection is: 

\[
\min \{ \mathbf{v}' \cdot \mathbf{v}_i \} \leq \delta, \text{anomaly;} \quad \min \{ \mathbf{v}' \cdot \mathbf{v}_i \} > \delta, \text{normal.}
\]

### 4.3.3 Efficient computation of Markov Chain kernel

For the Markov Chain method, since the system call sequence length is \( k \) and the alphabet size is \( |Z| \), there can be \( |Z|^k \) permutations of all possible system call sequences. The Markov Chain method needs to calculate all the possible transitions between system call sequences, which means the vector length is \( O(|Z|^{k+1}) \), so the algorithm complexity is \( O(|Z|^{k+1}) \). This computation soon becomes infeasible with the increase of the window size.

To calculate a \( k \)-Markov Chain kernel efficiently, all the subsequences of size \( k \) that do not exist in the log file of the privileged system program can be neglected as the STIDE kernel. Thus, the complexity of the \( k \)-Markov Chain kernel method is reduced to \( O(|A| \times |Z|) \) where \( Z \) is the alphabet size and \( A \) is the set of all the subsequences with size-\( k \) that exist in the traces of the privileged system program assuming it is already a priori knowledge. Without peeking into both the training and testing data, we can also build the vector from the training data and ignore those new features found in the testing data, however, the classification accuracy rate will suffer from ignoring the new features. Another approach, which uses specification guided method to find all the relevant features of the privileged system program, will be described in Chapter 6.

The main disadvantage of STIDE and Markov Chain anomaly detectors is their over-simplicity. They completely depend on the training data to cover all the possible scenarios; thus the over-fitting can be a problem. By substituting the simple linear threshold function with SVM, which emphasizes on maximizing generalization capability and overcome the overfitting problem, improvement can be achieved over the STIDE and Markov Chain anomaly detectors.
Different from STIDE and Markov anomaly detectors, the learned model of STIDE and Markov Chain kernel based SVM anomaly detectors are composed of a group of support vectors. The standard vectors used by STIDE and Markov anomaly detectors are much simpler than the support vectors. Theoretically, the learned models (composed of support vectors) by SVM has less true error, which is the expected error on unseen testing data from the same distribution with the training data, than models learned from other learning algorithms. We conduct experiments on the UNM data and the DARPA Lincoln Lab data and empirically show that STIDE and Markov Chain kernels based SVM have a better accuracy rate than the original STIDE and Markov Chain methods.

Another disadvantage of STIDE and Markov Chain anomaly detectors is the selection of a threshold. The threshold is critical in these two approaches, however, there is no efficient and effective method for selecting a suitable threshold value. Using the two class SVM method combined with the STIDE and Markov Chain kernels, it is no longer necessary to select a threshold value.

4.4 Experiments on STIDE and Markov Chain kernel based Support Vector Machine (SVM) anomaly detectors

We use a publicly available SVM software libSVM, which is an implementation of Vapnik’s Support Vector Machine [9], to conduct our experiments. The optimization algorithm used in libSVM are described in [30].

4.4.1 Experiments on the UNM data

We test our approach and compare them with the STIDE and Markov Chain anomaly detectors on the UNM sendmail data set [49]. UNM sendmail data includes hundreds of normal executions of sendmail daemons and several traces of sendmail daemons that are attacked. We compare the performance of the STIDE detector, Markov Chain detector, STIDE kernel based SVM and Markov Chain kernel based SVM anomaly detectors on UNM sendmail data. Table 4.1 shows the comparison results using the UNM data set (sliding
window size is 2).

In our experiments, we use two kinds of STIDE kernels, STIDE frequency kernel, which uses frequency of system call subsequences in a feature vector, and STIDE 0/1 kernel, which uses 0/1 to indicate the existence of a system calls sequence in a feature vector.

As we can see, the STIDE and Markov Chain kernel based SVM anomaly detectors achieve much lower false alarm rates than the original STIDE and Markov anomaly detectors. The detection rate and correlation coefficient of STIDE and Markov Chain kernel based SVM anomaly detectors are also better than the original STIDE and Markov Chain kernel based SVM anomaly detectors. This results comply with our analysis in the previous sections, i.e., STIDE and Markov Chain anomaly detectors are composed of simple threshold functions and have less generalization capability than STIDE kernel and Markov Chain kernel based SVM anomaly detector, thus STIDE and Markov Chain kernel based SVM can achieve higher detection rate and lower false alarm rate than the original STIDE and Markov Chain anomaly detectors.

<table>
<thead>
<tr>
<th>Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stide</td>
<td>90.7%</td>
<td>91.6%</td>
<td>9.8%</td>
<td>0.79</td>
</tr>
<tr>
<td>Stide kernel (0/1) SVM</td>
<td>95.2%</td>
<td>92.0%</td>
<td>1.87%</td>
<td>0.90</td>
</tr>
<tr>
<td>Stide kernel (frequency) SVM</td>
<td>94.1%</td>
<td>92.3%</td>
<td>3.42%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov Chain</td>
<td>94.2%</td>
<td>91.7%</td>
<td>4.3%</td>
<td>0.80</td>
</tr>
<tr>
<td>Markov Chain kernel SVM</td>
<td>96.7%</td>
<td>94.2%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
</tbody>
</table>

We also test the combination of SVM with a polynomial kernel and a Radial Basis Function (RBF) kernel. Table 4.2 shows the comparison between polynomial kernel, RBF kernel, STIDE kernel and Markov Chain kernel based SVM using the UNM dataset. The experiments show the STIDE kernel and Markov Chain kernel, achieve higher accuracy rate and correlation coefficient than general kernels, such as RBF kernel and Polynomial kernel. The false alarm rates are similar. The STIDE kernel and Markov Chain kernel is designed to capture more knowledge, such as time and order information, than standard kernels, which explains why the above experiments showed our STIDE and Markov Chain kernels achieve better accuracy.
than that of the SVM anomaly detectors based on polynomial kernels and RBF kernels.

Table 4.2 Comparison of polynomial kernel, RBF kernel, STIDE kernel and Markov Chain kernel based anomaly detectors on the UNM data

<table>
<thead>
<tr>
<th>Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial kernel</td>
<td>90.7%</td>
<td>80.7%</td>
<td>1.02%</td>
<td>0.82</td>
</tr>
<tr>
<td>RBF kernel</td>
<td>94.0%</td>
<td>89.8%</td>
<td>2.04%</td>
<td>0.88</td>
</tr>
<tr>
<td>Stide kernel (0/1) SVM</td>
<td>95.2%</td>
<td>92.0%</td>
<td>1.87%</td>
<td>0.90</td>
</tr>
<tr>
<td>Stide kernel (frequency) SVM</td>
<td>94.1%</td>
<td>92.3%</td>
<td>3.42%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov Chain kernel SVM</td>
<td>96.7%</td>
<td>94.2%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
</tbody>
</table>

4.4.2 Experiments on the DARPA Lincoln Lab data

We also used DARPA Lincoln Lab data to test our new anomaly detectors. Instead of only recording system call traces using utilities like `strace`, the DARPA data is recorded using SUN’s Basic Security Module (BSM). The detailed information about system call events such as arguments and return values are recorded.

The comparison of the performance of the STIDE detector, Markov Chain detector, STIDE kernel based SVM and Markov Chain kernel based SVM on the DARPA Lincoln Lab data is shown in Table 4.3 and Table 4.4. Table 4.3 shows the result on a single type of process, FTP process, in the DARPA Lincoln Lab data. Similar with UNM data, the results for the DARPA Lincoln Lab data show STIDE and Markov Chain based SVM have much lower false alarm rates and higher detection rate than the original STIDE and Markov anomaly detectors. These results on DARPA Lincoln Lab data comply with our analysis in the previous sections, i.e., STIDE and Markov Chain anomaly detectors have less generalization capability than STIDE and Markov Chain kernel based anomaly detectors, and the detection rates and the false alarm rates of the new SVM based anomaly detectors are better than the original STIDE and Markov Chain anomaly detectors.

Table 4.4 shows the combination result on multiple processes (`ftp`, `ffconfig`, `fdformat` and `eject`) in the DARPA Lincoln Lab data. The experiment also demonstrates STIDE and Markov Chain kernel based SVM anomaly detectors have a much lower false alarm rate and
higher detection rate than original STIDE and Markov anomaly detectors on these processes in the DARPA Lincoln Lab data, based on the same reasoning.

Table 4.3 Comparison of STIDE, Markov Chain, STIDE kernel based and Markov Chain kernel based two class SVM anomaly detectors on the FTP processes in the DARPA Lincoln Lab data

<table>
<thead>
<tr>
<th>Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stide</td>
<td>95.1%</td>
<td>95.6%</td>
<td>5.4%</td>
<td>0.83</td>
</tr>
<tr>
<td>Stide kernel (0/1) SVM</td>
<td>97.4%</td>
<td>98.3%</td>
<td>3.0%</td>
<td>0.92</td>
</tr>
<tr>
<td>Stide kernel (frequency) SVM</td>
<td>97.8%</td>
<td>98.1%</td>
<td>3.0%</td>
<td>0.85</td>
</tr>
<tr>
<td>Markov Chain</td>
<td>96.4%</td>
<td>96.3%</td>
<td>3.4%</td>
<td>0.83</td>
</tr>
<tr>
<td>Markov Chain kernel SVM</td>
<td>98.9%</td>
<td>99.1%</td>
<td>2.7%</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Table 4.4 Comparison of STIDE, Markov Chain, STIDE kernel based and Markov Chain kernel based two class SVM anomaly detector on multiple processes in the DARPA Lincoln Lab data

<table>
<thead>
<tr>
<th>Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stide</td>
<td>88.1%</td>
<td>86.2%</td>
<td>10.9%</td>
<td>0.53</td>
</tr>
<tr>
<td>Stide kernel (0/1) SVM</td>
<td>90.4%</td>
<td>89.3%</td>
<td>5.9%</td>
<td>0.70</td>
</tr>
<tr>
<td>Stide kernel (frequency) SVM</td>
<td>92.1%</td>
<td>90.3%</td>
<td>4.6%</td>
<td>0.71</td>
</tr>
<tr>
<td>Markov Chain</td>
<td>89.7%</td>
<td>88.1%</td>
<td>9.8%</td>
<td>0.65</td>
</tr>
<tr>
<td>Markov Chain kernel SVM</td>
<td>93.9%</td>
<td>94.5%</td>
<td>3.5%</td>
<td>0.81</td>
</tr>
</tbody>
</table>

4.5 Impact of window size on the coverage of system call sequence based intrusion detection

Previous research showed that the window size used in system call sequence based intrusion detectors, such as STIDE and Markov Chain anomaly detectors discussed in this section, has a significant impact on the detection coverage of these detectors [67]. Tan's paper "Why 6?: Defining the Operational Limits of STIDE, an Anomaly-Based Intrusion Detector" addressed this issue [67], and explained why the detection rate of an intrusion detector will not go up after its sliding window size is increased beyond 6. It also compared the coverage of the Stide anomaly detector [17] with a Markov-based anomaly detector [29].
4.5.1 Coverage of STIDE anomaly detector

Earlier research results proved that this STIDE anomaly detector has limited detection coverage [42]. Following the definitions in [42], a foreign sequence is a sequence whose individual symbols appear in the training data, but the foreign sequence itself does not appear in the training sequence. We say a sequence is normal if it is not foreign. A minimal foreign sequence is a foreign sequence whose proper subsequences are all normal.

The STIDE detector described in [67] uses a fixed-size detector window of size \( N \) to build a STIDE intrusion detector. The experiment results showed a strong relationship between the length of the minimal foreign sequence and the length of the detector window required to detect an attack (see Figure 4.1). The STIDE anomaly detector has the following detection coverage [42]:

1. If the largest minimal foreign sequence in the anomaly space has length \( M \), then a STIDE anomaly detector with window size \( N \geq M \) can correctly classify any sequence.

2. If the anomaly space contains a minimal foreign sequence of length \( M \), then a STIDE anomaly detector with window size \( N < M \) cannot correctly classify all sequences.

4.5.2 Coverage of Markov Chain anomaly detector

It was claimed in Tan's paper [42], through the use of experiments involving synthetic data, that the Markov-based detector is able to cover 100% of the anomaly space in their
experiment; i.e., it is able to detect minimal foreign sequences of any size. Instead, we show that a Markov-based detector with window size $N$ cannot detect a minimal foreign sequence with length larger than $N+1$, unless it also incorrectly classifies normal behavior as anomalous. Section 4.5.2.1 gives an example of this, and Section 4.5.2.2 presents a formal analysis of the coverage of a Markov-based anomaly detector.

### 4.5.2.1 An example

Consider the short training sequence "abcabcacabac". Following the procedure outlined above and described in [42], a Markov-based detector with window size 3 can be constructed, as shown in Figure 4.2. Note that the sequence "cbcab" does not appear in the training sequence, and note that all proper subsequences of "cbcab" do appear in the training sequence. Thus, "cbcab" is a minimal foreign sequence. The Markov-based detection algorithm in [42] will process this sequence by determining the surprise factor for transition $\text{cbe} \rightarrow \text{bca}$, which is $1.0 - 1.0 = 0$, and for transition $\text{bca} \rightarrow \text{cab}$, which is $1.0 - 0.5 = 0.5$. Thus, the anomaly "cbcab" is detected if and only if the anomaly threshold is less than 0.5. However, with such a low threshold, many sequences that should be classified as normal, because they appear in the training data, will be incorrectly classified as anomalous (e.g., "abeb", "abca"). Indeed, the sequence "cbcab" will be classified as anomalous if and only if either "cbca" or "bcab" is classified as anomalous. The difficulty stems from the fact that the sequences "cbca" and "bcab" are normal, while "cbcab" is anomalous: since the window size is 3, the detector is able to consider the last 3 symbols plus the next symbol to occur (which causes the transition to the next state). It is therefore able to consider 4 symbols, while the anomaly "cbcab" is a minimal foreign sequence of length 5.

### 4.5.2.2 Our proof on coverage of a Markov-based detector

We say that an anomaly detector can correctly classify a foreign sequence as anomalous if it determines at some point in the sequence that an anomaly has occurred. Similarly, we say that a normal sequence is correctly classified if the detector does not determine that an anomaly has occurred (unless the normal sequence is part of a larger foreign sequence).
Figure 4.2 Markov chain constructed from training data “abcabcbcacabc” using window size 3

**Theorem 1** If the anomaly space contains a minimal foreign sequence of length $M$, then a Markov-based detector with window size $N < M - 1$ cannot correctly classify all sequences.

**Proof:** Let $s = (s_1, \ldots, s_M)$ be a minimal foreign sequence, with $M > N + 1$. By definition, all subsequences $s_i = (s_i, \ldots, s_{i+N-1})$ of length $N$ are normal, as are all subsequences of length $N + 1$. This implies that all transition probabilities $p(s_i, s_{i+1})$ are positive, and their corresponding surprise factors are less than one. Let $\gamma$ be the anomaly threshold. Suppose there exists $i$ such that $1 - p(s_i, s_{i+1}) \geq \gamma$. In this case, the sequence $s$ will be correctly classified as anomalous; however, the normal sequence $(s_i, \ldots, s_{i+N})$ will be incorrectly classified as anomalous. Conversely, suppose that $1 - p(s_i, s_{i+1}) < \gamma$ for all $i$; then the sequence $s$ will be incorrectly classified as normal.

In Tan’s paper [67], a threshold of 1.0 is used. To compare our new claims with Tan’s paper [42], we also suppose the threshold is 1.0 and can draw the following corollary:

**Corollary 1** Using a threshold 1.0, a Markov-based detector with window size $N < M - 1$ cannot detect a minimal foreign sequence of length $M$.

**Proof:** Let $s = (s_1, \ldots, s_M)$ be a minimal foreign sequence, with size $M > N + 1$. By definition, all subsequences $s_i = (s_i, \ldots, s_{i+N-1})$ of length $N$ are normal, as are all subsequences of length $N + 1$. This implies that all transition probabilities $p(s_i, s_{i+1})$ are positive, and their corresponding surprise factors are less than one. Thus it will be identified as normal.
Theorem 2 If the largest minimal foreign sequence in the anomaly space has length $M$, then there exists a threshold that a Markov-based detector with window size $N > M - 1$ can correctly classify any sequence.

Proof: Let 1.0 be the anomaly threshold value. First, consider a normal sequence $s$, where any subsequence $s_i$ of length $N$ starting at position $i$ is normal. Thus, for all $i$, $s_i$ is normal, as is $s_{i+1}$; this implies that $s_i$ and $s_{i+1}$ are states in the Markov chain. Similarly, all subsequences of length $N+1$ are normal, so $(s_i, \ldots, s_{i+N})$ is normal, which implies that there is a transition from $s_i$ to $s_{i+1}$ with nonzero probability; then the surprise factor is less than one for each $i$, which indicates normality.

Second, consider a sequence containing a minimal foreign sequence $s = (s_i, \ldots, s_{i+L-1})$ where length $L \leq N$. Then no state in the Markov chain contains the subsequence $s$ since $s$ is foreign. Thus, while checking the sequence for an anomaly using the Markov-based detector, the desired target state will not exist, which implies that an anomaly has occurred.

If the minimal foreign sequence length $L = N+1$, then the subsequences $s_1 = (s_i, \ldots, s_{i+N-1})$ and $s_2 = (s_{i+1}, \ldots, s_{i+N})$ are both normal, and are thus states in the Markov chain. Since $(s_i, \ldots, s_{i+N})$ is foreign, the transition probability $p(s_1, s_2)$ is zero, leading to a surprise factor of one, which indicates an anomaly.

We can easily draw the following corollary from Theorem 2:

Corollary 2 Using anomaly threshold 1.0, if the minimal foreign sequence in the anomaly
space has length \( M \), then a Markov-based detector with window size \( N \geq M - 1 \) can correctly classify this sequence.

Given the above proof that the Markov-based detector cannot detect all anomalies using a given window size, the coverage of Markov-based anomaly detectors which are drawn from the experiments carried by Tan et al [42] (depicted in Figure 4.3) can not be applied on all data sets. A more accurate map of the coverage area of Markov-based detectors is shown in Figure 4.4.

### 4.5.3 Impact of window size on SVM anomaly detector

The coverage of STIDE and Markov Chain kernel based SVM detectors is also impacted by the sliding window size. Our experiment showed that when the sliding window size increases, the accuracy rate also increases and then drops down after some points (as Table 4.5, Table 4.6 and Table 4.7). These experiments are performed using UNM data.

<table>
<thead>
<tr>
<th>STIDE kernel-based SVM detector</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>winsize=1</td>
<td>94.0%</td>
<td>89.8%</td>
<td>2.04%</td>
<td>0.88</td>
</tr>
<tr>
<td>winsize=2</td>
<td>95.2%</td>
<td>92.0%</td>
<td>1.87%</td>
<td>0.90</td>
</tr>
<tr>
<td>winsize=3</td>
<td>95.2%</td>
<td>91.3%</td>
<td>1.02%</td>
<td>0.91</td>
</tr>
<tr>
<td>winsize=4</td>
<td>94.9%</td>
<td>90.5%</td>
<td>1.02%</td>
<td>0.90</td>
</tr>
<tr>
<td>winsize=5</td>
<td>94.5%</td>
<td>89.9%</td>
<td>1.02%</td>
<td>0.89</td>
</tr>
<tr>
<td>winsize=6</td>
<td>94.1%</td>
<td>88.9%</td>
<td>1.02%</td>
<td>0.88</td>
</tr>
<tr>
<td>winsize=10</td>
<td>93.5%</td>
<td>87.5%</td>
<td>0.68%</td>
<td>0.87</td>
</tr>
</tbody>
</table>

When the sliding window size increases, each feature for SVM detector represents a longer system call sequence, which renders more information than shorter system call sequence. For example, two traces \( \text{abcaba} \) and \( \text{ababca} \) is being considered. If we use a window size 2, assuming the features for the two traces are \( \{ \text{ab, ba, bc, cb, ca, ac}\} \), the feature vectors are all \( \{1, 1, 1, 0, 1, 1\} \) for the two traces. If we use a window size 3, assuming the features for the two traces are \( \{ \text{aaa, aab, aac, abc, aba, abb, bca, bcb, bab, cab, cba}\} \), the feature vectors for the two traces are \( \{0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0\} \) and \( \{0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0\} \). Thus, using
Table 4.6 Window size impact on the STIDE frequency kernel based SVM anomaly detectors on the UNM data

<table>
<thead>
<tr>
<th>STIDE frequency based SVM detector</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>winsize=1</td>
<td>95.6%</td>
<td>96.4%</td>
<td>5.47%</td>
<td>0.90</td>
</tr>
<tr>
<td>winsize=2</td>
<td>94.1%</td>
<td>92.3%</td>
<td>3.42%</td>
<td>0.88</td>
</tr>
<tr>
<td>winsize=3</td>
<td>93.7%</td>
<td>91.4%</td>
<td>3.08%</td>
<td>0.87</td>
</tr>
<tr>
<td>winsize=4</td>
<td>94.2%</td>
<td>91.9%</td>
<td>2.39%</td>
<td>0.88</td>
</tr>
<tr>
<td>winsize=5</td>
<td>93.4%</td>
<td>90.7%</td>
<td>2.73%</td>
<td>0.87</td>
</tr>
<tr>
<td>winsize=6</td>
<td>93.4%</td>
<td>90.2%</td>
<td>2.05%</td>
<td>0.87</td>
</tr>
<tr>
<td>winsize=10</td>
<td>92.1%</td>
<td>88.0%</td>
<td>2.05%</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Table 4.7 Window size impact on the Markov Chain kernel based SVM anomaly detectors on the UNM data

<table>
<thead>
<tr>
<th>Markov Chain kernel based SVM Detector</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>winsize=1</td>
<td>95.8%</td>
<td>92.9%</td>
<td>1.92%</td>
<td>0.91</td>
</tr>
<tr>
<td>winsize=2</td>
<td>96.7%</td>
<td>94.2%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
<tr>
<td>winsize=3</td>
<td>96.7%</td>
<td>94.3%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
<tr>
<td>winsize=4</td>
<td>96.9%</td>
<td>94.2%</td>
<td>0.90%</td>
<td>0.94</td>
</tr>
<tr>
<td>winsize=5</td>
<td>93.1%</td>
<td>94.6%</td>
<td>6.9%</td>
<td>0.88</td>
</tr>
<tr>
<td>winsize=6</td>
<td>88.9%</td>
<td>94.2%</td>
<td>18.3%</td>
<td>0.71</td>
</tr>
<tr>
<td>winsize=10</td>
<td>78.8%</td>
<td>90.1%</td>
<td>34.3%</td>
<td>0.62</td>
</tr>
</tbody>
</table>

When the window size is lower than the minimal foreign sequence for the STIDE and Markov Chain based kernels, normal traces and abnormal traces can not be represented in different feature vectors. When the window size is greater than the minimal foreign sequence for a STIDE or a Markov Chain kernel, a useful pattern can not be generated.

The accuracy rate increases with the increase of window size, it reaches the highest for both STIDE and Markov Chain kernel based SVM intrusion detectors when the window size is close to the minimal foreign sequence size. After that, the detection rate decreases with the window size increases for both STIDE and Markov Chain kernel based detectors.
4.6 Summary

This chapter investigates application based kernels for intrusion detection. Different from STIDE and Markov Chain anomaly detectors, the solutions of STIDE and Markov Chain kernel based SVM anomaly detectors are composed of a group of support vectors and decides a vector to be normal or abnormal. The standard abnormal vectors in STIDE and Markov Chain anomaly detectors are much simpler than the support vectors, and thus STIDE and Markov Chain anomaly detectors are less precise than the SVM based anomaly detectors using STIDE and Markov Chain kernel. Theoretically, the solution composed of these support vectors derived using SVM approach has less true error, expected error on unseen data from the same distribution with the training data, than that of other machine learning algorithms.

We also experiment on combining the STIDE kernel and Markov Chain kernel with SVM to improve the classification result for anomaly intrusion detection. The results provide strong evidence that STIDE kernels and Markov Chain kernels, in conjunction with SVMs, could offer an alternative to conventional anomaly detection algorithms (STIDE and Markov Chain methods) for detecting anomalies in system call sequences, with a much lower false alarm rate and a better detection rate. The two new SVM based anomaly detectors does not need threshold, which is critical for original STIDE and Markov Chain anomaly detectors.

In next chapter, we are going to present the one class SVM method for unsupervised learning for anomaly intrusion detection.
CHAPTER 5. One class SVM for anomaly detection

Traditional machine learning based anomaly detection methods need labeled audit data for the purpose of training. However, in intrusion detection field, the following reasons make labeling audit data for the purpose of training a difficult, if not impossible, task.

- Attacks must be identified first to label the audit data. It is not easy to identify all the attacks or get attack free audit data.
- Matching attacks with the related audit records is a time consuming task.
- The sheer volume of audit data makes it difficult to process them.
- The frequent needs of retraining because of environmental changes require frequent labeling of audit data.
- It is not easy to guarantee that normal data does not include any attacks.

Unsupervised learning is a promising method to solve this problem. In supervised learning, learning relies on labeled training data; while in unsupervised learning, training data does not need to be labeled. Unsupervised learning is very beneficial for the intrusion detection domain, since the labeled data is difficult to obtain while unlabeled data can be obtained very easily from audit data files.

5.1 Introduction

Unsupervised learning studies “how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns” [46]. Using unsupervised learning, no labeled audit data or attack free audit data is needed.
To group related instances into clusters or more general hierarchies of clusters (taxonomies) is very important in understanding data, and this kind of unsupervised learning is also called *clustering*. To group instances into anomaly and normal instances is the basic task for anomaly detection.

5.1.1 Clustering

There are two steps in the clustering task. In the first step, unlabeled data is used to train classifiers, and two or more clusters are formed from the unlabeled data. In the second step, for new data, calculate its distance to each cluster to decide which cluster it belongs to.

Some popular clustering methods are as following:

**K-means Clustering:** Cluster centers are randomly assigned first; then, points are repetitively assigned to clusters whose centers are the closest until clusters stop changing.

**Hierarchical Agglomerative Clustering:** All points are initially put in separate clusters.

Then, the closest pairs of clusters are merged together.

**Naïve Algorithm:** All possible k-clusterings are enumerated and the one that optimizes the clustering metric is selected.

**Farthest Point Clustering Algorithm:** An arbitrary center is selected first, then, the one farthest from its closest center is repetitively selected as the next center until k centers are selected.

**Divide and Conquer Algorithm:** Data is arbitrarily partitioned first. Then each partition is clustered. Lastly, the clusterings in each partition are put together and are clustered.

Portnoy et al. from Columbia University studied clustering methods for anomaly detection [53]. Their result showed the accuracy rates of the clustering methods for anomaly detection is not as high as using labeled data. However, it is still useful since the automatic clustering takes much less efforts in the training procedure than the supervised learning since it does not rely on labeled data. We propose to use one class SVM, a variation of SVM, for unsupervised learning for anomaly intrusion detection.
One class SVM, which is adapted from two class SVM and is used for clustering [57], performs better than the above mentioned clustering techniques. In the following section, we will describe the clustering technique used in one class SVM.

### 5.1.2 One class SVM

SVM is originally two-class based and used in supervised learning. It is adapted to one class SVM by a method introduced in [57] and is used for unsupervised learning.

The strategy of one class SVM is trying to use a hypersphere to describe the data in feature space and put most of the data into the hypersphere. The trade-off between the radius of the hypersphere and the number of training examples that it can hold is set by a parameter $v \in [0, 1]$. Lagrangian multipliers are used to solve this optimization problem [57]. The data points that are outside of the hypersphere are also called "outliers", and are negative examples; the data points that are inside the hypersphere are positive examples.

A hypersphere is used to do a linear division. However, some data sets are not linearly separable. Kernel functions, such as polynomial kernels and RBF kernels, are widely used in SVMs to map original input data sets to a high dimension feature space in order to make the data sets linearly separable. We experimented using STIDE kernel and Markov Chain kernel introduced in the previous chapter, combined with one class SVM, for anomaly detection.

### 5.2 Experiments on one class SVM based anomaly detectors

We experimented using the STIDE and Markov Chain kernels with SVMs and compared the SVM-based detectors with the STIDE and Markov Chain detectors as described in [17, 42]. We use a publicly available SVM software libSVM, which implements Vapnik’s one class SVM [9].

#### 5.2.1 Experiments on the UNM data

We first experiment using the sendmail data set from the University of New Mexico [49]. The results of our comparisons are shown in Table 5.1, using a ratio of 0.1 for the one class
Table 5.1  Comparison of STIDE, Markov Chain, STIDE kernel based and 
Markov kernel based SVM anomaly detectors on the UNM data

<table>
<thead>
<tr>
<th>Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>STIDE</td>
<td>90.7%</td>
<td>91.6%</td>
<td>9.8%</td>
<td>0.79</td>
</tr>
<tr>
<td>Stide kernel (0/1) two class SVM</td>
<td>95.2%</td>
<td>92.0%</td>
<td>1.87%</td>
<td>0.90</td>
</tr>
<tr>
<td>Stide kernel (frequency) two class SVM</td>
<td>94.1%</td>
<td>92.3%</td>
<td>3.42%</td>
<td>0.88</td>
</tr>
<tr>
<td>STIDE kernel (0/1) one class SVM</td>
<td>89.6%</td>
<td>88.1%</td>
<td>8.3%</td>
<td>0.67</td>
</tr>
<tr>
<td>STIDE kernel (frequency) one class SVM</td>
<td>93.1%</td>
<td>92.4%</td>
<td>3.4%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov Chain</td>
<td>94.2%</td>
<td>91.7%</td>
<td>4.3%</td>
<td>0.80</td>
</tr>
<tr>
<td>Markov Chain kernel two class SVM</td>
<td>96.7%</td>
<td>94.2%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
<tr>
<td>Markov kernel one class SVM</td>
<td>94.3%</td>
<td>93.7%</td>
<td>6.0%</td>
<td>0.80</td>
</tr>
</tbody>
</table>

The results show that the STIDE kernel based SVM anomaly detector has a slight higher
detection rate and lower false alarm than the original STIDE anomaly detector for the UNM
data. Markov kernel based SVM anomaly detector has also a slightly higher detection rate
and lower false alarm rate than the original Markov anomaly detector. This complies
with our analysis of generalization capability in Chapter 4, i.e., STIDE and Markov Chain
anomaly detectors are composed of simple threshold functions and have less generalization
capability than STIDE kernel and Markov Chain kernel based SVM anomaly detector, thus
by substituting the simple threshold function with SVM, STIDE and Markov Chain kernel
based SVM can achieve higher detection rate and lower false alarm rate than the original
STIDE and Markov Chain anomaly detectors.

We also test the combination of one class SVM with a polynomial kernel and a RBF kernel.
Table 5.2 shows the comparison between polynomial kernel, RBF kernel, STIDE 0/1 kernel
and STIDE frequency kernel based one class SVM using the same UNM dataset. We can
see, compared with general kernel methods such as polynomial kernels and RBF kernels, the
STIDE and Markov kernels we developed for anomaly intrusion detection achieve a higher
detection rate and lower false alarm rate.

As shown in Table 5.1, Table 5.2, compared to the two class STIDE kernel and Markov
kernel based SVM anomaly detector we introduced in previous chapter, the one class STIDE
kernel and Markov kernel based SVM anomaly detectors have lower detection rate and higher
Table 5.2: Comparison of polynomial kernel, RBF kernel, STIDE kernel and Markov Chain kernel based SVM anomaly detectors (ratio threshold = 0.1) on the UNM data

<table>
<thead>
<tr>
<th>SVM anomaly Detector (win size=2)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>One class SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polynomial kernel</td>
<td>90.7%</td>
<td>91.0%</td>
<td>11.4%</td>
<td>0.77</td>
</tr>
<tr>
<td>RBF kernel</td>
<td>92.1%</td>
<td>91.9%</td>
<td>4.5%</td>
<td>0.84</td>
</tr>
<tr>
<td>STIDE kernel (0/1)</td>
<td>89.6%</td>
<td>88.1%</td>
<td>8.3%</td>
<td>0.67</td>
</tr>
<tr>
<td>STIDE kernel (frequency)</td>
<td>93.1%</td>
<td>92.4%</td>
<td>3.4%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov kernel</td>
<td>92.3%</td>
<td>90.7%</td>
<td>6.0%</td>
<td>0.80</td>
</tr>
<tr>
<td>Two class SVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polynomial kernel</td>
<td>90.7%</td>
<td>80.7%</td>
<td>1.02%</td>
<td>0.82</td>
</tr>
<tr>
<td>RBF kernel</td>
<td>94.0%</td>
<td>89.8%</td>
<td>2.04%</td>
<td>0.88</td>
</tr>
<tr>
<td>STIDE kernel (0/1)</td>
<td>95.2%</td>
<td>92.0%</td>
<td>1.87%</td>
<td>0.90</td>
</tr>
<tr>
<td>STIDE kernel (frequency)</td>
<td>94.1%</td>
<td>92.3%</td>
<td>3.42%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov Chain kernel</td>
<td>96.7%</td>
<td>94.2%</td>
<td>1.36%</td>
<td>0.93</td>
</tr>
</tbody>
</table>

false alarm rate. However, since no labeled data needed for training, much less effort is needed for training as in the case of two class SVM. The trade-off of detection accuracy with saving of training effort is still preferable for many system administrators.

5.2.2 Experiments on the DARPA Lincoln Lab data

We also used the DARPA Lincoln Lab data to test the one class SVM anomaly detectors. The comparison of the performance of the five anomaly detectors - STIDE, Markov Chain detector, STIDE 0/1 kernel based SVM, STIDE frequency kernel based SVM and Markov Chain kernel based SVM - on the DARPA data are showed in Table 5.3. The experiment demonstrates STIDE and Markov Chain kernel based SVM anomaly detectors get a slight higher detection rate and lower false alarm rate than STIDE and Markov anomaly detectors on the DARPA data. Also, the Markov Chain kernel based SVM anomaly detectors get higher detection rate and lower false alarm rate than the STIDE kernel based SVM anomaly detectors.

5.3 Discussion of the experiment results

The training data for STIDE and Markov Chain anomaly detectors must be pure normal data. If the training data includes some intrusions, the STIDE and Markov Chain anomaly
Table 5.3 Comparison of STIDE, Markov Chain, STIDE kernel based one class SVM, Markov Chain kernel based SVM anomaly detectors on the DARPA data

<table>
<thead>
<tr>
<th>Detectors (win size=2)</th>
<th>Accuracy rate</th>
<th>Detection rate</th>
<th>False alarm rate</th>
<th>Correlation coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stide</td>
<td>88.1%</td>
<td>86.2%</td>
<td>10.9%</td>
<td>0.53</td>
</tr>
<tr>
<td>Stide 0/1 kernel two class</td>
<td>90.4%</td>
<td>89.3%</td>
<td>5.9%</td>
<td>0.70</td>
</tr>
<tr>
<td>Stide frequency kernel two class</td>
<td>92.1%</td>
<td>90.3%</td>
<td>4.6%</td>
<td>0.71</td>
</tr>
<tr>
<td>Stide 0/1 kernel one class</td>
<td>88.7%</td>
<td>87.1%</td>
<td>9.6%</td>
<td>0.69</td>
</tr>
<tr>
<td>Stide frequency kernel one class</td>
<td>88.8%</td>
<td>87.4%</td>
<td>9.7%</td>
<td>0.70</td>
</tr>
<tr>
<td>Markov Chain</td>
<td>93.0%</td>
<td>92.1%</td>
<td>6.7%</td>
<td>0.83</td>
</tr>
<tr>
<td>Markov Chain kernel two class</td>
<td>93.9%</td>
<td>94.5%</td>
<td>3.5%</td>
<td>0.81</td>
</tr>
<tr>
<td>Markov Chain kernel one class</td>
<td>93.2%</td>
<td>91.1%</td>
<td>6.7%</td>
<td>0.81</td>
</tr>
</tbody>
</table>

detectors built based on such training data cannot detect those intrusions. One class SVM does not need pure normal data, it can be normal data with or without intrusion data.

Another disadvantage of STIDE and Markov Chain anomaly detectors is the selection of a threshold. The threshold is critical in these two approaches, however, there is no efficient and effective method for selecting a proper threshold. One class SVM needs a threshold too, and the threshold is trade-off between the radius of the hypersphere and the number of training examples that it can hold. The threshold for one class SVM is not as critical as for original STIDE and Markov Chain anomaly detectors; as long as it is in some reasonable range, the result would not deviate significantly. We determine the threshold empirically.

5.4 Summary

We experiment on combining STIDE and Markov Chain kernels with one class SVM to improve the classification result. The experiments provide strong evidence that STIDE kernels and Markov Chain kernels, in conjunction with one class SVMs, could offer an accurate, effective and efficient learning method, which does not require labeled data, for anomaly intrusion detection.

A threshold is needed for one class SVM experiments. A technique to determine the proper threshold value can be an interesting research topic.
CHAPTER 6. Specification assisted anomaly detection

This chapter examines how to take advantage of the strengths of the specification based intrusion detection and anomaly intrusion detection, and overcome their weaknesses by integrating these two intrusion detection approaches together. We hypothesize that such integration can enhance the performance of anomaly detectors by improving detection rate and reducing the false alarm rate to achieve a better overall accuracy rate.

For the anomaly detection approach, we continue to experiment on Support Vector Machine (SVM). We first conduct a preliminary experiment of using a Finite State Machine (FSM) derived from program source code. We use FSM as a filter for the SVM. Subsequently, we experimented on integrating Extended Finite State Automata (EFSA), which is a specification based intrusion detection method developed at State University of New York, Stony Brook [60], with SVM.

6.1 Introduction

Specification based intrusion detection systems define specifications developed from system design or protocol. When the system behaviors deviate from the specifications, intrusion alarms are raised. The specification based intrusion detection systems have zero or very low false alarm rates since all normal behaviors (seen or unseen) comply to the specifications. However, specification based intrusion detection systems often have false positives (undetected intrusions). This is because it is hard for the specification approach to describe only the normal behaviors and excludes all the abnormal behaviors. Generally speaking, specifications describe a superset of normal behaviors.

An ideal specification should specify only normal behaviors and exclude all illegitimate be-
behaviors. However, development of specifications are mostly done manually, and it requires a lot of human effort. A general specification takes less effort, but it can not exclude all the illegitimate behaviors. A precise specification excludes almost all illegitimate behaviors but takes tremendous effort. Uppuluri et al. describe a procedure of gradually refining specifications from general to specific [68]. The trade-off between human effort spent on refining specifications and false negatives caused by simplification of specifications can be carefully considered and decided by the developers of the specification based intrusion detection system.

On the other hand, anomaly intrusion detection systems have high false alarm rates because of the existence of unseen normal behaviors. Anomaly detection systems develop normality descriptions from observed behaviors. The system behaviors are compared to the normality descriptions and deviations are reported as anomalies. Statistical methods and machine learning algorithms are two main approaches used to learn and generalize observed normal behaviors and to generate normal patterns. However, there are still many normal behaviors that can not be recognized as normal for many reasons, such as limited generalization capability of machine learning methods and the changing environment. Thus, the normality profile of an anomaly detection system often depicts a subset of normal behaviors or shifts from normal behaviors, which cause high false alarm rates.

Filter approach uses a specification based intrusion detector as a filter to assist anomaly detector. Any behaviors that are illegitimate according to the specifications are classified as anomalies. Then the anomaly detector is applied to classify the remaining data.

Feature selection approach is proposed to select essential features using specifications. Feature selection is critical for anomaly detection. The reasons are irrelevant features cause overfitting problems and too many features cause the performance of anomaly detectors to be inefficient. Although SVM theoretically does not need feature selection, reduction of features using some domain knowledge still provides efficiency and proper generalization capability. The features described in specifications are relevant and important; thus, using the specification based approach, a feature set is formed to only include relevant and important features.

A combined approach, which combines the filter approach and the feature selection approach, is employed to achieve better accuracy rates and select relevant features.
To prove these ideas, we design the first experiment to use a Finite State Machine (FSM) which is derived from source code to assist in anomaly detection. This FSM is used as a filter to first classify all the behaviors not accepted by the FSM as abnormal. Features generated from the FSM are used by SVM. Synthetic data are generated for training and testing.

Then, the specifications based method described in Sekar and Uppuluri's work [68] are integrated with SVM for intrusion detection. The specification is first automatically translated into Extended Finite State Automata (EFSA). Then the EFSA is used as a filter to first classify all the behaviors not accepted by the EFSA as abnormal. Subsequently SVM is employed to classify the data that is accepted by the EFSA, using the features found by the EFSA.

6.2 Experiment with FSM filter based SVM anomaly detector

There are many ways to specify program behaviors. One method is to use a high level language to specify program behaviors. The high level language for describing programs are called Domain Specific Language (DSL) [69]. Some well-known DSL include ADL (Assertion Definition Language) for programming interfaces [11], and EASEL, a language for building end-user applications [18]. BSML, a DSL developed at State University of New York, Stony Brook, is for describing security related behaviors for applications [68]. It does not describe the whole behavior model of a program, but only security related behaviors.

Another type of approaches to describe program behaviors is graph based, such as Colored Petri Nets, Program Flow Graphs and Finite State Machines. Colored Petri Nets are powerful for describing complicated, distributed cooperation behaviors. It is widely used in describing communication protocols and distributed misuse systems [28]. Helmer et al. proposed to use CPN to model distributed intrusion detection [22]. A Program Flow Graph is conventionally used to design and test a program. A Finite State Machine is another method to model program behaviors, and it is widely used for verifying programs. We use FSM to model program behaviors since it can be generated from programs, program traces and program binaries.

Sekar et al. have performed research on generating a FSM from program execution traces
The advantages of this method are program traces are easy to obtain, and no source code is needed in this method. The problem is that the execution traces can not represent all the program behaviors. Thus, training data must cover sufficient normal cases to obtain an accurate automaton.

Wagner et al. at UC Berkeley performed research on statically analyze the source and generate the model [73]. The execution of a program is described using a pushdown automata. The pushdown automata stores the return addresses of all unfinished function calls, with a pointer on the top, which points to the next statement to execute. The control flow of the program decides the transition of the pushdown automata.

Wagner et al. also model the security property of a program as a FSM. Then model checking is used to determine whether certain states that represent security violations in FSM can be reached from the pushdown automata.

The other method to build FSM is to extract FSM models from program binaries. Feng et al. designed a tool to build FSM from program binaries [16].

In our experiment, we use a method that is close to Wagner’s approach to build FSMS from program sources. The line numbers are regarded as states of the FSM, so that FSM can be built based on analyzing program flow control in a program’s source code. We present the design in the next section.

The main goals of combining FSM with SVM is to:

First, use specifications of programs to assist anomaly intrusion detection and get high detection rates and low false alarm rates. As we have explained in a previous section, the main purpose of using specification to assist anomaly detection is to join the strength of the two different types of intrusion detection approaches and alleviate their weaknesses.

Second, specification can assist to reduce irrelevant features. There is a feature space explosion if we use all the size-k system call sequences as features for SVM when k gets larger. We tentatively use all the size-k system call sequences existing in the program trace logs as features. However, this scheme has the problem that features need to be recalculated for each new set of training and testing data. This is very inefficient. We aim to select a manageable set of features using FSM for a SVM learning algorithm.
6.2.1 Design

6.2.1.1 Automatically generate FSM from program source code

It is observed that "a compromised application would not cause much harm unless it interacts with the underlying operating system" [73]. Most of the interactions between applications and the operating system are through system calls. Thus, many intrusion detection approaches monitor system call traces, such as STIDE and Markov Chain based anomaly detectors. Our approach also monitors system call traces of the program execution, and a FSM model of the program is constructed from the program source code to describe the legitimate system call traces of the program.

Since we only aim to extract the model for system call traces of a program, we ignore the variables, data flows and arithmetic operations and only watch for system calls and program flows. We include a mapping file which describes the mapping relationship from C functions to operating system calls. Each C function may map to a sequence of system calls, or even a FSM.

We analyze three kinds of flow controls:

**If - then statement:** To construct a FSM piece of an if-then statement, we include a starting state \( S_0 \) and an ending state \( S_1 \). \( S_0 \) transits to the starting state of the FSM constructed from the program piece between "if" and "then", with a null system call \( e \) on the transition. The ending state of the FSM constructed from the program piece between "if" and "then" transits to \( S_1 \), also with a null system call \( e \) on the transition.

**For statement and while statement:** To construct a FSM piece for a loop, suppose the starting state of the piece of program inside the loop is \( S_0 \) and the ending state is \( S_1 \). For simplicity, the transition from \( S_1 \) to \( S_0 \) is added with a null system call \( e \) on the transition, and, the number of loops is neglected. The FSM constructed in this way does not keep track of the number of loops, thus it defines a superset of specification of behaviors of a program.

**Function call:** We treat a function call as an independent piece of code, and construct a
1. **S0,**
2. **while (...) {**
3. **S1,**
4. **if (...) S2;**
5. **else S3;**
6. **if (S4)...;**
7. **else S2;**
8. **S5:**
9. **}**
10. S3;
11. S4;

![Figure 6.1 Source code of a piece of program](image)

![Figure 6.2 FSM translated from the code](image)

FSM from the function call definition. The previous state of the program is connected to the starting state of the FSM with a null system call $\epsilon$ and the ending state of the FSM is connected to the next state in the program with a null system call $\epsilon$.

Figure 6.1 illustrates a piece of source code and Figure 6.2 illustrates the FSM automatically generated for the source code.

The system call traces that are rejected by the FSM are illegitimate. However, the system call traces accepted by the FSM may not be a legitimate trace for this program because we do not calculate the control variables and consider all the condition constraints (such as loop counters and conditions of if statement). There are some paths that may be impossible to reach but we still include it in the FSM since we neglect the control variables in if-statement and loop statements. Nevertheless, this does not impact using the FSM as a filter to discover the traces that are unquestionably illegitimate or extract all possible subsequences of an arbitrary size. Some traces that are covered by the FSM can be abnormal, however, all the traces that are not covered by the FSM can not be normal.
6.2.1.2 Reduce features using FSM

Given the FSM model for a program, all possible system call sequences of arbitrary size \( k \) can be extracted. This can be done by enumerating all possible \( k \)-size sequences starting from each state.

This can solve the feature space explosion problem in an SVM anomaly detector. With the sliding window size \( k \) increasing, the number of features (all system call sequences of size \( k \)) increases exponentially. The time and space needed by the SVM anomaly detector grow exponentially and become unmanageable if all the size \( k \) system call sequences are all used as features. Using a FSM to derive all possible system call sequences of size \( k \) for a program, and using only the possible system call sequences of size \( k \) derived from the FSM as features for SVM learning, the feature space explosion problem can be solved.

6.2.1.3 Apply filter policy to SVM

The overall infrastructure of the hybrid intrusion detection, which is composed of a specification based intrusion detection module and an anomaly intrusion detection module. The FSM is the specification module, and it is generated from program source code. The FSM is used as a filter to classify all illegitimate behaviors (system call sequences that are not accepted by the FSM) as abnormal. The SVM is then applied to classify the remaining data.

6.2.1.4 Advantages of our design

Using a FSM to assist the SVM anomaly detectors, the following advantages can be achieved.

First, the detection rate will be increased. Any sequences not accepted by the FSM are classified as anomalies. This rules out the error SVM may encounter to classify such a sequence as normal.

Second, the false alarm rate will be decreased. The FSM approach does not introduce any false alarms to the SVM anomaly detector since this method does not classify any possible system call sequences as abnormal if the FSM is correctly built. Moreover, since many anoma-
lies are identified by FSM, the remaining anomalies become a smaller part, the threshold for
an SVM can be reduced, thus rendering a smaller false alarm rate.

Third, the time and space required for SVM learning is reduced tremendously using the
feature reduction based on FSMs built from programs.

Fourth, it does not add much manual effort. With the automatic building of a FSM from
a program's source code, the FSM filter can be integrated easily with the machine learning
approach. It does not involve much human effort like other specification methods.

6.2.2 Experiment and result

We use the benchmarking method introduced by Maxion et al. to test our hypothesis of
the advantages from the hybrid intrusion detection design [42]. In general, two kinds of data
are generated: training data and testing data with anomalies.

The normal data is generated from simulated execution of the program source code. The
data with anomalies is generated by inserting anomalies into normal data. Foreign symbol
and foreign system call sequences are system calls or system call sequences that do not exist
in any normal execution traces. Table 6.1 illustrates the comparison of FSM, SVM and SVM
with FSM as a filter and feature selection tool.

The new FSM guided anomaly detectors are observed to have a better accuracy rate
than the original SVM anomaly detectors. Using FSM to assist SVM anomaly detectors, the
detection rates increase since all the illegitimate behaviors are classified as anomalies by the
FSM filter, thus they can not be misclassified by normal as the SVM anomaly detectors.

Using FSM to do feature selection, only relevant features are selected, the feature set is
reduced to a manageable set with the detection rate and false alarm rate unchanged.

6.3 EFSA with SVM

Uppuluri's specification based intrusion detection system uses BMSL to define specifica-
tions of system and programs [68]. The specifications written in BMSL are then compiled into
EFSA, which extend standard FSA to be able to assign or examine values of a finite set of
Table 6.1 Comparison of FSM, SVM and SVM with FSM filter and feature selection

<table>
<thead>
<tr>
<th>Detector (win size=2, Stide and Markov Chain kernel)</th>
<th>Accuracy rate</th>
<th>Detection rate</th>
<th>False alarm rate</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSM</td>
<td>88.1%</td>
<td>86.6%</td>
<td>13.4%</td>
<td>0.71</td>
</tr>
<tr>
<td>Stide kernel two class (with FSM filter)</td>
<td>94.6%</td>
<td>83.5%</td>
<td>2.3%</td>
<td>0.87</td>
</tr>
<tr>
<td>(with FSM feature selection)</td>
<td>94.6%</td>
<td>83.5%</td>
<td>2.3%</td>
<td>0.87</td>
</tr>
<tr>
<td>(with FSM filter, feature selection)</td>
<td>96.6%</td>
<td>91.5%</td>
<td>2.1%</td>
<td>0.88</td>
</tr>
<tr>
<td>Markov Chain kernel two class (with FSM filter)</td>
<td>96.9%</td>
<td>88.0%</td>
<td>2.2%</td>
<td>0.91</td>
</tr>
<tr>
<td>(with FSM feature selection)</td>
<td>96.9%</td>
<td>88.0%</td>
<td>2.2%</td>
<td>0.91</td>
</tr>
<tr>
<td>(with FSM filter, feature selection)</td>
<td>97.4%</td>
<td>94.9%</td>
<td>2.1%</td>
<td>0.93</td>
</tr>
<tr>
<td>Stide kernel, one class (with FSM filter)</td>
<td>89.2%</td>
<td>80.5%</td>
<td>5.4%</td>
<td>0.81</td>
</tr>
<tr>
<td>(with FSM feature selection)</td>
<td>92.1%</td>
<td>84.5%</td>
<td>3.5%</td>
<td>0.84</td>
</tr>
<tr>
<td>(with FSM filter, feature selection)</td>
<td>92.1%</td>
<td>84.5%</td>
<td>3.5%</td>
<td>0.84</td>
</tr>
<tr>
<td>Markov Chain kernel, one class (with FSM filter)</td>
<td>92.7%</td>
<td>85.2%</td>
<td>3.9%</td>
<td>0.85</td>
</tr>
<tr>
<td>(with FSM feature selection)</td>
<td>92.7%</td>
<td>85.2%</td>
<td>3.9%</td>
<td>0.85</td>
</tr>
<tr>
<td>(with FSM filter, feature selection)</td>
<td>94.2%</td>
<td>89.9%</td>
<td>2.6%</td>
<td>0.88</td>
</tr>
</tbody>
</table>

variables. EFSA can be simulated efficiently using audit data as input and exits when audit data is not accepted (an anomaly is detected).

We employ EFSA in order to perform feature selection to assist anomaly detection using SVM.

6.3.1 Feature selection approach

Selecting relevant features can improve detection rates and decrease false alarm rates. Our previous methods for generating features enable us to obtain all possible subsequences with size $k$ as features, or generate subsequences with size $k$ that appear in training and testing data. The first method will cause feature space explosion when $k$ increases, and tremendous time and space are needed for processing the features. Using the second method, we need to peek into the testing data to find out all the features needed; also, features need to be rebuilt for each set of training and test data. Using EFSA to generate relevant features, the feature set is reduced to a manageable size for SVM anomaly detector with excellent results. This hypothesis is shown to be true from our experimental results.
Time and space required for SVM learning are reduced tremendously using the feature reduction based on EFSA built from program. Furthermore, it does not add much manual efforts.

6.3.2 Filter approach

We also employ EFSA as a filter for the SVM anomaly detector. EFSA is applied to the data first to find out intrusions, then the remaining data is fed into the SVM for anomaly detection.

Using EFSA as a filter, a higher detection rate can be achieved since some attacks that violate the specification indicated by EFSA, but is not detectable by SVM, are now being detected. The EFSA will not introduce any false alarms as EFSA represents specification of program or system, which depicts a superset of normal behaviors. Since the number of anomalies remained after applying EFSA filter are less than before, there is a high probability that SVM can perform better regarding to false alarm rate and detection rate.

6.3.3 Combined approach

We combine the specification based feature selection and specification based filter together with the SVM based anomaly detectors. Figure 6.3 shows the training phase of the combined anomaly detector, and Figure 6.4 shows the testing phase of the combined anomaly detector. In the training phase, BSM event stream is fed into the specification based filter first, the data that are detected as anomalies by the specification based method are filtered out. The remaining data are used to train the SVM based anomaly detector. The features that are selected by the specification as described in Section 6.3.2 are also used as the features of the SVM based anomaly detector. A model composed of support vectors are learned in the training phase. In the testing phase, BSM event stream is fed into the specification based filter first, some anomalies that can be detected by the specification based method are reported. The remaining data are fed into the SVM with the trained support vectors model, the SVM report alarm each time it classifies a data record as anomaly in the remaining data set.
Figure 6.3 Training phase of the hybrid anomaly detector

Figure 6.4 Testing phase of the hybrid anomaly detector
6.3.4 Experiment and result

Using DARPA Lincoln Lab data, Table 6.2 illustrates the comparison of SVM and SVM with EFSA as a feature selection tool and a filter on the FTP processes. In such data set, there are several attacks, such as ftp-write, guest-ftp, ftp-warezserver, ftp-warezclient. Uppuluri’s specification based anomaly detection system is augmented with misuse constraints, thus a 100 percent detection rate can be achieved. By taking off a misuse constraint which will detect guest-ftp attack, a detection rate of 76.5% can be achieved by the specification based intrusion detection system.

A FTP process is simple and only has a small size of system call set (about 30 system calls). We use a window size 2, and the EFSA feature selection procedure generates around 60 features. This reduces the feature set from 900 (i.e., $30^2$) to 60 features, i.e., a reduction of 93.3%. From Table 6.2, we can see the detection rate and false alarm rate remain the same after the feature selection, both for two class and one class SVMs.

By using EFSA as a filter, the detection rates increase, both for two class and one class SVMs. We also observed, the detection rate of one class SVM combined with a EFSA filter, has improved over the detection rate of the original one class SVM. Furthermore, the false alarm rates decrease, for both two class and one class SVM anomaly detectors.

Table 6.3 illustrate the comparison of SVM and SVM with EFSA as a feature selection tool and a filter on multiple processes in the DARPA data. There are many processes in the DARPA data with attacks, such as fdformat, eject and ffbconfig. In the specification based intrusion detection system, there is a generic specification for detecting attacks on these processes, which specifies regular rules in general and can detect attacks such as buffer overflow attacks. With appropriate misuse constraints, this specification based system can achieve 100% detection rate on these processes in the DARPA data. We take off the misuse constraints in the generic specification and the detection rate of the specification based intrusion detection system is 73.5% for these processes in the DARPA Lincoln Lab data.

Using the EFSA to do feature selection, a feature set with size 120 is generated, this is much smaller than the original feature set, which is 2500 (i.e. $50^2$ where 50 is the number of
system calls and 2 is the window size). The experiment demonstrate, there is no substantial changes in detection rate and false alarm rate.

Using the generic specification as a filter, the detection rates on multiple processes are improved for both two class and one class SVM anomaly detectors. Also, the false alarm rates decrease dramatically.

The EFSA filtered SVM anomaly detector performs better than both EFSA and SVM anomaly detectors.

- It achieves higher detection rate, lower false alarm rate.
- The feature selection problem of the SVM anomaly detector is solved.
- The new hybrid anomaly detector is also robust in the way that it has a very good accuracy rate for data with any percentage of anomalies.
- It becomes less sensitive to the threshold for one class SVM.
Table 6.3 Comparison of EFSA, SVM and ESFA guided SVM on multiple processes in the DARPA Lincoln Lab data

<table>
<thead>
<tr>
<th>Detector (win size=2 for STIDE and Markov Chain kernel)</th>
<th>Accuracy</th>
<th>Detection rate</th>
<th>False alarm rate</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFSA</td>
<td>100%</td>
<td>100%</td>
<td>0%</td>
<td>1</td>
</tr>
<tr>
<td>EFSA (generic, without misuse rules)</td>
<td>84.1%</td>
<td>73.5%</td>
<td>0%</td>
<td>0.83</td>
</tr>
<tr>
<td>Stide kernel, two class SVM</td>
<td>90.4%</td>
<td>89.3%</td>
<td>5.9%</td>
<td>0.70</td>
</tr>
<tr>
<td>(with EFSA feature selection)</td>
<td>90.4%</td>
<td>89.3%</td>
<td>5.9%</td>
<td>0.70</td>
</tr>
<tr>
<td>(with EFSA filter)</td>
<td>96.2%</td>
<td>94.3%</td>
<td>4.1%</td>
<td>0.95</td>
</tr>
<tr>
<td>(with EFSA filter, feature selection)</td>
<td>96.2%</td>
<td>94.3%</td>
<td>4.1%</td>
<td>0.95</td>
</tr>
<tr>
<td>Markov Chain kernel, two class SVM</td>
<td>93.9%</td>
<td>94.5%</td>
<td>3.5%</td>
<td>0.81</td>
</tr>
<tr>
<td>(with EFSA feature selection)</td>
<td>93.9%</td>
<td>94.5%</td>
<td>3.5%</td>
<td>0.81</td>
</tr>
<tr>
<td>(with EFSA filter)</td>
<td>96.9%</td>
<td>95.9%</td>
<td>2.1%</td>
<td>0.96</td>
</tr>
<tr>
<td>(with EFSA filter, feature selection)</td>
<td>96.9%</td>
<td>95.9%</td>
<td>2.1%</td>
<td>0.96</td>
</tr>
<tr>
<td>Stide kernel, one class SVM</td>
<td>88.7%</td>
<td>87.3%</td>
<td>9.6%</td>
<td>0.69</td>
</tr>
<tr>
<td>(with EFSA feature selection)</td>
<td>88.6%</td>
<td>87.1%</td>
<td>9.6%</td>
<td>0.68</td>
</tr>
<tr>
<td>(with EFSA filter)</td>
<td>92.9%</td>
<td>91.1%</td>
<td>4.0%</td>
<td>0.92</td>
</tr>
<tr>
<td>(with EFSA filter, feature selection)</td>
<td>92.8%</td>
<td>91.0%</td>
<td>4.0%</td>
<td>0.92</td>
</tr>
<tr>
<td>Markov Chain kernel, one class SVM</td>
<td>93.2%</td>
<td>91.1%</td>
<td>6.7%</td>
<td>0.81</td>
</tr>
<tr>
<td>(with EFSA feature selection)</td>
<td>93.1%</td>
<td>91.0%</td>
<td>6.7%</td>
<td>0.81</td>
</tr>
<tr>
<td>(with EFSA filter)</td>
<td>93.9%</td>
<td>92.5%</td>
<td>4.3%</td>
<td>0.93</td>
</tr>
<tr>
<td>(with EFSA filter, feature selection)</td>
<td>93.8%</td>
<td>92.4%</td>
<td>4.3%</td>
<td>0.93</td>
</tr>
</tbody>
</table>

6.4 Summary

By integrating specification based methods with anomaly intrusion detection, the strengths of two methods are combined together. By using the specification based method as a filter, some sequences with subsequences that do not exist in the space of all possible subsequences of the program are filtered out. This will definitely improve the detection rate of the original SVM anomaly detectors.

Also, lower false alarm rates are achieved and the SVM anomaly detector becomes more robust and less sensitive to selection of the threshold value. Furthermore, the specifications can assist the feature selection procedure of the anomaly detection method, which used to be a big problem for machine learning based anomaly detection.
CHAPTER 7. Intrusion detection software agents

Our hybrid Mobile Agents for Intrusion Detection System (MAIDS) includes both misuse intrusion detection agents and anomaly intrusion detection agents. Anomaly intrusion detection agents are automatically built from training data and launched to monitored machines for detecting anomalies; while misuse intrusion detection agents are automatically generated from SFT descriptions of intrusions, using CPN as a design template for intrusion detection system, and launched to monitored hosts to detect known attacks.

In this chapter, we first introduce automatic generation of misuse intrusion detection agents and then introduce automatic generation of anomaly intrusion detection agents in our agent based hybrid intrusion detection system.

7.1 Automatic generation of misuse intrusion detection agents

Before implementing any complicated system it is essential to come up with a good design for the system. This is very important for an IDS where a faulty implementation resulting from a bad design might give the end user a false sense of security. Currently, many IDSs are designed using ad hoc methods, and they are prone to design and implementation errors. Formal models, Colored Petri Net (CPN) and Software Fault Tree (SFT), are used in our project as the theoretical underpinnings for misuse intrusion detection design, implementation and verification [22].

7.1.1 Introduction

CPN has been extensively used to model complex and distributed systems [28]. CPN is proposed to model IDS design because CPN can be used efficiently to model the gathering,
classification and correlation activities of IDSs [74]. There also exist CPN tools (such as, Design/CPN [13]) which allow us to simulate attacks in order to validate the correctness of an IDS. Using these CPN tools, we can assure whether a given CPN representation of an IDS design is correct or not.

Even though CPNs have proven to be powerful in modeling IDSs, they are complicated. A designer must not only have adequate knowledge in the intrusion domain, but also have a good command of CPN. Tremendous effort is needed to manually develop a CPN for an IDS design. Furthermore, a minor mistake can cause a faulty CPN design, which will cause the software developed from this CPN design to fail. In addition, if we manually design a CPN for an IDS, every time a new kind of intrusion emerges, much human effort is needed to redesign the CPN to detect the new intrusion type.

SFT models are very powerful in describing how faults propagate through a system by modeling relationships between faults and associated events in the software system. SFT is proposed to model an intrusion by representing the combination and sequence of events which contribute to the intrusion [24]. The root node at the top-level of a SFT represents a certain kind of intrusion. The SFT model can describe a coordinated and complicated intrusion very well by identifying the related events in the intrusion. By modeling how intrusions occur in a system, SFT can assist in determining the requirements for an IDS [24]. SFT is not only powerful in describing intrusions, but also very simple. SFT is easy to design and understand, and can be easily learned and used by system administrators to model intrusions.

SFT can model intrusions and help in requirement analysis for an IDS, however, they are not suitable for IDS design. The reason is that SFT can only describe the events and sequences of events that cause the intrusions in a system, and it does not describe the procedure to detect and correlate these events. To model an IDS design, an execution model like CPN is needed. There are also no verification and simulation tools for SFT like what are provided for CPN.

Thus, we propose a software approach to derive CPN design templates for IDS from SFT descriptions of intrusions automatically. Together with the compiler previously developed in our research group for translating from CPN design to software intrusion detection agents [74], software code for intrusion detection agents can be generated automatically from the SFT
description of intrusions. The entire process comprises of primarily three stages:

1. A high-level description of intrusions is created using Software Fault Trees (SFTs).

2. This description is then translated into CPNs, which serves as the IDS design specification, using the software approach presented in this chapter. The CPN design for IDSs can be verified using formal tools such as Design/CPN.

3. The CPNs are then translated into the actual implementation of IDS software agents using the compiler previously developed by our research group [74].

The unique features of this work are the ease with which software intrusion detection agents can be automatically generated, and the usage of formal methods for assuring that the IDS developed satisfies a set of high-level requirements (SFT specification of intrusions). In the following sections, we first introduce how to convert from extended SFT to CPN, then introduce how to convert from CPN to intrusion detection software agents.

### 7.1.2 Conversion from extended SFT to CPN

Previous research work attempted to do conversion from basic FTs to PNs. A basic FT is a static model, which means it cannot represent sequential events. The FT model we use in our system is extended with time constraints, and these temporal features make it more powerful than basic FT because it can describe sequential events. The CPN we use for modeling intrusion detection is a high level PN which combines an ordinary PN with the strength of high level languages, by equipping the tokens with different attached data types, called token colors [28]. Compared to PNs, CPN are more compact. Our goal of this section is to show how to convert the extended SFT model to a CPN model for automatic generation of misuse intrusion detection agents.

#### 7.1.2.1 Relationships between extended SFT to CPN

The relationships between the basic elements in extended SFT, such as leaf nodes, AND nodes and OR nodes, and their corresponding CPN segments are as following:
Leaf Nodes. There are two types of leaf nodes in an extended SFT: basic event nodes and constraint nodes. Each basic event class in a SFT corresponds to a token source place in the CPN, and represents the kind of basic event in the system that must be monitored. For each event class, there is also a corresponding token color defined.

The constraint nodes in an extended SFT are also leaf nodes, but they are different from basic event nodes. They must be processed with other nodes, typically the AND gates they connect to, to develop accurate CPN templates for intrusion detection.

AND nodes without constraints. An unconstrained AND node in a SFT corresponds to a transition and outgoing place pair in the CPN. Each incoming arc in the CPN comes from either a source place or an outgoing place of another transition.

AND nodes with temporal constraints. Nodes connected to an AND node in an extended SFT may have an attached constraint node that requires the nodes to become true according to certain time constraints.

To support ordering, CPN tokens are required to contain timestamps or sequence numbers. If event A occurs before event B, the timestamp in the token representing event A must be less than the timestamp in the token representing event B. Likewise, if event A occurs before event B, the sequence number in the token representing event A must also be less than the sequence number in the token representing event B. The clock time of each machine monitored by the IDS must be synchronized; and the sequence numbers are maintained per context. No comparison may be made between sequence numbers across contexts.

OR Nodes. An OR node in a SFT corresponds to transitions with a single outgoing place in the CPN. There is no time constraint with OR nodes, because only one incoming event needs to be true to make an OR node true.

For a formal definition of extended SFT for intrusion detection, please refer Appendix A

7.1.3 Translation algorithm implementation

Based on the mapping relationship between SFT modeling intrusion and CPN modeling intrusion detection system described above, an algorithm is implemented to perform the translation.
For both the SFT and CPN representations, we use eXtensible Markup Language (XML) [5]. XML is suitable for representing information in a tree structure and is an emerging standard for representing exchangeable knowledge in many fields. Design/CPN [13], which we use to design and validate CPN for IDS, supports XML representation of CPN. For SFT, some popular SFT design software, such as Galileo [10], is planning to support XML representation of SFT. In our experiment, we use a normal XML design software to design SFT XML. We designed a Document Type Definition (DTD) for the extended SFT, and the DTD for the CPN is specified by the Design/CPN tool.

XML representation allows for convenient searching for information in the tree structure of nodes. The eXtensible Stylesheet Language (XSL) is often used to perform the conversions between XML documents [1]. Using XSL to retrieve information in one XML format and then reorganize and represent it in another format of XML file is very straightforward. The W3C Recommendation XSL Transformations (XSLT) Version 2.0 describes how to write XSL to transform one XML file to another XML file [72]. Since the conversation involves mainly tree nodes manipulation, including retrieving information from SFT tree nodes and generating new tree node of the CPN based on retrieved information. So we use XML for both SFT and CPN representations, and developed the XSL translator based on algorithms according to XSLT protocol [72].

The CPN definition file, which specifies variables and type definitions for a CPN file, also need to be generated. For type definitions, they are the same for all CPNs. So they are put into each definition file without change. For variable definition, the variables used in tokens, such as ‘time_eventA’, ‘seq_eventB’, are put into a CPN definition file when the CPN file is generated.

The experiment of using XSL to implement the translation algorithm shows the conversion algorithm is convenient to write and is flexible to modify. Using XSL, the effort to modify the converter can be minimized when we need to translate from a different SFT XML format which is generated from a different SFT tool, or to translate to a different CPN XML format which is required by a different CPN tool.

In the Appendix B, the examples of the SFT modeling intrusions and CPN modeling
intrusion detection design templates are provided.

7.1.4 Conversion from CPN design template to misuse intrusion detection agents

We have designed a compiler to translate a CPN template for intrusion detection system to software intrusion detection agents [74]. The MAIDS software agent system closely follows the system design as encapsulated by CPN representation. Leaf places in CPN translates to data source agents which reside on each monitored machine, and is responsible for creating tokens and holding them until they are picked up by mobile agents. A data source agent is in communication with a local database that is being populated by some local data sources, such as system and network audit files. A leaf transition, which has leaf places among its inputs, translates to a mobile agent. A leaf transition agent travels through the computer network, picks up tokens and correlates events at every site that it visits. The root place in the CPN corresponds to the console of the administrator who is monitoring the system.

The main design goal of the compiler is to preserve the correctness of the CPN while translating it into software agent code. It also has the goals of flexibility to allow integration of different CPN representations and generalities, which allows no restriction of the kind of intrusions CPN templates are for. Another important goal is to involve minimal manual intervention in the translation procedure.

Using the XSL translator, which can translate from a SFT modeling an intrusion to CPN modeling intrusion detection system, and the compiler which can translate the CPN to code for software agents, misuse intrusion detection software agents can be generated and launched automatically.

7.2 Automatic generation of anomaly intrusion detection agents

A serious and difficult challenge of using data mining approaches in anomaly intrusion detection is that it uses a large volume of audit data files, which usually disperses on several machines, to compute the profiles of users, programs and systems for attack discovery. There-
fore we propose to use intelligent software agents which can move between machines to collect
data, learn models and detect anomalies using the learned models.

In this chapter, the SVM anomaly detectors are implemented as software intrusion detec-
tion agents. Also, an anomaly intrusion detection system, which composes of a central console
and several monitored hosts, is implemented. From the central console, anomaly intrusion de-
tection agents are launched and moved to monitored hosts. These intrusion detection software
agents periodically report discovered anomalies to the central console.

7.2.1 Design

This system aims to have the anomaly detection mobile agents to learn from distributed
data sources on the network. There are two design goals:

First, data reduction, i.e., audit data from one site, can be reduced and moved to other
sites and combined with data on other sites or combined with newly generated data to generate
models for anomaly detection. There is need for data reduction because audit data files for
intrusion detection are normally very large and it is inconvenient to keep these data for future
use or move these data to other sites to learn a combined model. Using data reduction and
mobile agent technology, data can be reduced and combined with data on other sites or new
data to learn a more complete model. This work implements SVM intelligent agents to reduce
data, carry reduced data from one site to the other and learned a model from the combined
data from both sites.

Second, share of profiles, i.e., the model learned from one site can be distributed to another
site for classification. Similar with other machine learning algorithms, SVM learns a model
from a training data set, then the model can be used by a SVM classifier to classify the
testing data. The models learned by the SVM anomaly detectors are support vectors. This
work implements SVM intelligent software agents which are capable of learning a profile from
the training data on one site and then carry the profile to another site for classifying data on
that site.
7.2.1.1 Mobile agent platform

Voyager agent platform, version 4.0, from Recursion Software, is used because of its flexibil­ity and platform independency [54]. Mobile code can be generated in this mobile agent platform. An agent in Voyager is an instance of a Java class, which may be created either locally or remotely. There is support for two types of code mobility in Voyager. One agent can move another agent between hosts, or an agent can request its own migration. In either case this is accomplished by the local agent server instantiating a new object of the agent’s class at the destination server, remotely invoking a specified method, and destroying the local agent object. Voyager is pure Java based, thus it is platform independent.

We use Voyager agent platform to generate and move intelligent agents around a network, so every host monitored in the network by this system must be installed with this mobile agent platform.

7.2.1.2 Mobile intelligent agents design

There are two kinds of mobile agents in this system:

- **Model Learning Agents** which can move between machines, and learn models from training data on these machines (labeled training data is used for two class SVM and unlabeled training data is used for one class SVM). The models learned can be distributed to any monitored hosts for classification. Model Learning Agents can also carry out data reduction task, i.e., reduce data on one monitored machine, and carry them to another machine and combine them with data on other machine for learning a more complete model.

- **Anomaly Detection Agents** which are sent to monitored hosts classifying audit data on monitored machines using a learned model. The results are returned to the console.

The Model Learning Agents are able to move between central console and monitored hosts for model learning, or move between monitored hosts for combining reduced data. The Anomaly Detection Agents are able to move from central console to other monitored machines for data classification.
The working procedure of a support vector Model Learning Agent is as Algorithm 1. For each host, it moves to the host and extract feature vectors for SVM learning according to the method we introduced in previous chapters and appends the feature vectors to its internal data array. If no more host in the monitored host list, it learns a SVM model using the class `svm_train` provided by libSVM [9] based on the combined feature vectors, save the learned model in a string, returns to central console and saves the learned model in a predefined file.

**Algorithm 1 work(hostlist)**

```
data set S = ∅
for each host h in hostlist do
    Move to host h
    Open the training file on host h and extract feature vectors V
    Append the extracted training vectors V to its data set S
    If h is the last one in the hostlist, use the data in its data set S to build a support vector model M.
end for
Move back to its home (the central console)
Save the support vector model M on the central console as a file.
```

The working procedure of an Anomaly Detection Agent is as Algorithm 2. An anomaly Detection Agent first opens the file that contains the support vector model and saves it in its internal variable. Then in every predetermined time interval, it moves to the specified host, extracts the feature vectors from the testing data on the host using the method we discussed in previous chapters, uses the class `svm_predict` provided by libSVM [9] with the support vector model to classify the feature vectors, and returns to the central hosts with the classification results. The name of the testing file contains process information and timestamp, which are presented on the central console when the corresponding testing feature vector is classified as an anomaly.

### 7.2.2 Implementation

#### 7.2.2.1 Data reduction

In original BSM audit data, each BSM record represents a system call, it also include the following fields to describe information about a system call:
Algorithm 2 $work(h, t)$

1. Open the support vector model file on the central console and read in the support vector model to a string variable $M$
2. for every predetermined time interval $t$
   1. Move to the specified host $h$
   2. Open the testing file on host $h$
   3. Use the support vector model $M$ to classify the testing file
   4. Move back to central host and present the classification result on the central console.

**header** Start of record, including length of the record, system call name, etc.

**arg** System call argument value

**exec.args** Exec system call arguments

**exit** Program exit information

**in_addr** Internet address

**path** Path information (path)

**process** Process token information including process id and session id

**return** Status of system call

Since we only use system call name in our anomaly intrusion detection, we discard all the other information and only extract system call names. For each process id, we extract system call traces from the BSM event stream, and then use specified window size to generate system call feature vectors. Once system call feature vectors are generated from the BSM audit data, the BSM audit data can be discarded, thus space for storing the audit data and time for processing the audit data for building a new model are saved.

The generated system call feature vectors, from different sites and different time can be put together for learning a model, assuming these feature vectors are for the same program and have the same feature space (as Figure 7.1).
Figure 7.1 Generation of anomaly intrusion detection profiles

7.2.2.2 Intelligent mobile agents implementation

libSVM has a Java implementation, we use two classes in libSVM `svm.train` and `svm.predict` in Model Learning Agent and Anomaly Detection Agents which execute working procedures as described above [9]. For ModelLearningAgent, the implementation is as follows:

```java
public class ModelLearningAgent implements IAgent, Serializable {
    public ModelLearningAgent(); //constructor
    public work(); //execute the main procedure of the agent
}

public class AnomalyDetectionAgent implements IAgent, Serializable {
    public AnomalyDetectionAgent(); //constructor
    public work(); //execute the main procedure of the agent
}

    ModelLearningAgent::work() {
        ...
        svm_train cl(args); //args is a string array contains training file, model file
        cl.run();
        ...
    }

    AnomalyDetectionAgent::work() {
        ...
        svm_predict prd(args); //args is a string array contains testing file, model file
        ...
    }
```
and output file
prd.run();
;
}

7.2.2.3 Central console

We implement following central console for launch the anomaly detection agents and display alarms:

class Console{
    private String M; // the learned model
    private AnomalyDetectionAgent agents; // the handle of all the Anomaly Detection Agents

    public String learn(String hosts);
    //send a Model Learning Agent to learn a model M from the audit data on a list of monitored machines

    public String detectAnomaly(String modl, String host);
    //send an Anomaly Detection Agent using the model represented in string ‘modl’ to detect anomalies on monitored machine host, return the classification result as a string

    public displayO; // periodically get the alarm information from these Anomaly Detection Agents and present on user interface
}

In the central console, following function is used to initiate an agent:

Factory.create(ModelLearningAgent,‘‘//localhost:8000’’);

and Factory.create(AnomalyDetectionAgent,‘‘//localhost:8000’’);

A configuration file is used to define which machines the agents will learn models from and which machines are monitored for attacks.

Monitored machines can also be added dynamically from the central console. The central console gets the host name which a user wants to monitor, create an Anomaly Detection Agent
Table 7.1 Response time of agents

<table>
<thead>
<tr>
<th>agents</th>
<th>Number of records</th>
<th>Speed (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Learning Agent</td>
<td>132,738</td>
<td>121</td>
</tr>
<tr>
<td>Anomaly Detection Agent</td>
<td>23,218</td>
<td>56</td>
</tr>
</tbody>
</table>

and dispatch it to the monitored machine. The Anomaly Detection Agent will periodically move to monitored machine and classify the testing file on the machine and present the alarm information on the central console.

7.2.3 Experiment and result

In our prototype, three machines is used to set up the demo, two machines are monitored machines and one machine is for analysis, where agents are dispatched from and online reports are generated on the console.

The system efficiency and response time are tested. The Model Learning Agent moves to the monitored host and return to the analyst machine with learned model (support vectors) in about 121 seconds for 132,738 BSM events on 3 machines. The Anomaly Detection Agents move to the monitored host with a learned model and return to the analyst machine with classification results in about 56 seconds for 23,218 BSM events. The machines we are using are Pentium 5 (1.5G Hz CPU). The alarms are displayed as Figure 7.2.

The SVM anomaly detectors learn models based on observed program behaviors (i.e., system call sequences in training data), thus it is relatively stable across machines compared to user or system behaviors. A model learned from one machine can be used on other machines with satisfactory results. Moreover, agents are designed to move from one machine to another machine with reduced data, so data from audit files on several machines can be combined efficiently and a better model can be learned.

7.3 Summary

Our hybrid intrusion detection agent system includes two modules: SFT and CPN based misuse intrusion detection agent module and SVM based anomaly intrusion detection agent module. The misuse intrusion detection agents watch for coordinated attacks on network,
SVM Based Anomaly Intrusion Detection

The host name you want to monitor:

Host monitored:
solomon.cs.iastate.edu
mayoserver.cs.iastate.edu

Alarms from solomon:

Alarms from mayoserver:

Figure 7.2 The interface of the agent based anomaly detector
and the anomaly intrusion detection agents learn a profile of normal systems (in the form of support vectors) from reduced training data on several machines and use the normal profile to find the anomalies on monitored machines.

In this hybrid intrusion detection system, formal methods and formal tools are used in the development of the misuse IDS to reduce design and implementation errors. The translator presented in this chapter translates SFT descriptions of intrusions to XML CPN representations of IDS design templates, which can be recognized and displayed by Design/CPN tool. Subsequently, CPN IDS design template is translated into misuse intrusion detection agents. Our automatic tool provides an easy way of describing sophisticated intrusions using a SFT model, which is easy to learn and design, and updating the intrusion detection agents from the SFT intrusion descriptions.

The anomaly detection agents in this hybrid intrusion detection system are generated and dispatched to the monitored machines on the network for anomaly detection. The SVM based intelligent agents learn models and classify data using models learned from other sites or models learned by combining data from several sites. Using intelligent agent technique, the profile learning and data analysis using the SVM based anomaly detectors are more efficient since large volume training data dispersed over several machines can be processed locally and the processing results can be put together by the agents.
CHAPTER 8. Conclusions and future work

8.1 Conclusions

This dissertation first investigates approaches to improve anomaly detection. To address the false alarm problem caused by incomplete training data and the environmental changes in anomaly detection, kernel methods and Support Vector Machines are introduced. Traditional anomaly detectors, STIDE and Markov Chain anomaly detectors, build normal profiles using observed system call traces, and apply exact matching to identify all unseen patterns as anomalies. Unseen normal behaviors are always classified as anomalies using these two methods. In order to lower false alarm rates, we propose to use a SVM approach, which offers minimized true errors - the errors of predictions on future data.

We also derived two kernels, STIDE kernel and Markov Chain kernel from original STIDE and Markov Chain anomaly detectors for SVM. We proved STIDE and Markov Chain kernel anomaly detectors are kernel based learning methods combined with simple threshold classifiers. The new anomaly detectors, generated by combining STIDE kernel and Markov Chain kernel with SVM, demonstrate better learning ability than the original STIDE and Markov Chain anomaly detectors and shows an excellent accuracy rate for anomaly detection.

Moreover, to lessen the efforts needed for training anomaly detectors, unsupervised learning using one class SVM is also introduced for anomaly detection. One class SVM does not require labeled training data. Preprocessed raw audit data can be fed into one class SVM directly for training purpose. This facilitates the frequent training requests because of the environmental changes and helps reduce false alarms because of environmental changes.

To further reduce false alarm rates and increase detection rates, we also propose to combine specification based intrusion detection approaches with anomaly detection. By using a
specification based intrusion detector as a filter, the accuracy rate of SVM anomaly detectors is further improved. Furthermore, the specifications assist feature selection for the SVM anomaly detectors, solving the feature space explosion problem of SVM anomaly detectors.

Another contribution of this dissertation is automatic generation of misuse and anomaly intrusion detection agents. Software Fault Tree and Colored Petri Net models are used for modeling intrusions and intrusion detections. Misuse intrusion detection agents are automatically generated from SFT specifications of intrusions. Using this method, misuse intrusion detection agents can be generated efficiently with less design and implementation errors. Anomaly detection agents are also automatically generated and launched using profile models learned from audit data from a remote machine.

8.2 Future work

8.2.1 More kernels for intrusion detection

STIDE kernel and Markov Chain kernel are proved to be suitable for anomaly detection. We derived STIDE kernel and Markov Chain kernel from STIDE and Markov Chain anomaly detectors. The kernels are the critical parts in SVM anomaly detectors and decide the accuracy rate of the SVM anomaly detectors. More kernels may be derived for the intrusion detection field. Some kernels which are used and are successful in other fields, such as some string kernels in bioinformatics field, can be tested and adapted to anomaly detection field.

8.2.2 Automatic tuning of the parameter for one class SVM

Currently, the parameter for one class SVM is predetermined. Experiments prove that the parameter is not critical. As long as it is in some reasonable range, the detection rate and false alarm rate will stay the same. However, automatically deciding this parameter for one class SVM anomaly detectors will further reduce human interfere in the anomaly detection, which is always preferable.
APPENDIX A. Formal definition of an extended Software Fault Tree for intrusion detection

An extended SFT is a tree containing of four types of nodes:

1. Basic events, which are grouped into classes. A basic event can be expressed in a parametric form using:

   name_of_class(list of parameters)

   where list of parameters=\{V_i \mid 1 \leq i \leq n\}

   where \(V_i\) is a variable, a value or an expression, which is expressed in the same parametric form.

   For example, TCP_Connection(host1,host2,20,514) is a TCP_Connection class, it represents an TCP connection event that connects from port 20 to port 514. host1 and host2 are variables here.

   FTP(COMMAND,TCP_Connection(host1,host2,port1,21),RETR) is a FTP class, it represent an FTP RETR command that is sent to port 21. ‘host1’, ‘host2’, ‘port1’ are variables.

2. AND gate, corresponding to the standard logic AND gate.

3. OR gate, corresponding to the standard logic OR gate.

4. Constraint nodes, which are input nodes to an AND gate. Each constraint must be evaluated to true for the AND gate to be true.
Three kinds of time constraint nodes: ‘Occurs After’, ‘Immediately After’ and ‘Within Time’ are introduced based on interval temporal logic [2]:

- ‘Occurs After’ is the condition where one event is required to start after another event has started. It can be expressed as:

\[
\text{OccursAfter}(\text{event} A, \text{event} B)
\]

\[
\text{where } \exists \ m \ \text{Starts}(\text{Period}(\text{event} A), m) \land \text{Meets}(m, \text{Period}(\text{event} B))
\]

\[
m: \text{ a period}
\]

\[
\text{period}(x): \text{the period that node } x \text{ is true.}
\]

\[
\text{Starts}(i, j): \text{it is true when periods } i \text{ and } j \text{ begin simultaneously}
\]

\[
\text{Meets}(i, j): \text{it is true when period } i \text{ ends adjacent to the time when period } j \text{ begins}
\]

- ‘Immediately After’ is the condition where an event must occur after another event in the same context with no intervening events, and it can be expressed as:

\[
\text{ImmediatelyAfter}(\text{event} A, \text{event} B)
\]

\[
\text{where } \text{OccursAfter}(\text{event} A, \text{event} B) \land \# n (\text{OccursAfter}(\text{event} A, n) \land \text{OccursAfter}(n, \text{event} B))
\]

\[
n: \text{ an event}
\]

- ‘WithinTime’ is the condition where an event is required to follow another event within some amount of time, and it can be expressed as:

\[
\text{WithinTime}(\text{event} A, \text{event} B, t)
\]

\[
\text{where }
\]

\[
\text{OccursAfter}(\text{event} A, \text{event} B) \land \text{Overlaps}(<\text{StartOf(Period}(\text{event} A)), \text{StartOf(Period}(\text{event} A))+t>
\]
Figure A.1 SFT for FTP bounce attack

t: length of time after event A becomes true event B must become true

StartOf(i): start of time period i

Overlaps(i,j): true when period i overlaps period j

Figure A.1 illustrates an example for the extended SFT, and it is for a FTP bounce attack.
Section B.1 explains the FTP bounce attack and the SFT for it in more detail.
APPENDIX B. Examples of SFT modeling intrusions and CPN modeling intrusion detection design template

Two attack scenarios are used to demonstrate the extended SFT representations and the use of our XSL translation program to translate from the extended SFT representation to CPN in IDS design. We use two methods to test the correctness of the generated CPN. First, we use the simulation tool of CPN/design to find if tokens are generated as expected. If all conditions are satisfied, an intrusion alarm token should be generated in the CPN. Second, we use our CPN to software agents compiler to convert generated CPN to software agents, and test the functionality of these software agents to decide whether the generated CPN templates are correct or not [74].

B.1 FTP bounce attack

The FTP bounce attack is a well-known distributed attack. This attack exploits a weakness in certain FTP daemons and a trust relationship between two hosts. The FTP bounce attack is a good candidate for testing as it is a distributed attack involving three machines (attacker, relay and victim) as shown in Figure B.1. The relay machine has a vulnerability of enabling PORT commands regardless of source or destination. The victim machine trusts the relay machine and runs an RSH server.

A detailed explanation of this attack is provided in [8]. The basic steps involved are:

1. Create a malicious file containing a valid remote shell (RSH) message.

2. Identify a vulnerable FTP server (relay host) and a host which trusts the relay host and is running a RSH daemon (victim).
3. Upload the file to the relay host.

4. Redirect the output of the FTP server to the RSH port of victim machine using the PORT command.

5. Download the file directly to the RSH port of the victim.

6. The victim accepts the file as a command to be run as root user.

Figure A.1 shows the extended SFT for FTP Bounce Attack. As depicted in Figure A.1, first, a FTP PORT OK event must happen immediately after an FTP PORT event between 'src_host' and 'dst_host'; second, a FTP RETR event must happen; third, a TCP connection from the FTP data port of 'src_host' to the RSH port of 'vic_host' must happen; fourth, immediately after this, a FTP RETR OK event must happen and the FTP RETR OK event must occur after the TCP connection from FTP data port to RSH port event for a FTP Bounce Attack to happen.

The CPN diagram we get using the translator displayed by the Design/CPN tool is given in Figure B.2. As can be seen from the CPN diagrams, five events (FTP PORT request, FTP PORT success, FTP retrieve request, FTP retrieve success and RSH connection from FTP) happening according to some order constitutes positive identification of the FTP bounce attack.

We first use automatic simulation provided in the design/CPN tool to test the generated CPN. When the arcs from the token source places are enabled in order, a token is generated.
Figure B.2 Generated CPN for FTP bounce attack
in the FTP\_BOUNCE\_ATTACK place as expected.

We also verify the correctness of the CPN through testing the detection ability of the intrusion detection software agents generated from the CPN design using the CPN to software agents compiler [74]. We set up a vulnerable FTP Server on one monitored host so it can be used as the relay machine for the FTP Bounce Attack and run a RSH server on another monitored host, which trusts the host running the vulnerable FTP server.

At first the attack was run in isolation. The automatically generated software agents were able to detect the attack within 8-10 seconds of its completion. Then an attack script was used to generate two FTP bounce attacks interspersed between 48 normal FTP GET/PUT sequences. The generated agents were able to detect the attacks without giving any false positives or false negatives.

\section*{B.2 Trinoo Distributed Denial of Service attack (DDoS)}

A DDoS attack is a massive, coordinated attack, and normally involves hundreds or even thousands of machines that simultaneously attack the victim. In a DDoS attack, there are one or more master control programs and many daemons controlled by the masters. When masters issue attack commands to daemons, the daemons launch simultaneous attacks, such as flooding the victim [7]. This will disrupt normal operation of the victim or even bring down the victim.

The Trinoo DDoS attack [15] is used as a test case in our experiment. Trinoo DDoS attacks normally consist of two steps. First, a Trinoo network is created; composed of one or several masters and many daemons controlled by the masters. Second, the attack is launched by issuing an attack command to the master servers from the attacker. After getting the attack command from the attacker, the masters will command all the daemons to send packets to the victim machine.

Communication from Trinoo masters to daemons is via 27444/udp, and from Trinoo daemons to masters is via 31335/udp by default. When a daemon starts, it initially sends a "HELLO" command to the master. Masters maintain a list of active daemons controlled by
them and use a keep-alive procedure to find out which daemons are alive. When a Trinoo master sends a 'png' command to a daemon it controls, the daemon needs to reply to the master by sending the string 'pong' to indicate it is alive.

An attacker remotely controls Trinoo masters via TCP connections to port 27665/tcp (by default) on the masters' hosts. To start up a DDoS attack, an attacker need to connect to this port and issue commands. The attacker can issue a 'mtimer N.seconds' command to set the attack time and then issue 'dos victim_ip' command to begin the DDoS attack. When masters receive this DoS attack command, they issue 'aaa password victim_ip' command to all daemons they control to launch DoS attacks to the victim machine.

The detailed explanation of the Trinoo attack can be found in [15]. Figure B.4 illustrates the SFT for the Trinoo DDoS, and Figure B.5 is the CPN translated from the SFT model using the SFT to CPN XSL translator.

We first use an interactive simulation provided in the design/CPN tool to test the generated CPN. When the attack is simulated by changing the markings of token source places, transitions are fired in order and one token is generated in the id_TRINOO_ATTACK_p place on the root of the CPN.
We also set up an experiment to test the intrusion detection software agents being automatically generated from the CPN. We use the Trinoo DDoS attack described above as a test case. The generated software agents were able to detect the attacks without giving any false positives or false negatives.
Figure B.5 Generated CPN for TRINOO DDOS attack
BIBLIOGRAPHY


